САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СПИСОК-2012

МАТЕРИАЛЫ ВСЕРОССИЙСКОЙ НАУЧНОЙ КОНФЕРЕНЦИИ ПО ПРОБЛЕМАМ ИНФОРМАТИКИ

25–27 апреля 2012 г.,
Санкт-Петербург

Санкт-Петербург
ВВМ
2012

УДК 004(063)
С72

Печатается по рекомендации
кафедры системного программирования
Санкт-Петербургского государственного университета

С72 Список-2012: Материалы всероссийской научной конференции по проблемам информатики. 25–27 апр. 2012 г., Санкт-Петербург. — СПб.: ВВМ, 2012. — 520 с.

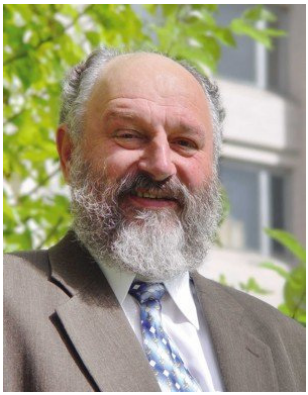
ISBN 978-5-9651-0686-8

Тематика сборника затрагивает широкий круг актуальных проблем теоретической и прикладной математики и информатики. Слово «СПИСОК», ставшее названием конференции, — это не только обозначение фундаментальной структуры данных, но и сокращение от названий трех направлений исследований: Системное Программирование, Интеллектуальные Системы, Обеспечение Качества. Результаты исследований в этих областях являются значительной частью того знания, которое в настоящее время можно назвать словом «информатика».

Для студентов и аспирантов естественно-научных специальностей.

ISBN 978-5-9651-0686-8

Системное программирование



Терехов
Андрей Николаевич

председатель программного комитета конференции
д.ф.-м.н., профессор
заведующий кафедрой системного программирования СПбГУ
директор НИИ Информационных технологий СПбГУ
генеральный директор ЗАО Ланит-Терком

ПОСТРОЕНИЕ ТРЁХМЕРНОЙ ГЕОМЕТРИЧЕСКОЙ МОДЕЛИ ПО НАБОРУ МРТ СНИМКОВ

С. Бояровски

студент 461 группы, Математико-Механический факультет, stefan.bojarovski@gmail.com

А. Петров

аспирант кафедры Системного программирования, Математико-Механический факультет, sanya.petrov@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В компьютерном моделировании биомеханических процессов очень часто используется метод конечных элементов, в основе которого лежит правильная подборка геометрии модели. Для численного анализа сложных анатомических структур требуется геометрическая модель внутренних органов.

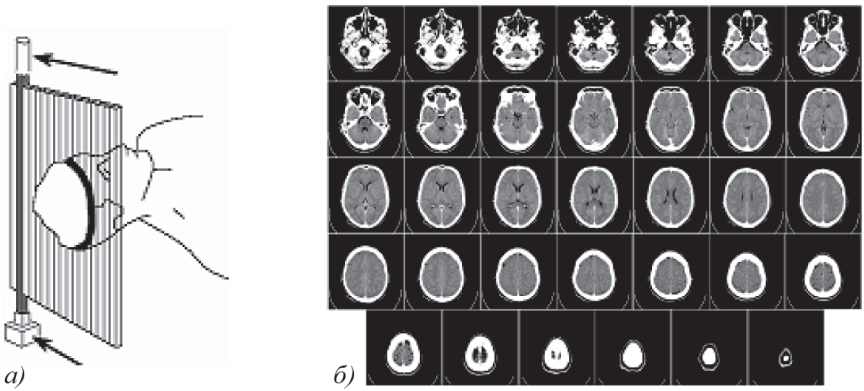
В данном докладе рассматривается выбранный нами подход к построению геометрической модели тела человека по заданному набору магнитно-резонансных томографических снимков. Также рассматриваем преимущества, которые даёт нам геометрическая модель при исследовании данных пациента.

Введение

В медицине, в медицинских исследованиях, при диагностировании болезней всё чаще применяются новые технологии — компьютерное моделирование, анализ и обработка данных с помощью методов компьютерного зрения. Наша цель в курсе «Алгоритмы 3Д в медицине» — ознакомиться с проблемами в медицине, которые компьютерное зрение и моделирование могут помочь решить, и используя наши знания в этих областях попытаться найти хорошие решения этим проблемам.

Одним из главных шагов при решении многих из наших задач — это получение геометрической модели рассматриваемого объекта. С помощью геометрической модели становится проще визуализировать объекты сложной структуры и выполнять над ними большое количество полезных операции. Особенно это оказывается полезным при работе с томографическими снимками.

Томографические изображения — это определённое количество поперечных снимков, которые содержат информацию о внутренней структуре объекта. Обычно это плотность тканей закодирована в виде интенсивности на чёрно-белом двумерном изображении. С помощью алгоритмов компьютерного зрения мы восстанавливаем трёхмерную геометрию всех объектов, которые отображены на снимках.



а) *б)*

Рис. 1. Пример применения томографии в медицине:

а — принцип работы томографии; *б* — результат томографической снимки мозга

Процесс построения трёхмерной геометрии состоит из нескольких независимых функций, которые можно применять по разному порядку для получения желаемых результатов. Эти функции разделены по нескольким модулям, в зависимости от задач которые они выполняют: ввод/вывод данных, фильтрация изображений, сегментация и построение трёхмерной маски и обработка геометрической модели.

Алгоритм фильтрации на основе шкалы Хаунсфилда

Шкала Хаунсфилда

Шкала Хаунсфилда — это количественная шкала рентгеновской плотности (радиоденсивности). Шкала единиц Хаунсфилда (денситометрических показателей, англ. *HU*) — это шкала линейного ослабления излучения по отношению к дистиллированной воде, рентгеновская плотность которой была принята за 0 HU (при стандартных давлении и температуре). Величина HU для материала X с линейным коэффициентом ослабления μ_X вычисляется по следующей формуле:

$$\frac{\mu_X - \mu_{\text{water}}}{\mu_{\text{water}} - \mu_{\text{air}}} \times 1000, \qquad (1)$$

где μ_{water} и μ_{air} — линейные коэффициенты ослабления для воды и воздуха при стандартных условиях. Стандарты, указанные выше, были выбраны для практического применения в компьютерной томографии живых организмов (в том числе человека), т. к. их анатомические структуры в значительной степени состоят из связанной воды.

Для веществ и органов, которые обычно встречаются в человеческом теле, вычислены следующие средние денситометрические показатели.

Таблица 1

Значения величины Хаунсфилда для тканей в человеческом теле

Вещество	HU
Воздух	−1000
Легкие	−700
Мягкие ткани	с −300 по –100
Жир	−84
Вода	0
Кровь	с +30 по +45
Мышца	+40
Кость	с +700 (губчатая кость) по +3000 (плотная кость)

Эта шкала позволяет нам выделить интересующие нас ткани на изображении и применять наши алгоритмы сегментации и построения геометрии только для определённого вида органов, что является очень полезной возможностью.

Фильтрация по шкале Хаунсфилда

Каждое устройство, которое создаёт томографические снимки, использует собственные параметры при обработке и записи данных изображения. По стандарту DICOM, все эти параметры должны быть записаны в заголовке каждого файла из набора, по определённому порядку. Это очень важно для сохранения целостности информации и избежания ошибок при работе со снимками на разных платформах.

Для восстановления значения величины Хаунсфилда для данного пикселя используется следующая формула:

$$HU = \text{Pixel Intensity} * \text{Rescale Slope} + \text{Rescale Intercept}. \quad (2)$$

Параметры Rescale Intercept и Rescale Slope зависят от устройства, с помощью которого были сняты изображения, и находятся в метаданных набора по адресу (0028,1052) и (0028,1053) соответственно. Pixel Intensity — это значение интенсивности пикселя.

С помощью этой формулы, легко можно отфильтровать изображения, удалив все значения пикселей, которые отличаются от интересующего нас диапазона на больше чем заданного порога. Таким образом, можно сразу выделить все пикселы в изображении, на которых, по шкале Хаунсфилда отображены кости.

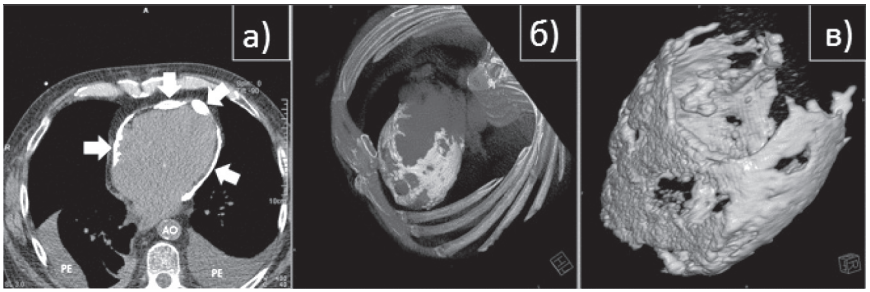


Рис. 2. Применение шкалы Хаунсфилда
при визуализации МРТ данных — объёмный рендеринг:

а — сердце человека на томографическом снимке; *б* — сердце человека в объёмном рендеринге; *в* — выделение слоя кальция около сердца

Алгоритм сегментации

Нам была предоставлена реализация алгоритма сегментации, который по заданному изображению строит трёхмерную маску. В маске выделена самая большая связанная область в изображении, пикселы внутри которой имеют значения больше некоторого порога. Другими словами, этот алгоритм выделяет самый большой объект в изображении, не являющийся фоном.

Маска представлена в виде трёхмерного булевого массива, который можно представить себе как облако точек. В зависимости от дальнейшего использования, это облако точек можно либо сразу использовать в таком виде, либо дополнительно обработать, чтобы получить нужное представление геометрии — например, реконструировать внешнюю оболочку, убрать внутренние точки или построить диаграмму Вороного.

Подход к построению геометрии

На основе фильтра Хаунсфилда

После применения фильтра Хаунсфилда, можно сразу применить алгоритм сегментации, и полученную маску уже использовать в дальнейших задачах. Можно сегментировать каждый диапазон на шкале в отдельных подходах, и потом соединить получившиеся результаты.

Однако, при применении фильтра Хаунсфилда, в результате часто появляются артефакты — области малого размера которые имеют такую-же интенсивность как и ткани в заданном диапазоне. Чтобы избежать такие артефакты можно применить фильтр размытия перед фильтрацией по шкале Хаунсфилда.

На основе других фильтров

Если мы хотим держать под контролем процесс сегментации, то можно итеративно, с участием пользователя, сегментировать изображение на основе различных фильтров.

Одна из особенностей человеческого тела — это вложенность областей разных органов на томографических снимках. Этот факт можно использовать для построения подхода итеративной сегментации. В таком подходе, для каждой выделенной области «рекурсивным образом» можно применять фильтр для удаления фона, сегментацию и разбиение полученной маски на несвязанные области. Этот процесс можно повторять до получения областей размера которых меньше некоторого порога.

Сравнение трёхмерной геометрии с другими средствами визуализации

Ниже мы рассматриваем три из самых часто употребляемых способов визуализации данных, и попытаемся пояснить их преимущества и недостатки:

- Мозаика — самый простой способ визуализации томографических данных. Определённое количество снимков располагается в виде мозаики. Не требует дополнительной обработки данных томографического изображения. Такая визуализация не представляет особое неудобство для специалистов, особенно для выполнения «простых задач» — обнаружение посторонних объектов там где их не должно быть, или механические повреждения некоторых органов.

Недостатком этого способа является то, что специалист получает косвенное представление о структуре и геометрической форме трёхмерного объекта. Таким образом, например, задача сегментации печени по схеме Couinaud-a уже представляет собой нетривиальной задачей.

- Объёмный рендеринг — техника, используемая для получения плоского изображения трехмерного дискретного набора данных. Преимущество объёмного рендеринга перед мозаикой — это возможность трёхмерной визуализации. Входными данными является регулярная сетка вокселов, где каждому вокселу соответствует усредненное значение (температура, плотность материала) в данной точке трехмерного объекта. Прямой объемный рендерер сопоставляет значению каждого воксела цвет и прозрачность. Это делается при помощи передаточной функции, которая может задаваться кусочно-линейной функцией или таблицей значений. После этого полученное RGBA значение выводится в кадровый буфер. После прорисовки всего объема получается цельная картинка.

Объёмный рендеринг — это очень сложная вычислительная задача, которая требует дополнительных вычислений. Хотя и получается хорошая

визуализация, на этом и заканчивают его возможности, так как в результате обычно не строится геометрическая модель.

- Трёхмерная геометрическая модель — предоставляет все возможности трёхмерной визуализации как и объёмный рендеринг, но при этом даёт ещё возможность использования полученной модели в последующих задачах, например в компьютерном моделировании, в анализе методом конечных элементов и применять над моделью множество полезных операции.

Кроме как для реконструкции анатомических структур, построение геометрической модели нужно для симуляции биомеханических процессов в теле человека. В настоящем времени, одна из самых перспективных и привлекательных задач — это быстрое и точное вычисление механического поведения мягкой ткани. Эта задача является предпосылкой для обширной и по большей части неисследованной области виртуальной и симулированной хирургии.

Заключение

В данном докладе был сделан обзор выбранного нами подхода к построению трёхмерной геометрической модели на основе заданного томографического изображения. Был рассмотрен алгоритм сегментации с помощью шкалы Хаунсфилда, и использование фильтров в сочетании с алгоритмом сегментации с целью построения геометрической модели. Были указаны преимущества которые даёт геометрическая модель при визуализации медицинских данных и при компьютерном моделировании в медицине.

Литература

1. *Bryan P*. Bergeron and Robert A. Greenes, Modeling and Simulation in Medicine: The State of the Art, Proc Annu Symp Comput Appl Med Care. 1988 November 9: 282–286.
2. Introduction to CT physics — elsevierhealth.com
3. *David W. Fanning*. Converting CT Data to Hounsfield Units.
4. *Markus H. Gross*. Computer Graphics in Medicine: From Visualization to Surgery Simulation, Vol. 32 No.1 February 1998 ACM SIGGRAPH.
5. *A. Kaufman D. Cohen* and *R. Yagel*. Volume Graphics, IEEE Computer. Vol. 26. No. 7, July 1993. Pp. 51–64.
6. *D. Epstein*. Geometry in Action, a collection of applications of computational geometry, Theory Group, ICS, UC Irvine.

ПОДХОДЫ К РЕКОМЕНДАЦИИ ТРЕКОВ В СОЦИАЛЬНЫХ СЕТЯХ

А. А. Дзюба

студент кафедры системного программирования,
alexandr.dzuba@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В данной статье рассмотрены различные аспекты применения алгоритма *Random Walk with Restarts* в *рекомендательных системах* для социальных сетей. Он основан на включении в процесс рекомендации различных отношений между объектами сети и позволяет достичь более релевантных результатов, чем традиционные алгоритмы, основанные на *коллаборативной фильтрации*.

Введение

Большинство социальных сетей позволяет пользователям публиковать различные объекты: видео, треки, документы. Некоторые, например Last.fm, предоставляют пользователю доступ к уже существующему контенту. В том и другом случае количество подобных объектов настолько велико, что сложно найти среди них интересующую информацию путем обычного просмотра. *Рекомендательные системы* решают эту задачу, предсказывая предпочтительность объекта для конкретного пользователя, основывая свой прогноз на данных, указанных пользователем явно, или собранных из истории его взаимодействия с социальной сетью.

Для составления рекомендаций обычно используются методы *коллаборативной фильтрации* [1]. Когда требуется предсказать, насколько понравится пользователю новый объект, для этого используются оценки пользователей, похожих на данного [2,3]. Как альтернатива коллаборативной фильтрации может быть использован алгоритм *Random Walk with Restarts* [4]. Он превосходит стандартные методы фильтрации по качеству, позволяя включать в рекомендацию разнородные знания из социальной сети. При этом данный метод универсален, он пригоден для работы с объектами любого типа: треки [4], фотографии [5]. К недостаткам алгоритма Random Walk with Restarts (RWR) можно отнести его вычислительную сложность.

В этой работе исследована применимость указанного алгоритма в крупных социальных сетях для рекомендации музыкальных треков. Вместе с тем большинство утверждений относительно алгоритма универсальны.

- Предложены модификации оригинального алгоритма, позволяющие улучшить качество рекомендаций.

- Исследованы способы включения в RWR различной информации из социальной сети и её влияние на скорость и качество работы алгоритма.
- Оригинальный алгоритм при работе использует структуры данных, которые приобретают колоссальные размеры в реальных социальных сетях. Представлен «облегченный» вариант алгоритма и описаны качество и время его работы.

Описание алгоритма

В качестве главного объекта исследования выбран алгоритм Random Walk with Restarts [4]. Он позволяет давать рекомендации на основе разнородных данных социальной сети, таких как история прослушивания треков, социальные связи, схожесть музыкальных вкусов, теги, которыми пользователи помечают треки, и многое другое.

Эти данные представлены в алгоритме в виде графа отношений между объектами. В простейшем случае такими объектами являются пользователи (U) и треки (Tr). Они и связи между ними являются вершинами и дугами графа соответственно, веса дуг определяются силой связанности.

Начиная в вершине *x,* RWR на каждом шаге выполняет случайный переход от текущей вершины к новой по инцидентным дугам, причем вероятность перехода прямо пропорциональна весу дуги. При этом каждый раз с фиксированной вероятностью *a* обход может вернуться в начальную точку *x*. Пусть $\mathbf{p}^{(t)}$ вектор-столбец, в котором $\mathbf{p}_{\square}^{\text{кф}}$ означает вероятность того, что на шаге *t* RWR находится в вершине *i*. Пусть **q** вектор-столбец, в котором $q_x = 1$, а остальные нули, и **S** — нормализованная по столбцам матрица смежности графа. Тогда стационарное состояние алгоритма может быть получено применением (1) до сходимости:

$$\mathbf{p}^{(t+1)} = (1 - a)\,\mathbf{S}\mathbf{p}^{(t)} + a\mathbf{q} \qquad (1)$$

Стабильное состояние даст нам вероятности оказаться в каждой вершине в долгосрочной перспективе. Таким образом $\mathbf{p}_i^{(l)}$, где *l* — номер шаг после достижения сходимости, может быть рассмотрена как мера связанности вершин *i* и *x*. Если в качестве начальной вершины взять пользователя *u*, то её связанность с треками можно считать мерой предпочтимости этих треков.

Способы построения графа социальной сети и их оценка

Как было сказано ранее, в граф социальной сети можно включать информацию из различных источников. Вершинами могут выступать пользователи, треки, теги и другие объекты сети. В оригинальной версии алгоритма [4] лучшие результаты показал социальный граф, включающий пользователей **(U),** треки **(Tr),** теги **(Tg),** и отношения **U-U**, **U-Tr, U-Tg, Tg-Tr**, а также обратные к ним (см. рис. 1).

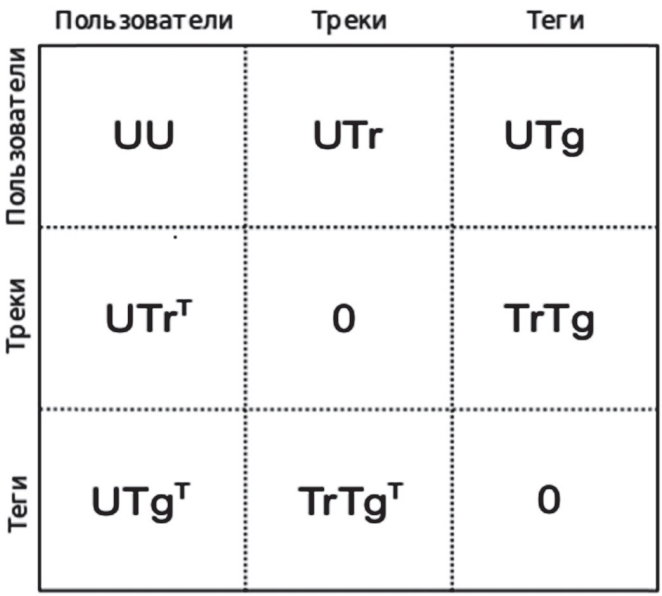


Рис. 1. Матрица смежности социального графа

Такой подход к построению графа дал лучшие результаты, нежели стандартный способ коллаборативной фильтрации.

Включение в граф отношений между объектами социальной сети

Для улучшения качества рекомендаций было проведено исследование способов расчёта весов дуг социального графа. В графе, включающем пользователей и треки, можно выделить 4 отношения: **U-U, U-Tr, Tr-U, Tr-Tr**. Лучший результат показали следующие методы взвешивания дуг этих отношений.

U-Tr, Tr-U. Для оценки веса дуги такого типа использовалось количество прослушиваний пользователем данного трека.

U-U, Tr-Tr. Для веов дуг этого типа были применены методы расчёта схожести между многомерными векторами-рейтингами треков и пользователей. Эти рейтинги представляют собой строки и столбцы матрицы смежности подграфа отношения **U-Tr**.

Расчёт схожести осуществлялся с помощью библиотеки Apache Mahout [7]. Средства этой библиотеки позволяют вычислять схожесть между многомерными векторами согласно различным метрикам, среди которых: косинус угла между векторами, *коэффициент корреляции Пирсона, количество со-*

впадающих элементов и другие. Кроме того, Mahout создан на платформе Hadoop [8], позволяющей производить распределенные расчеты.

В таблице 1 приведены оценки качества разных рекомендаций для набора данных Last.fm[1].

Таблица 1

Среднее значение метрик качества рекомендаций для 3148 пользователей Last.fm

	Базовый метод	*US*	*US + TS*
Mean HL (10)	2,675	3,925	4,150
Mean HL (5)	1,574	2,339	2,500

Для оценки практической применимости системы лучше всего подходит метрика *Half-life Utility* [6], имитирующая радиоактивный распад. Чем дальше релевантный элемент от начала списка рекомендованных элементов, тем меньше его вес. И вес элементов, стоящих друг от друга на расстоянии периода полураспада, отличается в два раза. В данной работе для сравнения рекомендаций главным образом использовалась эта метрика. *Mean HL* — Среднее для всех пользователей значение метрики Half-life Utility с периодом полураспада 10 и 5.

Базовый метод, описанный в [4], оперирует графом из отношений **U-Tr, Tr-U,** и **U-U,** где пользователи связаны знакомством с социальной сети.

В методе *US* в базовый социальный граф вместо социальных связей между пользователями включаются отношения схожести, рассчитанные как косинус угла между векторами-рейтингами.

Метод *US+TS* отличается от предыдущего добавлением в граф схожести между треками. Количество дуг при таком подходе вырастает вдвое (и время расчёта рекомендации растет сопоставимо) однако этот метод обеспечивает наибольшую точность.

Оба оптимизированных метода демонстрируют значительное превосходство над базовым. Метод без схожести треков немного проигрывает лидеру, но он почти в два раза быстрее и для него не нужно высчитывать схожесть треков, что существенно экономит вычислительные ресурсы социальной сети.

Упрощенный вариант алгоритма

Время вычислений рекомендаций и объем используемой памяти линейно зависит от количества дуг в социальном графе. В графах крупных социальных сетей может содержаться информация о миллиардах прослушиваний[2]. Поэтому становится актуальным вопрос упрощения алгоритма.

[1] Набор данных из более 3 тыс. пользователей и 30 тыс. треков [4].

[2] И в ситуации, когда речь идет об отношении миллионов пользователей и миллионов треков — матрица действительно получается сильно разреженная.

В качестве такого упрощения рассмотрены рекомендации на основе персональных подграфов. Если в полном варианте алгоритма весь социальный граф, будучи однажды построенным, использовался для рекомендаций каждому пользователю, то упрощенный алгоритм конструирует подграф по запросу пользователя, включая в него только наиболее близкие вершины.

Схема построения подграфа

G = (V, E), где *V* — множество вершин, *E* — множество дуг.

1) в *U* добавляется n_u пользователей, наиболее похожих на данного;
2) в *Tr* добавляется n_{tr} треков, наиболее предпочтительных для данного пользователя;
3) для каждого пользователя, добавленного на 1 шаге, в *Tr* добавляется m_u его любимых треков;
4) для каждого трека, добавленного на 2 шаге, в *Tr* добавляется m_{tr} его любимых треков;
5) *V = (U U Tr)*
6) *E* формируется из дуг, у которых обе инцидентные вершины принадлежат *V*.

Алгоритм рекомендации для персонального подграфа аналогичен полному варианту. На шаге 6 могут добавляться не все дуги, а фиксированное количество, зависящее от конкретной социальной сети и плотности дуг в графе.

Размеры персонального подграфа

Размеры подграфа регулируются константами *n* и *m* в описании алгоритма построения. Увеличение числа вершин и связей не всегда ведет к улучшению качества рекомендаций, поэтому для применения предложенного алгоритма требуется исследование этих величин.

На рис. 2 показана зависимость метрики *Mean Half-life (10)* от $n=n_u=n_{tr}$ для фиксированного $m=m_u=m_{tr}$. Каждый график соответствует определенному *m* в диапазоне 10–50.

У показанных функций есть максимумы, и чем больше m, тем больше этот максимум. При увеличении *m* от 10 до 25 рост максимумов функций ощутим, а затем он снижается настолько, что дальнейшее увеличение *m* лишь увеличивает время работы алгоритма за счет включения большего числа вершин. График функции при этом поворачивается по часовой стрелке.

Среди экспериментов, в которых значение контрольной метрики получилось наилучшим, самым выгодным является построение персонального подграфа при $n=5$, $m=26$. Это означает, что в рекомендации учитываются предпочтения 6 наиболее близких друзей и треки, похожие на 6 любимых у пользователя. При таких пропорциях значение метрики *Mean Half-life (10)*

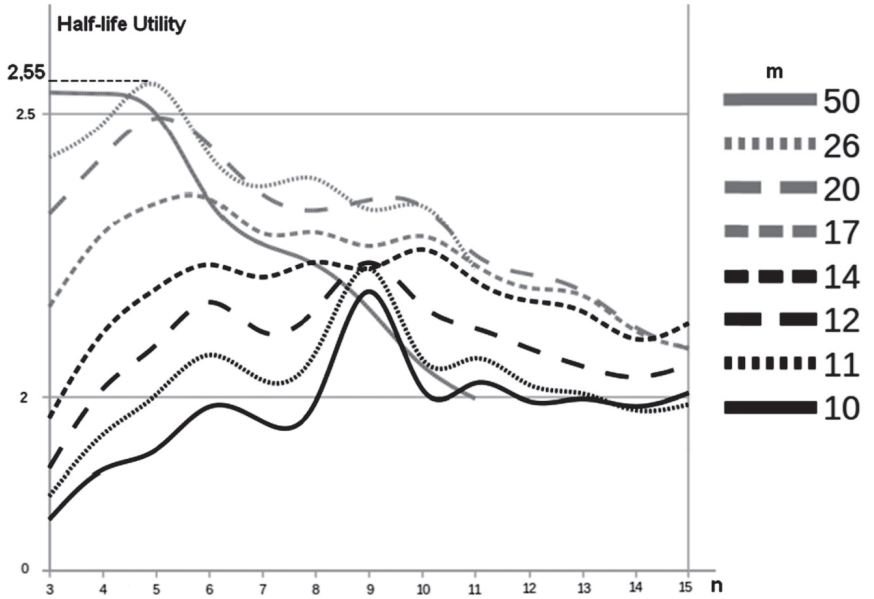


Рис. 2. Зависимости качества алгоритма от параметров построения графа

равно 2.55, что сравнимо с результатами базового алгоритма. Время составления графа и выполнения алгоритма RWR мало настолько, что их можно осуществлять сразу после запроса пользователя на рекомендацию.

Похожее исследование несложно провести для любого набора данных социальной сети и выяснить наилучшие пропорции конструирования персональных подграфов.

Заключение

В данной статье описаны различные аспекты применения алгоритма Random Walk with Restarts. Описан способ, значительно улучшающий качество алгоритма с помощью включения в него отношений схожести между различными пользователями и между различными треками. Для больших сетей, социальный граф которых может быть чрезмерно велик, предложен способ построения персональных подграфов, дающих рекомендации лишь одному пользователю. Такой подход делает время расчёта небольшим даже для крупных социальных сетей.

Литература

1. *Badrul Sarwar, George Karypis, Joseph Konstan, John Riedl.* Item-based collaborative filtering recommendation algorithms // *Proceedings of the 10th international conference on World Wide Web* (WWW «01). 2001. С. 285–295.
2. Netflix Prize — http://www.netflixprize.com/
3. *Takacs, Gabor and Pilaszy, Istvan and Nemeth, Bottyan and Tikk, Domonkos.* Matrix factorization and neighbor based algorithms for the netflix prize problem // *Proceedings of the 2008 ACM conference on Recommender systems* (RecSys «08). 2009. С. 267–274.
4. *Konstas, Ioannis and Stathopoulos, Vassilios and Jose, Joemon M.* On social networks and collaborative recommendation // *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (SIGIR «09). 2009. С. 195–202.
5. *Ivan Cantador, Ioannis Konstas, and Joemon M. Jose.* Categorising social tags to improve folksonomy-based recommendations. // *Web Semant.* 9, 1. 2011. С. 1–15.
6. *Herlocker, Jonathan L. and Konstan, Joseph A. and Terveen, Loren G.* and *Riedl, John T.* Evaluating collaborative filtering recommender systems // *ACM Trans. Inf. Syst.* 22, 1. 2004. С. 5–53.
7. Apache Mahout — http://mahout.apache.org/
8. Apache Hadoop — http://hadoop.apache.org/

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМОВ ОБРАБОТКИ ИЗОБРАЖЕНИЙ С АНАТОМИЧЕСКИМИ ОГРАНИЧЕНИЯМИ НА ОСНОВЕ HTML 5.0

Добролеж А. Б.

студентка 5 курса кафедры системного программирования,
abdobrolezh@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Выполненная работа посвящена демонстрации результата пластической операции. Реализованное приложение позволяет редактировать изображение лица человека в web интерфейсе, а именно — разглаживать морщины, исправлять контур лица и носа.

Введение

В 21 веке все медицинские процессы ускоряются. К примеру, ЭКГ можно снять дома, просто приложив к себе IPhone с датчиком и перекинув данные врачу за считанные секунды. Устройства, с помощью которых производится диагностирование и мониторинг, уменьшаются в размере, цены на диагностирование и прогнозирование падают.

Многое из вышесказанного верно и про пластическую хирургию. Компьютерное прогнозирование операций в этой области позволяет наглядно показать клиенту эффективность принятых мер, а так же дать ему почувствовать эффект от хирургического вмешательства непосредственно ДО самого вмешательства.Создание редакторов 2D изображений для различных косметических процедур и пластической хирургии являются интересной областью разработки. Особенностью этих редакторов является возможность добавления информации об анатомии лица человека, такой как различные фильтры, маски, выявленные алгоритмы старения, ограничения на цвет и форму различных областей изображения. В качестве одного из примеров таких редакторов можно рассмотреть приложение AlterImage, позволяющее прогнозировать результаты пластических операций. Несмотря на солидный набор функций, приложение обладает очевидным недостатком: оно написано только под платформу Microsoft Windows, и для запуска его на других операционных системах приходится идти на ухищрения, не доступные рядовому пользователю: например, пользователям Mac OS X предлагается установить виртуальную машину VMWare, и пользоваться приложением из-под нее. Кроме того, постоянное обновление версии программы требует дополнительных усилий системных администраторов.

В данной работе основными задачами являются косметические операции по удалению морщин и подтяжке овала лица (изменение контура), а так же подготовка framework»а для легкого расширения на новые хирургические операции и части тела.

В работе было сделано обобщение некоторых достижений в прогнозировании операций и портирование их на современную площадку. Несмотря на то, что на данных момент уже существуют средства и инструменты, позволяющие прогнозировать и демонстрировать клиентам результаты таких пластических операций, как «подтяжка лица» и морщин (например, рассмотренный выше AlterImage), все эти инструменты реализованы под специфические платформы и зачастую требуют вмешательства квалифицированных специалистов для их изначальной установки и настройки. Эти средства привередливы к программному обеспечению и рабочим станциям, на которых они исполняются, что зачастую приводит к необходимости покупки новых компьютеров в угоду требованиям программ. Разработка новых версий этих приложений под другие платформы трудоемка, не говоря уже об адаптации этих средств под мобильные платформы и устройства.

Целью моей работы было создать средство, которое бы было лишено этих недостатков, было бы легко расширяемым и которое было бы разработано с акцентом на развитие современных технологий обработки данных. Получившийся проект работает на практически любой операционной системе, будь то популярная Microsoft Windows, редкий Mac OS X или свободно поставляющийся Linux. Такая кроссплатформенность делает его удобным в использовании и практически не накладывает ограничений на рабочие станции клиентов. В связи с бурным развитием мобильных технологий и широкого в них проникновения HTML5 можно ожидать, что в скором будущем эта программа сможет быть успешно запущена на планшетных компьютерах, что, безусловно, сделает его исключительно удобным в практике использования практикующими врачами.

Решение

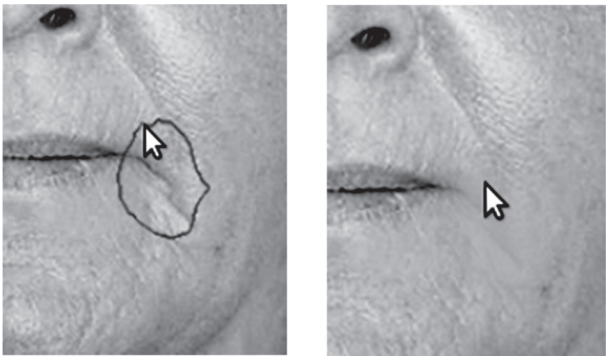
Отдельное внимание было уделено быстродействию приложения: создание интуитивно понятного интерфейса исключительно важно для комфортной работы врача с приложением.

Для работы с цветом кожи и сглаживанием морщин было принято решение перейти от традиционной цветовой схемы RGB на CIELAB, который реализует линейность изменения цветов относительно человеческого восприятия.

В результате на данный момент реализовано приложение на базе HTML5.0. Приложение было протестировано в браузерах Chrome на системах Windows, Linux, Mac OS X и доказало свою работоспособность. Приложение имеет функциональность специализированного графического редактора, предназначенного для медицинских работников. Загрузив в него любую фото-

графию лица человека со своего рабочего компьютера, пользователь программы может обработать изображение с помощью набора индивидуально разработанных графических средств. На данный момент средства включают в себя:

1) Инструмент «сглаживание». Инструмент предназначен для выделения некоторой области изображения и сглаживания морщин, попавших в эту область. После активации инструмента, пользователь может выделить область, после чего к ней будет применен специально разработанный алгоритм по сглаживанию морщин. Стоит отдельно отметить, что этот алгоритм учитывает анатомические особенности кожи лица, а именно ее цветовую гамму. Кроме того, этот же алгоритм используется в инструменте «сглаживание морщин». Он предназначен для разглаживания небольших участков кожи, и он применяет этот алгоритм к динамически создаваемому круглому выделению.



2) Инструмент «подтяжка». Инструмент предназначен для изменения овала лица, подтяжки щек. После выбора инструмента пользователь может исправить элементы лица. При этом основная работа ложится на плечи эвристического алгоритма, который был оптимизирован для успешной работы на фотографиях лиц людей.

3) Инструмент «кроппинг». Классический инструмент, позволяющий убрать лишнюю рамку у изображения для точного редактирования, проведя так называемое «кадрирование». После выбора инструмента появляется рамка, которую можно передвигать и по которой в последствии можно обрезать изображение. Инструмент полезен в связке с инструментом «зеркалирование», что позволяет восстанавливать симметрию лица.

Заключение

Основные задачи — т.е. удаление морщин и подтяжка овала лица на данный момент реализованы в приложении и показывают достаточно работоспособные результаты на тестовой подборке фотографий.

Но несмотря на неплохую работоспособность, показанную в тестах, приложение на данный момент обладает некоторым количеством недоработок. Основным из них является небольшое количество инструментов для работы с изображениями, а так же недостаточное количество анатомических ограниченией и фильтров. Наполнения приложения дополнительными возможностями является нашей приоритетной задачей.

Литература

1. 2D Image Canvas — http://www.w3.org/TR/2dcontext/
2. JCrop — http://deepliquid.com/content/Jcrop.html
3. Сайт по ECMAScript — http://www.ecmascript.org
4. CIELAB — http://www.fho-emden.de/~hoffmann/cielab03022003.pdf
5. Pixastic — http://pixastic.com/

BM3D И ВАРИАЦИОННЫЙ ПОДХОД В ЗАДАЧЕ ПОВЫШЕНИЯ РАЗРЕШЕНИЯ ИЗОБРАЖЕНИЯ

Ю. А. Землянский

кафедра системного программирования, математико-механический факультет; yuri.zemlyanskiy@gmail.com

В. В. Пасиченко

кафедра информатики, математико-механический факультет; victor.passichenko@gmail.com

А. Т. Вахитов

кандидат физико-математических наук, кафедра системного программирования, математико-механический факультет

Санкт-Петербургский государственный университет

Аннотация: Работа посвящена методам super-resolution»а — алгоритмическим способам получения изображения высокого разрешения (HR) из одного или нескольких изображений низкого разрешения (LR). Задача формулируется как задача оптимизации. В процессе решения используется BM3D — современный алгоритм, для фильтрации шума и блюра в изображениях. Реализованый алгоритм сравним по качеству работы с текущими state-of-art решениями.

Введение

Качество изображения определяется количеством пикселей на единицу площади. Прямой подход — увеличение количества сенсоров в светочувствительной матрице в камере — увеличивает стоимость оборудования и повышает уровень шума в изображении. Кроме этого вышеупомянутый подход не всегда возможен. Альтернатива — алгоритмический подход к задаче повышения качества изображения — superresolution.

Алгоритмы superresolution»а могут применяться для цифрового зума, увеличения качества камер слежения (например, чтобы различать номера машин), в современных смартфонах.

Постановка задачи

Задача superresolution формулируется как стадартная обратная задача: LR изображения получается из HR изображением применением линейного оператора (блюр, геометрический сдвиг, downsampling) и добавлением шума, требуется восстановить исходное изображение.

Идеи, лежащие в основе алгоритма

Алгоритм одновременно восстанавливает изображение высокого разрешения и его спектр (например, по базису DCT, FFT). При этом допускает неполное их соответствие, засчет чего снижается влияние шума в исходных изображениях. Совместно минимизируются два функционала. Первый отвечает за то, насколько изображение высокого разрешения соответствует изображениям низкого разрешения. Второй — насколько изображение соответствует реальности. Согласно подходу compressive sensing, это определяется l_0 нормой спектра изображения.

Такая формулировка задачи впервые использовалась для задачи debluring»а в статье [1].

Итеративный алгоритм, минимизирующий эти функционалы использует BM3D фильтр, описанный в статье [2].

Дополнительно, для случая upsamling»а (увеличение разрешения одного изображения) получается добавить в алгоритм стохастическую оптимизацию, что увеличивает скорость и качество работы алгоритма.

Реализация

Исходный алгоритм реализован на базе программного пакета Matlab, который предоставляет обширную библиотку для работы с изображениями.

Заключение

В рамках существующей работы предложен и реализован superresolution алгоритм. На наборе тестовых данных алгоритм показывает лучшее качество работы, чем существующие state-of-art алгоритмы.

В качестве дальнейшей работы планируется провести blind эксперименты (когда линейные пребразования. которые используются для получения LR изображений неизвестны), использовать алгоритм для video superresolution»а.

Литература

1. *A. Danielyan, V. Katkovnik, K. Egiazarian*. BM3D Frames and Variational Image Deblurring.
2. *K. Dabov, A. Foi, V. Katkovnik, K. Egiazarian*. Image denoising by sparse 3D transform-domain collaborative ltering.

ПОДДЕРЖКА МЕХАНИЗМА РЕФАКТОРИНГОВ В METACASE-СИСТЕМЕ QREAL

А. С. Кузенкова

студентка 3 курса кафедры Системного программирования;
kuzenkovanastya@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В статье рассматриваются вопросы применимости рефакторингов в области модельно-ориентированной разработки ПО. Приводится анализ инструментариев, в которых имеется возможность определять рефакторинги моделей. Описывается механизм задания рефакторингов моделей, реализованный в metaCASE-системе QReal в виде прототипа, предоставляющего возможность задавать рефакторинги для различных языков среды и применять их к моделям на этих языках.

Введение

Рефакторинг — широко известный способ повысить качество программного обеспечения. Рефакторинг определяют как изменение внутренней структуры системы с целью сделать её проще для понимания и внесения дальнейших изменений, при этом не изменяя существующей функциональности. При программировании с помощью текстовых языков рефакторинг активно применяется уже много лет и является хорошо изученной областью знаний [1].

В последнее время становится популярным модельно-ориентированный подход (Model-driven engineering, MDE) к разработке ПО. В соответствии с ним создаваемая система представляется в виде набора моделей, описывающих её с различных точек зрения, и впоследствии по ним генерируется (частично или полностью) исходный код системы. Особенностью данного подхода является то, что разработчик большую часть времени работает не с кодом, а с моделями. В связи с этим возникает потребность в рассмотрении понятия рефакторинга и для визуальных моделей тоже.

Рефакторинг на уровне моделей является довольно молодым направлением исследований. Многие вопросы в этой области остаются открытыми для экспериментов и дальнейшего изучения.

Основные задачи рефакторинга моделей

Рассмотрим общий сценарий рефакторинга модели. Система описана с помощью нескольких моделей, характеризующих её с разных точек зрения, включая её статическую и динамическую структуру. Часть исходного кода системы генерируется из её визуального представления, а часть, возможно,

пишется вручную. Таким образом, процесс рефакторинга моделей имеет несколько основных шагов [2]:

- внести необходимые изменения в выбранную нами модель;
- синхронизировать измененную модель с её окружением (например, моделями, связанными с данной);
- заново провести генерацию участков кода, связанных с функциональностью, описываемой измененной моделью;
- согласовать рукописный код с полученным после генерации;

Вносить необходимые изменения в модель можно и вручную, однако часто требуется уметь формально описать изменения, которые хочется осуществить, и применить их сразу к нескольким элементам модели или к нескольким моделям в целом. На данный момент не существует универсального способа описания подобных изменений и данный вопрос требует дальнейшего рассмотрения и изучения.

При обеспечении согласованности моделей до и после рефакторинга неизбежно возникает две основные проблемы. Во-первых, при модельно-ориентированном подходе модели являются ключевыми артефактами разработки, по которым генерируются остальные — исходный код, документация, скрипты сборки и прочее. Если изменяется модель системы, то соответствующим образом должны измениться и все остальные зависящие от неё элементы. Если некоторые из них были изменены вручную, то повторная генерация должна учитывать эти изменения. Данная проблема является отдельной областью исследований и выходит за рамки данной работы.

Во-вторых, согласованность моделей может пониматься как семантическая и синтаксическая корректность в том смысле, что модель, подвергнутая рефакторингу, всё ещё соответствует некоторому языку моделирования и удовлетворяет определенным ограничениям, этим языком накладываемым. Эта проблема является одной из ключевых при осуществлении рефакторинга моделей.

Еще одной проблемой в этом направлении является то, что на данный момент не существует однозначного общепринятого определения понятия качества модели [2]. Несмотря на то, что основной целью рефакторинга является повышение качества системы, а из моделей происходит генерация исходного кода системы, модель и код находятся на разных уровнях абстракции, а так как разработка происходит на уровне модели, то качество кода и качество модели не всегда будут иметь взаимно однозначное соответствие. В данном подходе к разработке существенными становятся такие параметры, как удобство визуального восприятия модели, её читабельность, адаптируемость и др. Отталкиваясь от этой мысли, появляется задача выделить класс рефакторингов, которые являются применимыми и на уровне модели, и на уровне кода, и установить между ними соответствие, а также отдельно рассмотреть изменения, которые относятся непосредственно к рефакторингу моделей.

Существующие решения

В настоящее время существует несколько инструментариев, предоставляющих возможность осуществлять интегрированную поддержку рефакторинга моделей. При этом класс моделей, для которых реализована поддержка рефакторинга в данных средствах разработки, весьма ограничен.

Eclipse Modeling Framework

Eclipse Modeling Framework (EMF) предлагает полноценный инструментарий для разработки метамоделей и поддерживает работу с моделями, соответствующими этим метамоделям [3]. Компонент EMF, отвечающий за рефакторинг моделей, называется EMF Refactor. Он активно использует компонент, отвечающий за преобразование моделей, поэтому сначала более подробно рассмотрим как устроено EMF-преобразование модели.

EMF-преобразование модели определяется как преобразование исходной модели в целевую на основе некоторых заранее определенных правил [4]. Каждое правило имеет две основные составные части: LHS (Left-hand side) — левая часть правила и RHS (Right-hand side) — правая часть правила. Соответствия между элементами правой и левой части правила устанавливаются с помощью номеров, которые указываются перед именем класса. LHS описывает предусловие преобразования, а RHS, соответственно, постусловие. Те элементы модели, которые присутствуют в LHS и отображаются в RHS, должны присутствовать в исходной модели и не изменятся в ходе преобразования. Элементы, которые присутствуют в LHS, но отсутствуют в RHS, удаляются в процессе преобразования. А те элементы RHS, с которыми нет соответствия элементов LHS, должны создаваться в целевой модели.

Применимость правил может быть ограничена с помощью описания дополнительных условий — NAC (Negative application condition). NAC описывает условие, при котором правило рефакторинга не применимо. Если задано несколько таких условий, то каждое из них нужно проверить перед проведением преобразования, и если хотя бы одно из них справедливо для данной модели, то преобразование не производится. По сути LHS представляет собой положительное предусловие, а NAC — отрицательное предусловие. Задание NAC является весьма полезным, если преобразование модели зависит от некоторых свойств элементов, не входящих в область преобразования.

EMF Refactor состоит из двух основных модулей. Модуль генерации рефакторинга интегрирован с функциональностью каждого рефакторинга модели (описанного с помощью EMF model transformation). С помощью этого модуля созданный рефакторинг генерируется в специальный плагин среды Eclipse. Компонент EMF Tiger позволяет графически представить диаграмму трансформации модели. Сгенерированный плагин является расширением компонента EMF Refactor, и новый рефакторинг появляется в соответству-

ющем меню системы. Для него также доступны предварительный просмотр целевой модели и отмена внесенных изменений.

Generic Modeling Environment

Generic Modeling Environment (GME) — это среда метамоделирования, базирующаяся на языке UML. Данная среда предоставляет возможности создания предметно-ориентированных моделей для широкого класса систем. The Constraint-Specification Aspect Weaver (C-SAW) — плагин GME, который представляет собой механизм для преобразования моделей [5].

C-SAW переводит исходную модель в целевую при помощи спецификаций преобразования. Данные спецификации определяют, что происходит с каждым конкретным элементов в данном преобразовании. Спецификация состоит из нескольких аспектов и стратегий. Аспекты являются отправной точкой преобразования, а стратегия определяет само преобразование. Аспекты и стратегии описываются с помощью специального встроенного языка ограничений — Embedded Constraint Language (ECL). Описанный механизм также применяется в частном случае преобразования моделей — рефакторинге моделей. Браузер рефакторингов моделей (model refactoring browser) — плагин, встроенный в GME, который в совокупности с C-SAW предоставляет функциональность для работы с рефакторингами моделей. Основной целью этого браузера является предоставление разработчикам интерактивного инструментария для использования и создания рефакторингов моделей. В браузере имеется некоторый заранее определенный набор рефакторингов для визуальных языков общего назначения, пользователю же предоставляется возможность для создания собственных рефакторингов как для языков общего назначения, так и для предметно-ориентированных языков.

Стоит отметить, что главным отличием механизма рефакторингов GME от соответствующего аналога в EMF является то, что правила рефакторинга описываются с помощью текстового языка, а не визуального.

Fujaba

Fujaba — кроссплатформенная CASE-система для модельно-ориентированной разработки программного обеспечения, поддерживающая генерацию в Java [6].

Рефакторинг модели в Fujaba задается в виде кода на языке Java. В частности, это должен быть наследник специального класса AbstractRefactoring, реализующий несколько ключевых методов, таких как checkPrecondition () и perform (). При этом алгоритм рефакторинга можно визуализировать с помощью диаграммы активностей UML, в каждом из элементов-действий которой с помощью диаграммы взаимодействий задаются действия по трансформации графа модели. Для задания LHS и RHS используются стереотипы UML <<create>> и <<destroy>> [6].

Моделирование в Fujaba осуществляется с помощью диаграмм классов и некоторых поведенческих диаграмм UML, поэтому осуществляемые рефакторинги также сильно ориентированы на язык UML (например, инкапсулирование поля или выделение метода класса в отдельный элемент). При этом и язык описания рефакторинга создан с учетом, что будет рефакториться модель, по которой генерируется объектно-ориентированный код на Java — например, такие действия, как проверка наличия метода в классе или установка значения видимости поля вынесены в отдельные элементы языка рефакторинга [7]. На наш взгляд, данный подход сложно применить для более общего случая, когда целевым языком генерации не обязательно является Java или даже любой другой объектно-ориентированный язык. К тому же, получаемые диаграммы, описывающие рефакторинг, получаются довольно громоздкими и сложными для восприятия.

Реализация

Целью данной работы стало создание прототипа механизма рефакторингов в среде визуального программирования QReal, которая разрабатывается на кафедре системного программирования Санкт-Петербургского государственного университета с 2007 года силами студентов и преподавателей кафедры [8]. QReal имеет средства быстрого создания новых графических языков и инструментальных средств для них, благодаря чему на его основе было создано несколько CASE-систем для различных областей.

В качестве языков визуального программирования, на которых будет осуществляться апробация рефакторингов, были выбраны наиболее зрелые языки среды QReal — язык блок-схем и язык программирования роботов Lego. Последний входит в состав среды QReal: Robots [9] и позволяет задавать логику поведения робота Lego Mindstorms NXT 2.0 в виде последовательности управляющих блоков и исполнять программу на роботе, интерпретируя диаграмму и посылая команды роботу через Bluetooth или USB.

Язык задания рефакторингов

При рассмотрении существующих аналогов возникает существенный вопрос — должен ли язык описания рефакторингов быть текстовым или визуальным.

Среди преимуществ текстового языка можно выделить:

- широкие возможности задания рефакторингов — текстовый язык представляет более мощный аппарат для задания правил рефакторинга по сравнению с визуальным языком;
- компактность — как правило, для описания правила рефакторинга на текстовом языке требуется несколько строк кода, тогда как для описания аналогичного правила с помощью визуального языка требуется создать большую подробную модель;

Преимуществами визуального языка являются:

- отсутствие требования специальной подготовки — характерной чертой визуального языка является то, что он, как правило, интуитивно понятен пользователю, либо не требует значительного времени для понимания, как его использовать;
- наглядность — визуальное представления правила рефакторинга позволяет лучше понять, как будет происходить преобразование модели;

Поскольку QReal является средой визуального программирования, достижение наглядности и понятности видятся нам главной целью при разработке всех входящих в неё инструментов.

При задании модели рефакторинга может потребоваться использовать элементы того языка, для которого этот рефакторинг создается. В связи с этим одной из задач в контексте данной работы является создание языка рефакторинга, гибким образом интегрирующего в себя элементы исходного языка.

Элементы языка делятся две основные группы:

- Шаблон задания рефакторинга — элементы, описывающие каркас правила рефакторинга:
  - блок «ДО»;
  - блок «ПОСЛЕ»;
  - связь преобразования.
- Основные элементы языка рефакторинга:
  - элемент;
  - связь;
  - выделенный сегмент.

Блок «ДО» представляет собой предусловие — в нем находится шаблон, который мы хотим изменить. Блок «ПОСЛЕ» содержит постусловие — результат преобразования после применения данного рефакторинга. Элемент «Связь преобразования» — стрелка в направлении от блока «ДО» к блоку «ПОСЛЕ» используется исключительно для наглядности.

Блок «Элемент» обозначает любой элемент исходного языка, а блок «Связь», соответственно, любую связь исходного языка. Перед описанием рефакторинга пользователь может выделить несколько элементов на диаграмме, которые он хочет использовать в правиле как единый блок — «Выделенный сегмент».

При выборе языка, для диаграмм которого хочется создать рефакторинг, описанный выше язык автоматически интегрируется со всем элементами исходного языка и подключается к системе QReal. При этом у каждого элемента вновь созданного языка появляется числовое поле «ID», благодаря которому можно устанавливать отношение между элементами правой и левой части правила: все элементы в левой части правила нумеруются произвольным образом, а в правой части правила элементам (если требуется) указываются соответствующие номера.

Определяя правило языка, имеется возможность использовать существующие значения свойств элементов диаграммы — для этого используется ключевое слово «EXIST». Например, это может быть полезно, если мы хотим заменить элемент, а имя оставить прежним.

Работа с рефакторингами

Рассмотрим процесс работы с инструментами поддержки рефакторингов с QReal.

Предположим, что мы подключили нужный язык для задания рефакторингов и с помощью этого языка создали правило. Далее нужно сохранить наше правило, после этого оно появится в специальном диалоге с рефакторингами, доступными на данный момент в системе. В этом диалоге слева расположен список рефакторингов, а справа соответсвующее ему изображение с диаграммой выделенного правила рефакторинга.

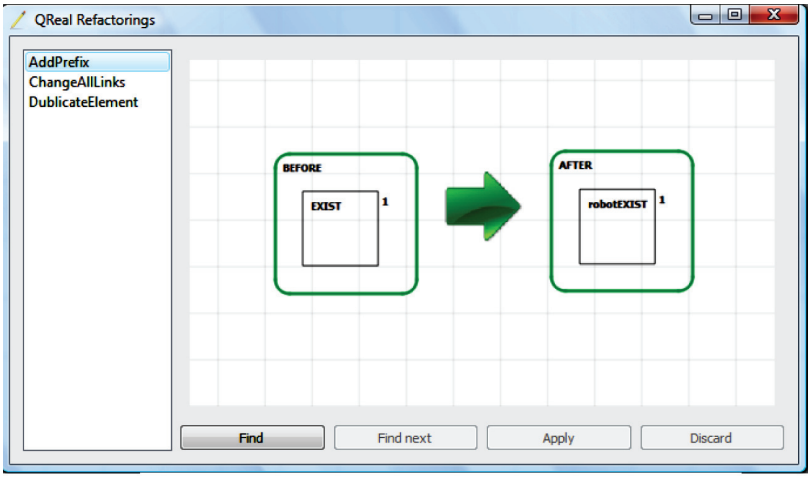


Рис. 1. Диалоговое окно для работы с рефакторингами

В этом же диалоге предоставляются следующие возможности: найти в текущей диаграмме место, где можно применить рефакторинг, далее есть варианты: либо применить в найденном месте выбранное правило рефакторинга, либо найти другое место, где это правило можно применить, либо прекратить поиск. В случае если для текущей диаграммы выбранное правило неприменимо, то пользователю выдается сообщение об этом.

Автоматическое расположение элементов на диаграмме

Описанный выше механизм задания рефакторингов можно рассматривать как перенесение понятия рефакторинга с уровня кода на уровень

модели. Но, как отмечалось выше, иногда можно выделить рефакторинги, которые имеют отношение только к моделям. На наш взгляд, к таким рефакторингам можно отнести автоматическое расположение элементов на диаграмме.

В процессе создания диаграммы пользователь может располагать элементы произвольно, в результате чего диаграммы становятся плохо читаемыми, в результате чего сложно понять изначальную задумку автора. Поэтому полезно, если среда поддерживает возможность автоматически располагать элементы в соответствии с некоторыми формализованными правилами размещения элементов. В связи с этим в системе QReal был реализован механизм автоматического расположения элементов на диаграмме «слева направо», «справа налево», «сверху вниз» и «снизу вверх».

Примеры

Рассмотрим примеры нескольких правил рефакторинга, созданных с помощью разработанного механизма.

1. Изменение имени всех элементов диаграммы: добавление префикса:

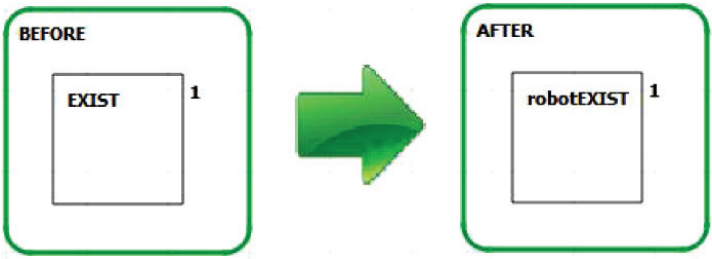


Рис. 2. Добавление префикса

2. Изменение направления всех стрелок на диаграмме:

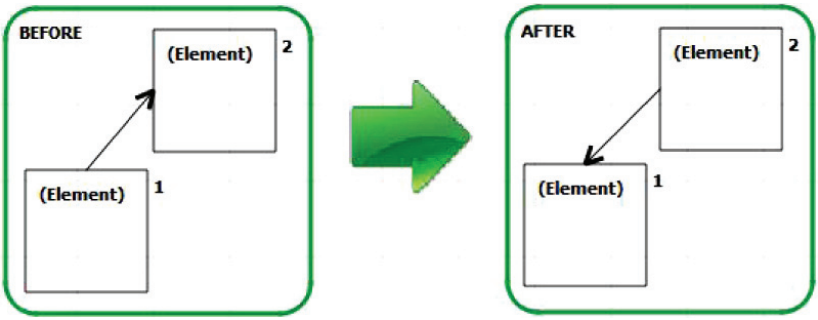


Рис. 3. Смена направления стрелок

3. Автоматическое расположение элементов:

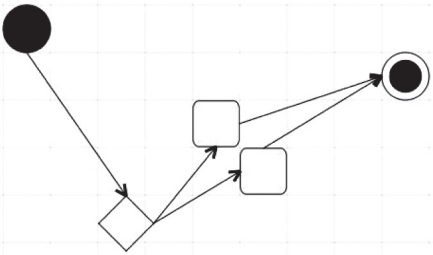


Рис. 4. Диаграмма до применения автоматического расположения

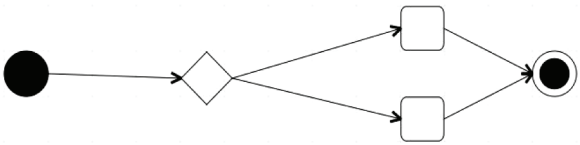


Рис. 5. Диаграмма после применения автоматического расположения

Заключение

Данная работа направлена на исследование вопросов применимости рефакторингов в области модельно-ориентированной разработки ПО. В настоящее время не существует однозначных рекомендаций, которым нужно следовать при решении основных проблем, возникающих в данной области.

В результате проведенного анализа существующих решений было выявлено, что инструментариев, имеющих возможность задавать рефакторинг моделей, немного, и их возможности весьма ограничены. Стоит отметить, что, как правило, это связано с проблемой сохранения целостности модели в результате применения рефакторинга. Естественным является желание иметь универсальный способ задания рефакторингов моделей в рамках MDE-подхода, но для обеспечения целостности модели требуется знание семантики языка. Однако, формальное описание семантики используемых языков не является типичной практикой при модельно-ориентированной разработке ПО, поэтому каждый разработчик средств поддержки рефакторингов моделей вынужден искать пути решения этой проблемы в контексте своей инструментальной среды.

На данный момент основными решениями проблемы обеспечения целостности модели являются либо сильное ограничение на возможности задания рефакторинга, либо перенесение полной ответственности за сохранение целостности на пользователя. В нашем подходе в настоящее время используется последний вариант, но в дальнейших исследованиях планируется предложить другие способы обеспечения целостности модели и реализовать их в среде QReal.

В процессе работы был реализован механизм задания рефакторингов в metaCASE-системе QReal, позволяющий создавать несложные правила рефакторингов и применять им к диаграммам, созданным на основе некоторых визуальных языков среды. Данный механизм дает возможность пользователю быстро вносить большое количество однотипных изменений в диаграмму и тем самым ускорить процесс разработки.

Литература

1. *Martin Fowler*. Refactoring. Improving the Design of Existing Code // Addison-Wesley, 1999.
2. *Tom Mens, Gabriele Taentzer, Dirk Müller*. Challenges in Model Refactoring // Proc. 1st Workshop on Refactoring Tools University of Berlin (2007). Pp.75–81.
3. *Сорокин А. В., Кознов Д. В.* Обзор Eclipse Modeling Project // Системное программирование. Вып. 5. СПб.: Изд-во СПбГУ, 2010. С. 6–31.
4. *Enrico Biermann, Karsten Ehrig*, etc., EMF Model Refactoring based on Graph Transformation Concepts // Electronic Communications of the EASST. Volume 3. 2006. Pp. 3–19.
5. *Jing Zhang, Yuehua Lin, Jeff Gray*. Generic and Domain-Specific Model Refactoring using a Model Transformation Engine // Volume II of Research and Practice in Software Engineering. 2005. Pp. 199–218.
6. *Pieter Van Gorp, Niels Van Eetvelde, Dirk Janssens*. Implementing refactorings as graph rewrite rules on a platform independent metamodel // Proceedings of the 1st International Fujaba Days. University of Kassel, Germany, October 13–14, 2003. Pp. 17–24.
7. *Tom Mens*. On the Use of Graph Transformations for Model Refactoring // Lecture Notes in Computer Science. Volume 4143. Generative and Transformational Techniques in Software Engineering, 2006. Pp. 219–257.
8. *Кузенкова А. С., Дерипаска А. О., Таран К. С., Подкопаев А. В., Литвинов Ю. В., Брыксин Т. А.* Средства быстрой разработки предметно-ориентированных решений в metaCASE-средстве QReal // Научно-технические ведомости СПбГПУ: Информатика, телекоммуникации, управление. Вып. 4 (128). СПб.: Изд-во Политехнического Университета, 2011. С. 142–145.
9. *Брыксин Т. А., Литвинов Ю. В.* Среда визуального программирования роботов QReal: Robots // Материалы международной конференции «Информационные технологии в образовании и науке». Самара, 2011. С. 332–334.

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМОВ ДЕФОРМИРОВАНИЯ ТРЕХМЕРНОЙ ГЕОМЕТРИИ АНАТОМИИ ЧЕЛОВЕКА НА ОСНОВЕ WEBGL

А. С. Лушников

студент 5-го курса кафедры системного программирования;
aslushnikov@gmail.com

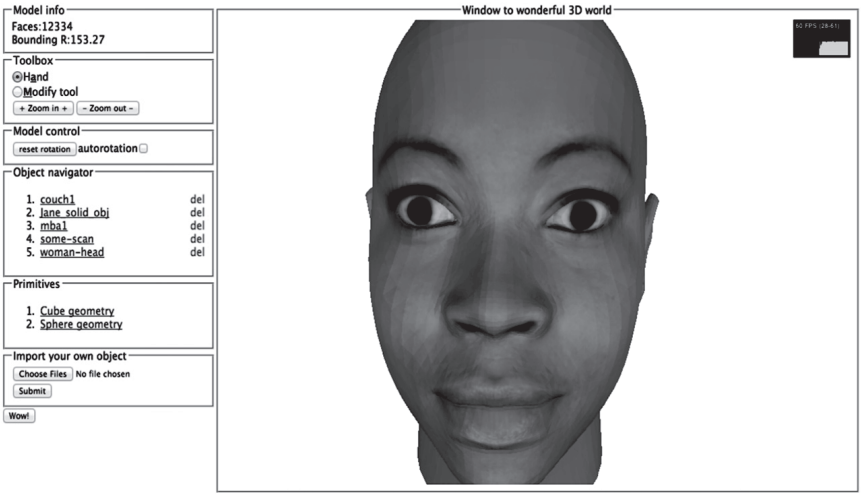
Санкт-Петербургский государственный университет

Аннотация: Выполненная дипломная работа посвящена организации демонстрации и манипуляции формой 3D модели в браузере. Разработанная система на основе технологии WebGL позволяет визуализировать 3D модель в браузере и выполнять манипуляции над ее формой.

Введение

Каждый программный продукт на протяжении всех стадий жизни сопровождает много разнообразных проблем. Это и проблемы его поддержки, и проблемы переносимости, и проблемы контроля версий и устаревания. Эти проблемы характерны как для рядовых программ, которые знакомы каждому пользователю, так и для специализированных программ, используемых врачами в их ежедневной практике для оценки, визуализирования и прогнозирования результатов пластических операций.

Один из современных подходов к распространению программного обеспечения — SAAS — призван решить целый класс таких проблем. В нашей работе было решено использовать этот подход для создания средства визуализации и изменения формы 3D моделей.



Цель выполненной работы — создать программу и веб сервис для визуализации и изменения формы трехмерной модели с использованием вычислений на сервере. Использование клиент-серверной системы дает возможность перенести трудоемкие алгоритмы обработки геометрии на серверную часть, значительно снизив требования для клиентских рабочих станций

Решение

Для визуализации 3D-графики было решено использовать веб-решение, что дает преимущества подхода Software as a service: изменение и обновление приложения происходит небольшими итерациями, все пользователи пользуются одной и той же версией приложения. При выборе движка для рендеренга 3D графики в браузере делался выбор между Flash, Unity 3d и WebGL [1]. Благодаря отсутствию плагинов для поддержки, был выбран последний вариант. В разработке WebGL участвуют компании Apple, Google, Mozilla и Opera, как следствие в их основных браузерах технология WebGL уже поддерживается. Начиная с версии iOS 4.2 WebGL поддерживается в Mobile Safari, однако он разрешен для использования только приложений iAds.

В качестве способов изменения модели изначально рассматривался инструмент, деформирующий вершины геометрии в соответствии от заданной функции, однако код хотелось написать таким образом, чтобы можно было с легкостью добавить любые другие инструменты. Для этого был использован паттерн программирования strategy, который позволяет сторонним разработчиком добавить новый инструмент, реализовав свой вариант класса.

В связи с тем, что технологии отображения 3D графики в браузере являются очень молодыми, была проведена серия тестов, призванных охарактеризовать тезис о применимости web-технологии к рендеренгу и обработке 3D-графики в браузере на различных платформах на данном этапе развития HTML 5.0. В качестве предварительного результата было установлено, что рендеринг модели на стороне GPU происходит на должном уровне быстродействия: для моделей из 120000 полигонов значение FPS держалось на уровне 60. Однако использование javascript-side кода для анализа геометрии и для пересечения ее с лучом на каждом этапе рендеренга понижало количество fps до 8. Сходный эксперимент при использовании модели из 7000 полигонов показал одинаковое значение fps в 60 единиц в обоих случаях.

В результате работы на данный момент мы располагаем клиент-серверным приложением на HTML 5.0. В среду приложения можно загружать любые obj-файлы вместе с их текстурами и файлами описаний материалов, после чего загруженный файл будет отображен слева в списке доступных для загрузки объектов. На данном этапе развития приложения существует два основных инструмента:

— «Handtool»: с помощью этого инструмента загруженный объект можно удалять и приближать, можно его вращать, используя прием «захвата» объекта и его «перетаскивания»
— «Modify tool»: с помощью этого инструмента можно искажать вершины геометрии объекта. При выбора инструмента и наводе его на объект появляется небольшойсхематичный шарик. На поверхность этого шарика после нажатия мышки спроецируются те вершины модели, которые попали в его внутренность.

Были проведены эксперименты по тесселляции объектов, однако тесселлирование сложных геометрий дает неожиданные неверные результаты. Соответствующий баг послан в сообщество разработчиков Three.js.

Серверная часть приложения написана на платформе node.js [2] и отвечает за хранение пользовательских объектов и за их конвертацию в JSON для движка three.js с помощью специального python-скрипта. Все клиент-серверные взаимодействия происходят с использованием технологии AJAX.

Серверная часть играет важную роль в следующем шаге развития проекта. Он связан с возможностью использования legacy-кода, написанного на других языках программирования (преимущественно на C++) для обработки геометрии и 3D-модели нашего клиента. Для решения этой проблемы используется технология Apache Thrift [5], позволяющую эффективно разрабатывать программные проекты на нескольких языках программирования, обеспечивая их взаимодействие с помощью сгенерированных по шаблону типов, их кода сериализации-десериализации и полного клиент-серверного стека коммуникации.

Изначальная поддержка языка Javascript обусловила наш выбор в пользу этой технологии против конкурирующей Google Protocol Buffers. Эффективное внедрение этой наработки позволит нам переиспользовать код, реализующий специальные алгоритмы обработки геометрических моделей, с минимальными усилиями и временными затратами.

Примечательно, что в процессе интеграции этой технологии была выявлена серьезная ошибка в node.js модуле проекта Apache Thrift, связанная с десериализацией вещественного 8-байтного типа. Ошибка была устранена, а соответствующий pull-request был отправлен разработчикам модуля.

Заключение

Разработанный программный продукт уже показывает интересные результаты, однако он все еще далек от заключительной стадии реализации. В дальнейшем планируется покрыть математические участки кода модульными тестами, провести комплексное тестирование приложения. Включенная поддержка использования вычислений на стороне сервера будет раскрыта за счет новых алгоритмов, реализующих и раскрывающих эту возможность. Программный код многих модулей требует некоторого переоформления и до-

полнительного документирования, что облегчит его дальнейшую поддержку и развитие и позволит привлечь дополнительные силы на его развитие. Небольшая документация проекта, написанная в текущий момент, требует основательной доработки и коррекции.

Литература

1. WebGL — http://www.khronos.org/webgl/
2. Node.js platform — http://nodejs.org/
3. Express.js, webapp mvc framework — http://expressjs.com/
4. three.js webGL framework — https://github.com/mrdoob/three.js/
5. Apache Thrift framework — http://thrift.apache.org/
6. Node-thrift framework, patched fork — https://github.com/aslushnikov/node-thrift
7. Jade, html preprocessor — http://jade-lang.com/
8. jQuery, js library — http://jquery.com/
9. jQuery form plugin — http://jquery.malsup.com/form/
10. underscore.js, js swiss knife — http://documentcloud.github.com/underscore/

РАСШИРЕННЫЙ АНАЛИЗ ОБРАЗА ПАМЯТИ НА ПЛАТФОРМЕ WINDOWS

А. А. Овчинников

студент 4 курса Математико-механического факультета;
anton.ovchinikov@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Исследование образа памяти является существенной частью компьютерного криминалистического анализа (computer forensics). В статье рассмотрены причины и цели использования образа памяти как важного объекта анализа, а также показано преимущество использования данного вида анализа совместно с такими дополнительными объектами, как файл подкачки и образ исполняемого файла на диске.

Введение

Компьютерный криминалистический анализ (computer forensics) представляет собой комплекс мероприятий, направленных на изучение вычислительных устройств и носителей информации в целях поиска и фиксирования доказательств (например, в ходе расследования гражданских или уголовных дел).

До недавнего времени главным объектом изучения при компьютерном криминалистическом были жесткие диски ([1]). С учетом особенностей функционирования наиболее распространенных на данный момент операционных (Windows, Mac OS, GNU/Linux) и файловых (NTFS, ext, HFS) систем, имеется возможность восстанавливать в том числе и удаленные с носителя данные, что позволяет проследить хронологию действий, совершенных с носителем. Однако, на данный момент с изъятием и изучением исключительно долгосрочных носителей данных связано несколько проблем:

1. Распространение облачных хранилищ данных.

Вся информация или ее часть может храниться на удаленных серверах в интернете, и изъятие этих данных может быть существенно осложнено как технически, так и с законодательной точки зрения. Подозреваемый может использовать свой компьютер (ноутбук, мобильное устройство) как терминал для доступа к данным, не загружая их на само устройство.

2. Распространение твердотельных накопителей (SSD, Solid State Drive)

Обладая многими достоинствами по сравнению с обычными жесткими дисками (HDD, Hard Disk Drive), твердотельные диски обладают особенностями функционирования, значительно осложняющими вос-

становление удаленных файлов. Основная проблема состоит в том, что SSD может самостоятельно очищать блоки памяти, если ОС помечает их как удаленные. Это необходимо для подготовки к записи, потому что ячейки флеш-памяти должны быть очищены перед тем, как на них будут записаны новые данные (см. команду TRIM[1]).

3. Шифрование и запаковка файлов

Использование подозреваемым RAM-дисков, шифрованных контейнеров и разделов существенно препятствует анализу. Похожая ситуация наблюдается с запакованными и обфусцированными файлами (например, запакованные вредоносные программы). Бурное развитие индустрии вредоносного программного обеспечения приводит к появлению новых, более совершенных методов упаковки (полиморфные упаковщики, протекторы[2]), целью которых является усложнение анализа.

4. Невозможность получения полного представления о системе

Анализируя жесткий диск компьютера, иногда трудно восстановить полную картину деятельности его обладателя. История браузеров, файлы журналов могут намеренно удаляться для уничтожения такой информации, как посещенные веб-сайты, использованные сетевые соединения и т. п.

По этим причинам предлагается использовать образ физической памяти компьютера как основной объект анализа.

Использование образа памяти

Образ памяти захватывается на работающей системе (например, с помощью программы FTK Imager), после чего анализируется. В памяти может быть найдена информация, которую трудно или невозможно получить, располагая только жестким диском: расшифрованные файлы или пароли, распакованные вредоносные программы, открытые в момент захвата образа сетевые подключения. Например, последнее может помочь в случае, если некто подозревается в управлении ботнетом. Более того, в памяти может быть найдена описанная выше информация, которая в момент захвата уже не использовалась. К примеру, можно установить, какие процессы или сетевые подключения недавно были завершены.

---

[1] http://en.wikipedia.org/wiki/TRIM

[2] Протектор — программа, преобразующая другую программу в вид, который осложняет исследование и отладку путем добавления антиотладочных приемов и шифрования.

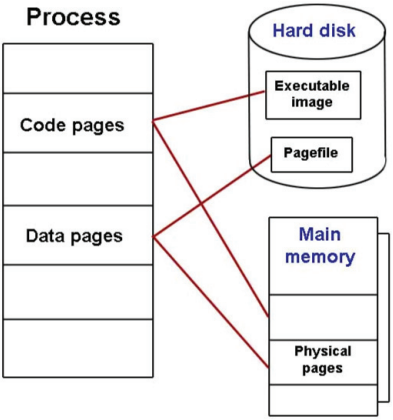


Рис. 1. Отображение адресного пространства процесса

Особый интерес при исследовании процесса представляет его адресное пространство (см. Рис. 1), восстановление которого позволяет приступить к более тщательному анализу. Например, можно привлечь вирусного аналитика для исследования полученного адресного пространства, если есть подозрение, что в контексте процесса исполняется вредоносный код. Однако, обладая исключительно образом памяти, полностью восстановить адресное пространство процесса иногда просто невозможно. На то есть две основные причины:

1. Файлы подкачки (pagefiles)

Windows использует файлы подкачки для «расширения» физической памяти, перемещая в них данные, не помещающиеся в оперативной памяти.

2. Отображаемые в память файлы (memory-mapped files)

Это могут быть не только файлы, специально отображенные запущенным процессом, но и части файла образа этого процесса. Например, отображенным на память может быть PE-заголовок программы.

По этим причинам предлагается использовать т.н. «расширенный» образ памяти, то есть образ, дополненный файлом подкачки системы (снятый по возможности в одно время с образом памяти), исполняемым файлом исследуемого процесса и файлами используемых процессом динамических библиотек. Таким образом, появится возможность полностью восстановить пользовательское адресное пространство процесса.

Поиск процессов

Для успешного восстановления необходимо уметь находить в образе памяти структуры EPROCESS. В семействе ОС Windows NT данная структура находится в адресном пространстве ядра и описывает процесс для ядра ОС (см. Рис. 2). EPROCESS содержит структуру KPROCESS как подструктуру (KPROCESS представляет собой данные, необходимые для планировщика ОС); также EPROCESS содержит ссылки на другие важные для процесса структуры (такие как список потоков, VAD-дерево, Process Environment Block и т.п.). Также EPROCESS содержит ссылку на директорию страниц процесса, с помощью которой происходит трансляция виртуальных адресов в физические и которая необходима для восстановления адресного пространства процесса.

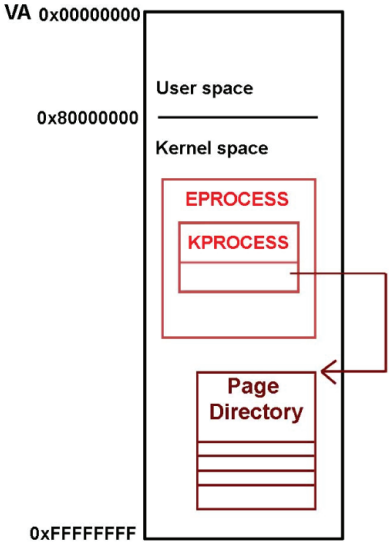


Рис. 2. Структуры EPROCESS и KPROCESS в памяти

Наиболее эффективным на данный методом поиска структур EPROCESS в образе памяти является сигнатурный поиск. Данный метод состоит в выделении некоторых полей структур, содержание которых подчиняется определенным эвристикам. Например, поле DirectoryTableBase в составе KPROCESS должно содержать адрес в пространстве ядра (> 0x80000000), выровненный по адресам, кратным 0x20; адреса ThreadListHead.Flink и ThreadListHead.Blink должны также указывать на виртуальный адрес в пространстве ядра. Или, например, было замечено, что поля Type и Size в подструктуре DISPATCHER_HEADER должны быть одинаковыми для всех процессов (конкретные значения зависят от версии Winows, для Windows 7

они составляют 0x03 и 0x26 соответственно). Найдя в памяти места, где эти эвристики выполняются, можно, проведя более тщательные проверки (например, проверки принадлежности некоторых адресов пространству ядра), убедиться, что найдена именно структура EPROCESS.

Восстановление адресного пространства

После нахождения структуры EPROCESS требуемого процесса можно начинать восстановление адресного пространства. В общем случае восстановление сводится к перебору всех PDE- и PTE-записей (и соответсвующих таблиц страниц). В зависимости от места, где находится страница для данного PTE, производится необходимое действие. Например, если PTE указывает на страницу в файле подкачки, то идет обращение туда; если PTE описывает страницу отображенного в память файла, то идет обращение к файлу, и т.п.

Получаемые страницы копируются в отдельный («выходной») файл. Также необходимо поддерживать индекс — вспомогательную структуру, позволяющую понять, какое смещение в получаемом файле соответствует данному виртуальному адресу. В простейшем случае, это массив структур типа «диапазон — смещение».

В итоге, получается проиндексированное адресное пространство процесса. Из него восстанавливается исполняемый образ файла, который потом можно будет запускать, исследовать и отлаживать. Делается это путем корректирования PE-заголовка имеющегося файла образа (необходимо скорректировать размеры секций, точку входа). Стоит отметить, что таким образом может получиться файл, во многом отличный от того, который был получен с диска. Хорошим примером может служить троянская программа, хранящаяся на диске в запакованном виде. После восстановления её адресного пространства из образа памяти, анализ программы существенно упрощается.

Заключение

В статье рассмотрены основные преимущества использования расширенного образа памяти и его применение для решения практической задачи восстановления полного адресного пространства процесса. Данный подход может облегчить анализ запакованных или зашифрованных вредоносных файлов, а также может помочь при извлечении из образа информации произвольного типа (изображений, pdf-документов и т.п.).

Литература

1. *J. Medeiros.* NTFS Forensics: a programmers view of raw file systems data extraction, 2008.
2. *М. Руссинович.* Внутреннее устройство Microsoft Windows (4-е изд.), 2008.

3. *J. Kornblum.* Using Every Part of the Buffalo in Windows Memory Analysis. Digital Investigation Journal. 2007.
4. *Schuster.* Searching for Processes and Threads in Microsoft Windows Memory Dumps. Digital Forensic Research Workshop. 2006.
5. *M. Burdach*. An Introduction to Windows Memory Forensic. 2005.
6. *Okolica, G. L. Peterson*. Windows operating systems agnostic memory analysis. Digital Forensic Research Workshop. 2010.
7. *B. Dolan-Gavitt, A. Srivastava, P. Traynor.* Robust Signatures for Kernel Data Structures. 2009.

РАСПОЗНАВАНИЕ НАРИСОВАННЫХ ДИАГРАММ В ПРОЕКТЕ QREAL

М. С. Осечкина

кафедра системного программирования, 5 курс;
osechkina.masha@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Рассматриваются основные цели и задачи для создания инструмента статического распознавания диаграмм в рамках проекта QReal. Описывается созданный прототип распознавания простейших диаграмм.

Введение

Диаграммы и схемы являются незаменимыми инструментами в бизнесе и индустрии. Они широко используются при разработке бизнес-отчетов, научных статей, технической документации и других типов документов. Кроме того, создание диаграмм является неотъемлемой составляющей любого UML-средства и частью DSM-подхода. Большинство людей, которым требуется создать диаграмму, используют специализированные графические редакторы, к примеру, Microsoft Visio. Как правило они довольно эффективны, но, как показывают эксперименты, рисование диаграммы от руки занимает на 90% меньше времени, чем ее создание в специализированном редакторе [1]. Можно предположить, что инструмент для распознавания отсканированных, сфотографированных или нарисованных с помощью графического планшета диаграмм значительно сэкономит время пользователя.

Существует множество решений, предназначенных для графических планшетов, которые позволяют распознавать нарисованные пользователем диаграммы динамически, — в процессе создания диаграммы с учетом последовательности рисования штрихов и их направлений. В некоторых случаях у пользователя может не оказаться под рукой планшета или другого необходимого программного и аппаратного обеспечения для рисования диаграммы и сопутствующего динамического распознавания. Иногда удобнее рисовать диаграмму от руки на листе бумаги или, например, при обсуждении какой-то задачи на доске перед аудиторией. Впоследствии нарисованную диаграмму как правило надо где-то хранить, возможно, осуществлять на ней поиск, редактировать. В данном случае было бы полезно иметь инструмент для статического распознавания диаграмм, при котором не делается никаких попыток к распознаванию до запроса от пользователя и не учитывается последовательность штрихов и их направление.

На кафедре системного программирования математико-механического факультета СПбГУ разрабатывается metaCASE-система QReal [3]. В QReal для быстрой разработки новых редакторов диаграмм имеется метаредактор с графическим редактором формы элементов. Он позволяет создавать метамодели новых языков, описывая имеющиеся на диаграммах элементы и связи между ними, задавать визуальное представление элементов на диаграммах. Созданную метамодель можно скомпилировать в подключаемый модуль и подключить его к QReal прямо в процессе работы. Хотелось бы иметь инструмент распознавания, на вход которому можно подать сфотографированную или отсканированную диаграмму и получить диаграмму QReal, по которой можно генерировать код. При этом требуется, чтобы набор распознаваемых объектов можно было динамически расширить: хотелось бы, чтобы при создании нового редактора можно было просто загрузить в инструмент распознавания описание новых элементов, и после этого множество распознаваемых диаграмм было автоматически расширено. В рамках этой работы описывается прототип статического распознавания простейших диаграмм.

Аспекты распознавания нарисованных диаграмм

Процесс распознавания сфотографированных и отсканированных диаграмм включает в себя множество подзадач. Ниже описаны некоторые задачи для полноценной системы статического распознавания диаграмм.

Прежде всего потребуется отделить фон от контура. Есть различные алгоритмы и библиотеки, предоставляющие такую возможность.

Диаграммы, нарисованные пользователем, могут содержать в себе названия объектов, комментарии, кратность связи и другую текстовую информацию. Для распознавания этой информации прежде всего потребуется обнаружить текст на диаграмме для последующего распознавания. Может случиться так, что распознанный текст даст дополнительную информацию о свойствах объекта: полезно уметь различать к какому объекту и свойству объекта относится текст. К примеру, надпись рядом с объектом может обозначать имя этого объекта. Логично, если при обработке такой ситуации, инструмент распознавания автоматически присваивал бы имени распознанного объекта распознанное значение текста. В случае если текст находится рядом с концом связи и удовлетворяет шаблону кратности связи, следовало бы автоматически изменять кратность у распознанной и сгенерированной связи на конечной диаграмме QReal.

В большинстве случаев графическое представление объекта в QReal является набором из простых форм — отрезков, окружностей и дуг. Оно может быть несвязным, к примеру, представлять собой 2 концентрические окружности. Графическое представление объекта может включать не только набор ломанных, но и заштрихованные области. Кроме того в некоторых редакторах графическое представление объекта — иконка, иногда цветная. Если поль-

зователю захочется нарисовать диаграмму от руки так, что цветовая палитра элементов соответствует графическому представлению объектов в QReal, можно воспользоваться данным фактором при распознавании, но в то же время, если пользователь будет рисовать лишь одним цветом, распознавание должно оставаться корректным.

Иногда тип связи можно определить по виду ее концов, например, у наследования на конце — стрелка, у агрегирования — заштрихованный ромб. Кроме того порой тип связи и ее направление легче определить с помощью семантики [2] — в редакторах QReal указывается связи какого типа могут соединять данные элементы, а также допустимое направление этих связей. Распознавание должно включать в себя обработку таких мелких деталей, как концы связей.

Может случиться так, что на диаграмме часть объектов были распознаны верно, а часть — нет. Полезно иметь возможность выделить неверно распознанные объекты и попробовать распознать их с учетом ошибки. Ошибка в распознавании может произойти как при неправильном разделении на объекты (объединили то, что не следовало, приписав связь к объекту, или не включили в объект, то что следовало включить), так и при неверном распознавании верно выделенных объектов. Во втором случае можно усовершенствовать этап распознавания отдельных элементов, включив в систему обучение.

QReal — metaCASE-система, в которой набор возможных объектов на диаграмме может быть динамически расширен при создании нового редактора. Требуется, чтобы в полноценной системе распознавания диаграмм QReal множество распознаваемых объектов тоже можно было бы динамически расширить.

Для прототипа мы ограничились черно-белой картинкой, на которой контур отделен от фона. В качестве множества распознаваемых объектов были выбраны некоторые элементы из редакторов QReal, представляющие собой простые многоугольники, и связи. Будем считать, что все связи имеют один тип, не имеют направления и представляют собой кривую, соединяющую два элемента. Основное требование к объектам — отсутствие точек, в которых сходится более двух кривых, и замкнутость. На этапе создания прототипа, мы поставили такое ограничение для простоты выделения объектов. В прототипе не предполагается поиск и распознавание текста на диаграмме, заштрихованных областей, цветных фигур, графическим представлением которых являются иконки, типов и направления связей. Пользователь рисует диаграмму, при этом не предпринимается никаких попыток к распознаванию, после команды пользователя, — нажатия определенной кнопки, происходит распознавание. Чтобы алгоритмы, используемые в прототипе, можно было применить для создания полноценной системы распознавания отсканированных диаграмм, было принято решение не использовать информацию о пути мыши, а интерпретировать полученную диаграмму как растеризованную картинку.

Алгоритм

Алгоритм распознавания диаграмм можно разделить на несколько этапов.

1) Выделение объектов на диаграмме. На этом этапе необходимо представить диаграмму как набор объектов, которые впоследствии можно будет распознавать.

В прототипе выделение объектов происходит в два этапа. Первый этап — разделение на компоненты, которые являются непрерывными кривыми и представляют собой цепь из пикселей. Второй этап — слияние некоторых компонент. Разделение на компоненты происходит следующим образом: рассматривается произвольный пиксель из контура диаграммы, если при пересечении диаграммы с квадратом размера 3X3 с центром в этом пикселе получается множество пикселей, которое может получиться при растеризации ровно одного отрезка, то считается, что все пиксели из пересечения относятся к одной компоненте. В ином случае для каждого пикселя из окрестности, не отнесенного ни к какой компоненте, строится своя компонента. При таком построении получается, что в точках пересечения объектов компоненты либо начинаются, либо заканчиваются. Затем удаляются слишком мелкие компоненты для последующего слияния. Сначала происходит слияние компонент, которые образуют углы объекта — если в небольшой окрестности конца компоненты нет другого ее конца, но есть только один конец другой компоненты, то 2 компоненты сливаются. Затем на основании гладкости сливаются компоненты, которые являются продолжением друг друга. После чего ищутся циклы из несамопересекающихся компонент, которые сливаются в одну компоненту. После этого каждая компонента интерпретируется как отдельный объект.

2) Разделение объектов на элементы и связи между ними. После того как все объекты выделены требуется понять какой объект является элементом, а какой — связью между элементами.

В рамках заданных ограничений для прототипа принцип разделения на элементы и связи предельно прост в силу однообразия множества распознаваемых объектов. Несложно понять, что выделенный на диаграмме прототипа объект имеет самопересечение тогда и только тогда, когда этот объект — многоугольник. Таким образом, в случае если объект имеет самопересечение он интерпретируется как элемент, а если не имеет, то — как связь.

3) Распознавание элементов и связей. В рамках прототипа распознавание связи фактически сводится к определению ее крайних точек и, возможно, элементов, которые она соединяет. Распознавание элементов проводится с помощью инструмента распознавания жестов мышью, который был разработан ранее в рамках проекта QReal [4].

4) Вывод распознанной диаграммы. На этом этапе происходит прорисовка распознанных элементов и связей между ними так, что их местоположение соответствует местоположению исходных объектов на диаграмме, нарисованной пользователем.

Заключение

В работе был описан прототип распознавания нарисованных диаграмм в рамках проекта QReal, описаны идеи алгоритмов, используемых в данном прототипе. В дальнейшем планируется создать полноценную систему распознавания диаграмм, для этого необходимо максимально расширить множество распознаваемых диаграмм.

Литература

1. *Khalde S. Refaat, Wael N. Helmy, AbdelRahman H. Ali, Mohamed S. AbdelGhany, Amir F. Atiya*. A New Approach for Context-Independent Handwritten Offline Diagram Recognition // UCLA Engineering Computer Scince, URL: http://www.cs.ucla.edu/~krefaat/Refaatetal08.pdf
2. *Isaac J. Freeman, Beryl Plimmer*. Connector Semantics for Scetched Diagram Recognition // Conferences in research and practice in information technology, URL: http://crpit.com/confpapers/CRPITV64Freeman.pdf
3. *Кузенкова А. С., Дерипаска А. О., Таран К. С., Подкопаев А. В., Литвинов Ю. В., Брыксин Т. А*. Средства быстрой разработки предметно-ориентированных решений в metaCASE-средстве QReal // Научно-технические ведомости СПбГПУ, Информатика, телекоммуникации, управление. Вып. 4 (128). СПб.: Изд-во Политехнического Университета. 2011. С. 142–145
4. *Осечкина М. С*. Многоштриховые жесты мышью в проекте Qreal // Сайт кафедры системного программирования СПбГУ, URL: http://se.math.spbu.ru/SE/YearlyProjects/2011/YearlyProjects/2011/445/445_Osechkina_report.pdf

СРЕДСТВА ОПИСАНИЯ ГЕНЕРАТОРОВ КОДА ДЛЯ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ РЕШЕНИЙ В METACASE-СРЕДСТВЕ QREAL

А. В. Подкопаев

студент; podkoav239@gmail.com

Т. А. Брыксин

ст. преп. кафедры системного программирования; timofey.bryksin@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В статье описывается задача разработки генераторов кода по графическому представлению программы применительно к DSM-средствам. Приводится способ задания генераторов с помощью языка описания генераторов, представляющего из себя шаблоны и встроенные в них управляющие конструкции. Представлена реализация этого подхода в metaCASE-системе QReal в виде прототипа, включающего в себя интерпретатор языка и редактор для него.

Введение

На данный момент существует достаточно много систем визуального моделирования — CASE-систем[1]. Некоторые из них являются также metaCASE-системами[2] (Actifsource[3], Microsoft Visual Studio Visualization and Modeling SDK[4], MetaEdit+[5]), что позволяет внутри них разрабатывать новые визуальные языки [1] и редакторы к ним.

Одной из задач, возникающих при создании предметно-ориентированного инструментария внутри metaCASE-систем, является разработка генератора запускаемого программного кода или другого текстового представления данных, необходимого при разработке продукта (документация, прототип приложения, интерфейсы подсистем). Подобные генераторы необходимы практически для каждого визуального языка, таким образом для DSM-подхода[6] становится крайне важно упростить процесс разработки подобных трансляторов.

---

[1] CASE-система, http://en.wikipedia.org/wiki/CASE_tool

[2] metaCASE-система, http://en.wikipedia.org/wiki/MetaCASE_tool

[3] Actifsource http://www.actifsource.com/

[4] Microsoft Visual Studio Visualization and Modeling SDK http://archive.msdn.microsoft.com/vsvmsdk

[5] MetaEdit+ http://www.metacase.com/

[6] Предметно-ориентированное моделирование, (Domain Specific Modeling, DSM), http://en.wikipedia.org/wiki/Domain-specific_modeling

Хотя логически генераторы достаточно простые, написание их на языке общего назначения превращается в длительный и сложный процесс. Задачей данной работы была разработка системы, которая упростила бы этот процесс.

Постановка задачи

В рассматриваемой области генератор для визуальных языков — это инструмент, который обходит граф модели и получает некоторое текстовое представление этого графа. Такие трансляторы по способу их работы можно разделить на две категории: с использованием шаблонов генерации [2] и без. Генераторы без использования шаблонов обычно включают в свой исходный код большое количество управляющих символов системы ввода-вывода (табуляций, переводов строки) для текстового форматирования результата. Написание подобных систем осложнено отсутствием наглядности в представлении итогового кода. Для решения проблемы можно использовать шаблоны генерации, которые фиксируют структуру получаемого результата, оставляя за самим генератором возможность подставлять в необходимые позиции требуемое наполнение.

Необходимо было повысить уровень абстракции при создании генераторов, так как слишком много усилий тратится на рутинную работу. При написании «вручную» на языке общего назначения нескольких генераторов было замечено, что большинство из них имели схожие логические блоки команд, например, позволяющие осуществлять навигацию по графу модели или работу с файлами. Было решено сделать специальный язык и выделить часто используемые в генераторах действия в отдельные языковые конструкции. Программы представляли бы из себя шаблоны с управляющими конструкциями, что позволило бы использовать основной плюс шаблонного метода (наглядность), уйдя от написания большого количества рутинного кода, заменяя целые наборы строк, отвечающих за типовые операции, одной-двумя инструкциями.

Подобный подход используются и в индустрии. В средстве MetaEdit+ компании MetaCase применяется язык описания генераторов, который представляет из себя набор команд для перехода между объектами, получения их параметров, передачи таких параметров в результирующий поток [3]. В таком подходе не используется наглядность шаблонного метода, но он гибок, что позволяет писать достаточно широкий класс генераторов. В то время как в продукте Actifsource используется специальный редактор описания именно шаблона генерации [4]. Это вносит существенное ограничение на визуальный язык, заставляя его быть довольно громоздким. Однако, повышается наглядность такого подхода в связке с редактором, который исключает из видимого текста конструкции возвращения свойств объекта, перенося их на особые символы. Например, подчеркнутая конструкция в шаблоне оз-

начает, что в этом месте результата должно появится не сама она, а соответствующее поле объекта диаграммы.

Было принято решение совместить достоинства этих двух подходов. В отличие от средств MetaEdit+ на первое место выводятся не управляющие конструкции, а скелет итогового приложения — шаблон. С другой стороны, сохраняются управляющие конструкции, позволяющие уменьшить влияние на визуальный язык.

QReal

QReal[1] представляет из себя инструментальную среду визуального моделирования. Каждая диаграмма, созданная в среде, хранится в репозитории как набор элементов диаграмм с их связями и свойствами. В рамках QReal существует механизм метаредактора, который позволяет для необходимого класса задач создать подходящий визуальный язык [5]. Этот визуальный язык также задается как диаграмма, в дальнейшем называемой метамоделью языка.

Проект разрабатывается на базе кафедры системного программирования Санкт-Петербургского Университета с 2007 года.

Язык описания генераторов Geny

Язык создавался для генераторов, использующих репозиторий QReal. Объекты в нем хранятся под универсальными идентификаторами. По идентификатору через API репозитория можно получить свойства, связанные с объектом (имя, надпись, расположение на диаграмме, тип, родительские/дочерние объекты и т.д.). Некоторые из свойств являются списковыми, например, связанные с данным объекты, другие — просто текстовые поля или числа. Для списков объектов необходимо иметь конструкцию перехода по ним, а для обычных полей объектов — уметь передавать их в поток вывода. При генерации кода создается много файлов, поэтому нужна была легкая система для работы с ними.

Эти требования и легли в основу модели расширения шаблона генерации.

Основные концепции языка

- **Принцип «прямой передачи».** Все, что находится в программе и не обрамлено управляющими конструкциями, передается в результирующий поток (например, в выходной файл). Этот принцип позволил освободить код генератора от большого количества лишних символов. Благодаря этой концепции работа над генераторами становиться максимально

[1] QReal, http://qreal.ru/

прозрачной по отношению к представлению итога функционирования создаваемой системы.

- **Принцип «текущего объекта».** В конкретный момент времени генератор кода часто обращается к атрибутам только одного объекта. Этот объект можно задавать с помощью конструкций объектных переходов, то есть конструкций позволяющих сменить объект, над которым происходит работа. «Текущий объект» есть текущий узел при обходе графа. Механизм позволяет не писать имя объекта, при возвращении его атрибутов в поток, также укорачивает объектные переходы к связанным сущностям.
- **Система заданий.** Каждый файл программы на разработанном нами языке Geny представляет из себя именованное задание. Задания могут использовать внутри себя другие. Используется «текущий объект» вызванного места. Этот механизм обеспечивает модульность программ на языке Geny, делая возможным написание более сложных генераторов. Задания организуются в проект с помощью специального файла, в котором указывается путь до сохраненной диаграммы, а потом перечисляются файлы, в которых записаны необходимые задания. Интерпретатор исполняет задание Main, которое уже вызывает другие по необходимости.
- **Управляющие строки.** Такие строки начинаются со служебной конструкции #! и необходимы для выполнения служебных операций:
  - **a.** toFile fileName — перенаправляет генерацию блока в файл с переданным именем;
  - **b.** foreach — служит для выполнения блока кода с объектами из некоторого списка;
  - **c.** saveObj markName — сохраняет «текущий объект» по переданному имени;
  - **d.** switch/case — позволяют делать условные переходы, сравнивая определенные значения с атрибутами «текущего объекта»;
  - **e.** {,} — отделяют блоки кода для других конструкций.
- **Управляющая конструкция внутри контекста.** Конструкция является аналогом помеченных мест для вставки в шаблонах. Она позволяет вставить представление параметра объекта или результат выполнения задания. Для передачи атрибута «текущего объекта» в языке Geny необходимо написать название этого атрибута между @@ @@ (к примеру, если нужно имя объекта — @@name@@), если нужен параметр некоторого помеченного объекта, то нужно перед именем параметра написать имя метки и @ (@@methodA@methodVisibility@@). Вставка задания происходит командой @@!task taskName@@.
- **Система меток объектов.** Расширяет предыдущий метод, сохраняя с помощью управляющей конструкции saveObj объект по некоторому имени. В дальнейшем по имени можно возвращать свойства помеченной сущности. Это позволяет работать сразу с несколькими объектами — напри-

мер, если в дальнейшем в генераторе придется пользоваться «текущим объектом» в контексте работы с другими объектами. К примеру, если необходимо для UML-класса получить его код на C++, нужно обойти все методы класса и при генерации их реализаций нужно знать имя класса, которое является атрибутом объекта-класса. Удобно запомнить этот объект, а потом использовать с помощью метки.

Редактор языка Geny

В процессе апробирования языка было выявлено, что необходимо разработать специализированный редактор, который позволил бы в большей степени повысить наглядность кода, упростив написание генераторов. Связано это с тем, что язык Geny содержит много малоинформативных с точки зрения пользователя префиксов (#!) и символов (@@), которые по возможности необходимо скрывать. Для более интуитивного восприятия кода генератора также крайне полезно делать цветовые акценты на управляющих конструкциях — подчеркивать их или выделять жирным шрифтом.

В итоге был разработан редактор, который внес следующие изменения в редактирование Geny-генераторов:

- Управляющие строки выделяются не посредством префикса #!, а с помощью цвета фона. Это позволяет разделить два языка — язык описания генератора и целевой язык генерации. На данный момент изменение типа строки происходит с помощью переключения его явно для каждой строки. В дальнейшем планируется сделать механизм, который будет анализировать введенную строку и присваивать ей тип управляющей, в случае если она похожа на конструкцию языка Geny, а пользователю необходимо будет лишь корректировать результат в случае ошибки.
- Обрамление посредством @@ управляющих конструкций, которые возвращают свойства объекта, заменено выделением жирным шрифтом, которое происходит автоматически при вводе названия свойства объекта, в случае подтверждения его автодополнением. Список доступных названий свойств для текущего генератора берется из метамодели визуального языка.
- На основе информации об управляющих блоках реализован алгоритм, который во время редактирования форматирует строки, относящиеся к коду Geny. Вложенные блоки табулируются, что также способствует повышению удобочитаемости генератора.
- Убрана необходимость писать управляющую строку #! {, которая открывает блок кода, так как она не несет полезной для пользователя информации. Для каждой управляющей строки определяется, является ли она открывающей для блока, что означает следование #! {за ней, а значит их можно логически объединить. Так сопоставляется начало блока с закрывающей строкой #!}.

- Есть возможность скрыть управляющие строки, чтобы можно было наглядно увидеть шаблон генерации. Это позволяет иметь более четкое представление, что в итоге будет сгенерировано.

Сам редактор интегрирован в среду QReal. Проект Geny-генератора дополняет метамодель визуального языка информацией о способе трансляции диаграмм, созданными с помощью данного языка, в итоговый код, что позволяет использовать генератор при работе над такими диаграммами.

```
1  Task JavaClass
2  / Produce Java class from repo
3
4  foreach Class in elementsByType(Class)
5              toFile name.java
6  class name {
7                          foreach MethodsContainer in children
8                                      foreach Method in children
9              methodVisibility methodReturnType
10             methodName( @@!task MethodParameters@@ ) {
11             }
12                                     }
13                         }
14
15                         foreach FieldsContainer in children
16                                     foreach Field in children
17             fieldVisibility fieldType fieldName;
18                                     }
19                         }
20             }
21 }
22 }
```

Рис. 1. Представление генератора прототипа Java-класса в редакторе

```
6  class name {
9              methodVisibility methodReturnType
10             methodName( @@!task MethodParameters@@ ) {
11             }
14
17             fieldVisibility fieldType fieldName;
21 };
```

Рис. 2. Представление генератора прототипа Java-класса без управляющих строк

Использование редактора позволяет уменьшить количество управляющих строк примерно на треть за счет сокрытия #! {, так как почти каждая конструкция языка Geny подразумевает структуру. Замена обрамлений @@, необходимых для интеграции управляющих конструкций внутрь шаблонных строк, на выделение жирным шрифтом не только уменьшает объем кода, но и помогает ярче отделить конструкции от текста, непосредственно передаваемого в выходной поток. А режим отображения без управляющих

строк дает ясное представление о том, как будет выглядеть итог работы генератора.

В данном примере количество управляющих строк сократилось с 20 до 14, было убрано 7 обрамлений @@, 14 префиксов #!. Учитывая, что размер генератора в редакторе составляет 22 строки, это является существенным улучшением.

Заключение

В рамках данной работы был проведен обзор реализованных подходов к решению поставленной задачи, в результате которого были выявлены пути для улучшения существующих моделей разработки генераторов. На основе сделанных выводов были разработаны язык описания генераторов Geny и интерпретатор к нему, которые апробировались на примерах и корректировались по результатам тестирования. После чего было принято решение реализовать специализированный редактор для созданного языка с целью упростить использование конструкций Geny. Данный редактор позволил сократить объем кода генераторов, улучшить их восприятие.

В дальнейшем планируется переписать часть внутренних генераторов QReal с C++ на Geny, модернизировать язык, в соответствии с требованиями, которые будут возникать при использовании языка. Также в будущем необходимо будет провести более тесную интеграцию редактора с другими DSM-инструментами QReal, в первую очередь с метаредактором. В частности, добавить возможность автоматического обновления генератора при изменении метамодели для некоторых простых случаев, таких как переименование объектов, свойств.

Литература

1. *Raphael Mannadiar* and *Hans Vangheluwe*. 2010. Domain-specific engineering of domain-specific languages. In Proceedings of the 10th Workshop on Domain-Specific Modeling (DSM «10). ACM, New York, NY, USA, Article 11, 6 pages.
2. *M. Mazaud, R. Rakotozafy*, and *A. Szumachowski-Despland*. 1987. Code generator generation based on template-driven target term rewriting. In on Rewriting techniques and applications, Pierre Lescanne (Ed.). Springer-Verlag, London, UK, 105–120.
3. *Tolvanen, J.-P*. Making model-based code generation work — Practical examples (Part 2), Embedded Systems Europe, Vol. 9, 64 (March), 2005.
4. *M. Riser, R*. Carrara Modelle im Rampenlicht: Voraussetzungungen für agiles Arbeiten mit Modellen, OO-Magazin OBJEKTspektrum, June 2010.
5. *Кузенкова А. С., Дерипаска А. О., Таран К. С., Подкопаев А. В., Литвинов Ю. В., Брыксин Т. А*. Средства быстрой разработки предметно-ориентированных решений в metaCASE-средстве QReal // Научно-технические ведомости СПбГПУ, Информатика, телекоммуникации, управление. Вып. 4 (128). СПб.: Изд-во Политехнического Университета. 2011. С. 142–145.

РАЗРАБОТКА ВИЗУАЛЬНОГО ИНТЕРПРЕТАТОРА МОДЕЛЕЙ В СИСТЕМЕ QREAL

В. А. Поляков

4 курс, кафедра системного программирования;
vap.polyakov@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В статье рассматривается задача интерпретации визуальных языков моделирования применительно к metaCASE-системе. Представлен способ задания семантики визуальных языков моделирования, являющийся сочетанием идей двух основных существующих подходов: подхода, используемого в языке Executable UML, и Dynamic Meta Modeling. Приводится описание реализации предложенного способа в системе QReal в виде прототипа, включающего в себя все стадии работы интерпретатора от создания редактора семантики до непосредственного «исполнения» модели.

1. Введение

Предметно-ориентированный подход к разработке программного обеспечения в последнее время становится всё более популярным, в том числе и в области проектирования, основанной на моделях. При таком подходе обычно используются визуальные языки моделирования, которые бывают статические, описывающие структуру разрабатываемой системы, и поведенческие, определяющие динамику ее поведения.

Возможность интерпретации поведенческих диаграмм значительно ускорила бы процесс проектирования, предоставив разработчику средство для поиска ошибок и отладки моделей ещё до завершения их создания. Так или иначе, этот подход уже был реализован в некоторых CASE системах (например, Borland Together[1]), в данном исследовании рассматривается его применение к metaCASE системе, таким образом, интерес представляют средства быстрого и удобного создания интерпретаторов для разрабатываемых динамических визуальных языков.

2. Реализация

Данное исследование является логическим продолжением предыдущей работы автора [6], в которой была определена инфраструктура визуального интерпретатора в системе QReal[2] [7], разработана основа модуля по рабо-

[1] Borland Together, http://www.borland.com/us/products/together/
[2] QReal, http://qreal.ru/

те с отладчиком целевого языка генерации и были реализованы несколько частных примеров визуальных интерпретаторов. В этой работе исследуется вопрос задания семантики произвольных поведенческих языков и дальнейшей визуализации интерпретации по ней.

2.1. Обзор предметной области

Для того, чтобы появилась возможность интерпретации поведенческих визуальных языков, необходимо формально определить семантику данного языка. При этом способ задания такой семантики должен быть удобным, точным и визуальным для наглядности. При изучении этой задачи были рассмотрены существующие подходы к определению и заданию семантики визуальных языков.

Executable UML [2, 3, 4] является подмножеством языка UML с чётко определённой семантикой для каждого элемента. Подход к интерпретации, использующийся в нём, основан на представлении поведения системы в виде набора диаграмм состояний для каждого активного объекта. Действия при проходе каждого состояния в этих диаграммах записываются при помощи различных языков действий (action languages), необходимых для создания экземпляров классов, установки связей, выполнения операций над атрибутами и т.п. Данный способ базируется на таких понятиях, как класс, объект (экземпляр класса) и диаграмма состояний. В общих случаях у системы бывает более сложное поведение, нежели конечный автомат.

Dynamic Meta Modeling был описан в статье [1], а позднее использован при создании инфраструктуры визуального отладчика моделей в Eclipse [5] (без реализации, зато с представлением возможного пользовательского интерфейса). Главным в DMM-подходе является момент визуализации процесса интерпретации модели. Исходную модель рассматривают в виде графа, а для интерпретации используют правила его преобразования. Подробный анализ данного метода изложен в той же статье [1]. Несмотря на то, что DMM-подход формален и точен, обладает высокой понятностью для продвинутых пользователей, а также универсальностью и анализируемостью в связи с отсутствием большого количества различных сложных особенностей, которые нужно учитывать при создании семантики, он ещё не был реализован.

2.2. Описание собственного решения

По результатам исследования и анализа существующих способов задания семантики визуальных языков был предложен собственный подход, разработанный применительно к metaCASE системе QReal. В нём можно выделить три основных момента, перечисленных ниже.

Во-первых, необходимо добавить к уже существующему в системе разделению логической и графической модели при представлении и хранении

данных новую семантическую модель. В ней будут содержаться дополнительные семантические атрибуты, процедуры, записанные либо на известном текстовом языке, либо на специальном языке, созданном для блок-схем [6], над семантическими и логическими атрибутами. Данные процедуры являются возможным поведением элемента, а вместе с семантическими атрибутами они будут составлять семантику элемента.

Данная модель должна определяться в специальном редакторе, похожем на редактор для задания метамодели языка, основываясь на метамодели исходного языка. По ней должен динамически генерироваться, компилироваться и загружаться в систему соответствующий редактор семантики, содержащий элементы исходного языка вместе с их семантикой, а также автоматически добавляющий фиксированный набор новых элементов и новых атрибутов к уже существующим элементам.

Из новых атрибутов можно выделить метку семантического статуса элемента в правиле, по которой в дальнейшем будет определяться, нужно ли этот элемент удалить, создать или оставить без изменений. В набор новых элементов входят: элемент «Rule», связь «Replacement» и различные унификаторы.

Элемент «Rule» является контейнером для всех элементов редактора семантики и будет целиком содержать ровно одно семантическое правило, т.е. правило преобразования графов и реакцию на него, описание которой будет дано позже. Направленная связь «Replacement» означает замену в модели одного элемента на другой. Унификаторы же являются обобщёнными элементами и бывают двух типов: узла и связи. Сравнение элемента модели с ними всегда возвращает истину.

Во-вторых, процесс визуализации интерпретации будет происходить при помощи применения к исходной модели различных правил преобразований графов. Эти правила задаются в редакторе семантики, описанном выше, и применяются по принципу: первое правило, для которого нашёлся хотя бы один соответствующий подграф в модели, применяется к первому такому подграфу. Интерпретация будет происходить до тех пор, пока применимо хотя бы одно правило.

В общих чертах, правила преобразования графов включают в себя поиск подграфа в модели с последующим удалением, заменой или созданием новых узлов и ассоциаций. Принцип работы такого процесса напоминает выполнение алгоритмов Маркова на строках.

В-третьих, для интерпретации поведения объектов и организации их взаимодействия при применении правил было введено понятие «реакция на правило», представляющее собой некий код (на том же языке, что и поведение отдельного элемента, описанное выше), который будет исполнен после преобразования модели согласно правилу и будет изменять различные атрибуты элементов правила, а также, возможно, делать что-то более сложное, что позволяет выбранный язык.

2.3. Прототип

На основе подхода, описанного выше, в систему QReal в виде плагина был встроен прототип визуального интерпретатора моделей. Этот плагин состоит из трёх основных компонент.

Первая компонента отвечает за создание корректного редактора семантики. К заданной метамодели визуального языка добавляются нужные новые элементы и атрибуты, полученный результат сохраняется на диск и генерируется файл с настройками сборки. После происходит непосредственный процесс сборки и динамической загрузки нового редактора в работающую систему. Все эти действия происходят в автоматическом режиме.

Особенностью полученного редактора семантики является наличие специального элемента и связи, обозначающих текущее положение потока исполнения. При интерпретации вместо отрисовки этих элементов происходит просто подсветка места в диаграмме, в котором находится поток исполнения.

Вторая компонента является модулем преобразования графов. Модель, нарисованная в соответствующем редакторе, является типизированным ориентированным мультиграфом с атрибутами. В данном модуле реализован алгоритм поиска одного графа как подграфа в другом с учётом равенства типов, направлений связей и атрибутов. Также учитывается наличие потока исполнения на элементах.

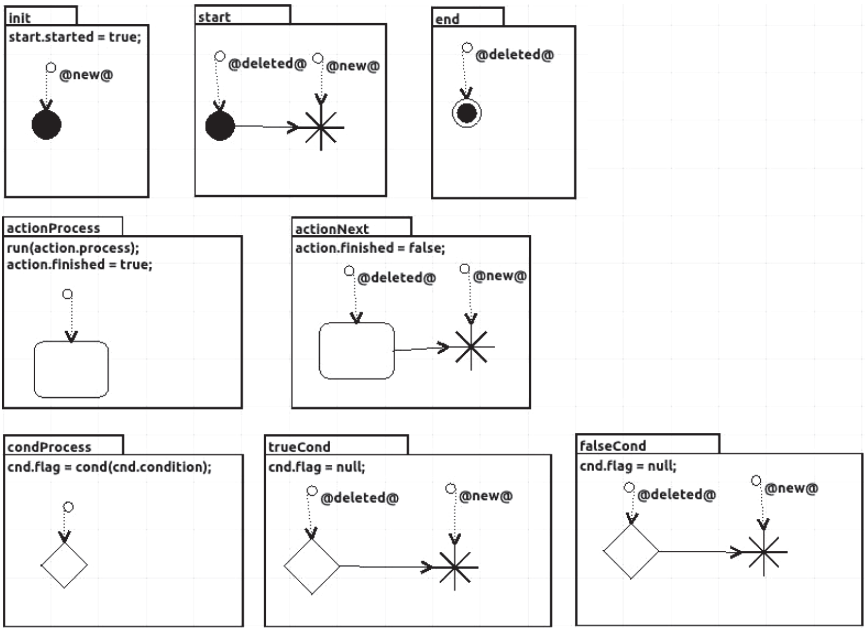


Рис. 1. Семантика языка блок-схем

Третья компонента предоставляет возможность интерпретации реакции на применение правила. В специальном атрибуте элемента «Rule» в редакторе семантики можно писать на расширении уже созданного текстового языка [6]. Расширение состоит в том, что реализована возможность как чтения, так и записи новых значений в атрибуты элементов (доступ осуществляется по имени элемента и названию атрибута).

Типичный алгоритм работы разработчика визуального языка можно представить следующим образом. В первую очередь, разработчик в специальном окне указывает расположение метамодели языка, который он хочет интерпретировать. После этого в системе автоматически генерируется и подгружается редактор семантики для этого языка. Вторым действием разработчика является создание описания семантики в этом редакторе. Потом при помощи специальной кнопки разработчик загружает эту семантику в интерпретатор, рисует диаграмму при помощи исходного визуального языка и начинает её интерпретацию, следя за информационными сообщениями в специальном окне.

Пример задания семантики для языка блок-схем и пример визуальной интерпретации по неё показаны на рисунках ниже.

Как видно из рисунка 1, при создании семантики для визуального языка можно активно использовать унификаторы, считывание и запись в атрибуты элементов, перемещение потока исполнения.

Результаты применения различных правил выводятся на экран, как показано на рисунке 2. Также происходит подсветка элемента с потоком исполнения.

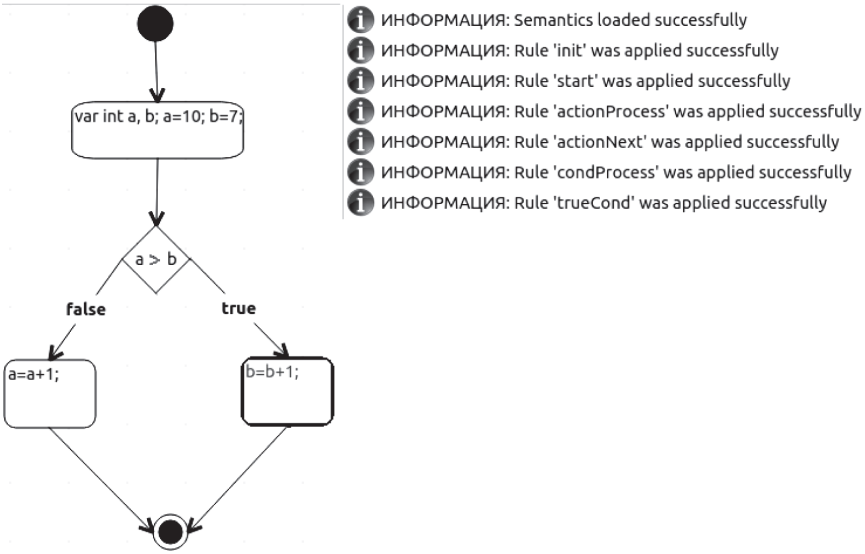


Рис. 2. Пример работы визуального интерпретатора

3. Заключение

В результате данной работы были изучены существующие подходы к визуальной интерпретации поведенческих диаграмм и был проведен анализ таких подходов на предмет применимости по отношению к системе QReal. По итогам анализа было создано собственное решение, по возможности сохранившее положительные черты рассмотренных подходов. Также был реализован прототип данного решения для системы QReal.

Анализ этого прототипа поможет оценить степень применимости данного подхода и расставит приоритеты в дальнейшем развитии реализации визуального интерпретатора и корректировки самого подхода.

Данное исследование можно продолжить по следующим направлениям: более детальная формализация созданного подхода с учётом его последующей реализации и расширение функциональности существующего прототипа.

Также возможно дальнейшее обобщение предложенного метода. Например, задание функции-веса для правил, которая будет влиять на порядок их применения, даст возможность визуализировать различные алгоритмы на графах. Выбор правила или места применения правила во время очередного шага, когда таких возможных правил или мест несколько, сделает процесс интерпретации ещё более интерактивным и наглядным.

Литература

1. *Jan Hendrik Hausmann*. Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Languages, PhD thesis, 2005, URL: http://is.uni-paderborn.de/uploads/tx_sibibtex/Dynamic_Meta_Modeling_-_A_Semantics_Description_Technique_for_Visual_Modeling_Languages.pdf
2. *Mellor Steven J., Balcer Marc J.* Executable UML, A foundation for model driven architecture // Addison Wesley, 2002.
3. http://www.devx.com/enterprise/Article/10717
4. http://se.cs.depaul.edu/ise/zoom/projects/statechart/SE690DetailedPresentation.ppt
5. *Nils Bandener*. Visual interpreter and debugger for dynamic models based on the Eclipse platform, Diploma Thesis, 2009, URL: http://is.uni-paderborn.de/uploads/tx_dsorexams/Diploma_Thesis_Nils_Bandener.pdf
6. *В. А. Поляков, Т. А. Брыксин*. Разработка визуального интерпретатора моделей в системе QReal // Материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-Запада «Технологии Microsoft в теории и практике программирования». СПб.: Изд-во СПбГПУ, 2011. С. 58.
7. *Т. А. Брыксин, Ю. В. Литвинов*. Технология визуального предметно-ориентированного проектирования и разработки ПО QReal // Материалы второй научно-технической конференции молодых специалистов «Старт в будущее», посвященной 50-летию полета Ю. А. Гагарина в космос. СПб., 2011. С. 222–225.

СИСТЕМА АНАЛИЗА РЕКОНСТРУКТИВНЫХ ХИРУРГИЧЕСКИХ ОПЕРАЦИЙ ПРИ ПОМОЩИ MICROSOFT KINECT

А. С. Ромашкин, А. Г. Петров

Кафедра Системного программирования
Математико-Механический факультет

Санкт-Петербургский государственный университет

Аннотация: Выполненная дипломная работа посвящена использованию сенсора Microsoft Kinect для анализа анатомии пациента и планирования реконструктивных операций. Разработанная система на основе Microsoft Kinect позволяет бесконтактным образом просматривать медицинские изображения пациента в операционной комнате и проводить измерения по телу пациента во время операции.

Введение

Реконструктивные операции после травм, ожогов и онкологических заболеваний позволяет вернуть человеку прежнее качество жизни. При планировании таких процедур хирург использует все доступные способы исследования анатомии пациента для достижения наилучшего с функциональной и визуальной точки зрения эффекта. Среди используемых медицинских изображений применяются магнитно-резонансные снимки, снимки компьютерной томографии, ультразвуковые и рентгеновские изображения.

Появление новых технологий машинного зрения и доступных устройств позволяет решить следующие проблемы:

- Сохранение максимальной стерильности в операционной комнате.
- Проведение анализа поверхности пациента во время операции.

В общении с хирургами клиник Санкт-Петербурга и госпиталя г. Савонлинна (Финляндия) были подтверждены указанные проблемы.

В рамках дипломной работы были реализованы следующие модули программного комплекса:

1. Модуль визуализации и управления 2D медицинскими изображениями при помощи жестов.
2. Модуль визуализации и управления 3D моделями при помощи жестов.
3. Модуль численного анализа поверхности пациента для планирования операции.

Программный комплекс находится на этапе подготовки к пилотному использованию.

Microsoft Kinect

Kinect — это игровой «контроллер без контроллера», первоначально представленный для консоли Xbox 360 (1 июня 2009) и значительно позднее для персональных компьютеров под управлением ОС Windows (1 февраля 2012) [1].

Kinect — это горизонтально расположенная коробка на небольшом круглом основании, которую помещают выше или ниже экрана [2]. Размеры — примерно 23 см в длину и 4 см в высоту. Состоит из двух сенсоров глубины, цветной видеокамеры и микрофонной решетки. Свободно распространяемое программное обеспечение осуществляет полное 3-х мерное распознавание движений тела, мимики лица и голоса.

Программной обеспечение, поставляемое вместе с контроллером, позволяет отслеживать в реальном времени скелет пользователя и различать фон от пользователя переднего плана.

Kinect открывает новую страницу для всего машинно-пользовательского взаимодействие и выпуск контроллера Kinect For Windows только подтвердили амбиции Microsoft на этот счет.

У этой сравнительно молодой технологии есть недостатки: при довольно хорошем распознавании скелета, максимальное разрешение камеры и сенсора глубины 640x480, что ограничивает точность распознавания тонких жестов и что заставила Microsoft отказаться от распознавания пальцев в текущей версии Kinect вообще.

Проблематика

Просмотр 2D\3D моделей во время операции используется и сейчас повсеместно, но при этом для поворота\листания или масштабирования используется мышь или клавиатура. Устройства ввода проходят специальную подготовку по обеспечению стерильности. Во время операции ими управляет либо сам хирург, либо ассистент.

Использование результата дипломной работы позволит:

- Исключить взаимодействие с лишними предметами во время операции.
- Упростит процесс взаимодействия хирургической команды во время операции.
- Позволит нескольким людям одновременно просматривать медицинские изображения.

При проведении реконструктивных операций хирургу необходимо знать примерный объем тканей, выбираемый для переноса на другую часть тела. Например, при закрытии раны после ожога хирургу необходим оценить и сравнить донорскую область и область куда планируется пересадка покровных тканей. Использование результата дипломной работы позволит хирургу выделять контур интересующей области на теле пациента и получать оценку площади поверхности и объема покровных тканей.

Реализация

1. *Модуль визуализации и управления 2D медицинскими изображениями при помощи жестов.*

В качестве форматов для просмотра была выбрана поддержка форматов изображений (JPG, PNG, BMP) и медицинские изображения в формате DICOM.

DICOM (англ. Digital Imaging and Communications in Medicine) — отраслевой стандарт создания, хранения, передачи и визуализации медицинских изображений и документов обследованных пациентов.

В первую очередь был реализован просмотр снимков в стиле Touch-просмотра фотографий. Миллионы людей ежедневно сталкиваются с ним, просматривая фотографии на мобильных устройствах. При проведении тестов выяснилось, что просмотр практически идентичных DICOM снимков не удобен в этом стиле. В связи с этим был реализован классический статичный простой просмотр фотографий, в которую встроен DICOM парсер.

Реализованы следующие жесты управления изображениями:

- Листание: ощутимое движение ладоней в ту или иную сторону на расстоянии >40 см от груди.
- Масштабирование: Разведение/сведение ладоней на расстоянии >40 см от груди.

2. *Модуль визуализации и управления 3D моделями при помощи жестов.*

Реализованы следующие жесты управления 3D моделями

- Вращение 3D модели: движение ладонью в ту или иную сторону на расстоянии >40 см от груди.
- Приближение и отдаление 3D модели: разведение/сведение ладоней на расстоянии >40 см от груди.

В работе используется инструментарий для работы с 3D моделями под названием HelixToolKit [3].

3. *Модуль численного анализа поверхности пациента для планирования операции.*

Сценарий работы хирургам с системой выглядит следующим образом:

- До проведения операции или во время операции хирург задает в системе толщину покровных тканей
- Во время операции хирург выделяет замкнутый контур на теле человека при помощи указательного пальца или до операции наносит разметку на тело пациента при помощи маркера.
- Программа на мониторе показывает изображение области тела, выделенный контур и численные характеристики области внутри контура.

Для реализии слежения за пальцем, требовалась соответствующая библиотека. Для начала требовалось выбрать инструментарий для работы с Kinect: либо официальная библиотека KinectForWindows SDK, либо свободная библиотека PrimeSense OpenNI SDK. У обеих библиотек есть свои достоинства и недостатки [6], но в итоге было принято решение об использовании KinectForWindows SDK.

Из-за того что KinectForWindows вышла не очень давно, оказалось что под нее практически нет готовых библиотек для слежения за пальцами. Единственная работающая библиотека Candescent NUI [7] оказалась мало пригодной для жизни, так как не использует информацию о скелете, предоставляемую Kinect SDK. Было принято решение реализовать свою библиотеку слежения за пальцами.

Нахождение контура по нарисованному фломастером на теле человека следу было реализовано при помощи несложных манипуляций цветов через EmguCV [4]. В результате нахождения внутренней части контура строится 3D поверхность из треугольников, ограниченная этим контуром, после чего вычисляется площадь этой 3D поверхности, как сумма входящих в нее треугольников. Объем вычисляется как площадь контура, умноженная на введенную глубину.

Заключение

В рамках выполненной работы были реализованы следующие модули программного комплекса: модуль визуализации и управления 2D медицинскими изображениями при помощи жестов, модуль визуализации и управления 3D моделями при помощи жестов, модуль численного анализа поверхности пациента для планирования операции. Так же была реализована собственная библиотека для распознавания движения пальцев. В дальнейшем планируется провести пилотные испытания системы, провести интеграцию с медицинскими изображениями для автоматического определения толщины покровных тканей выбранной области.

Литература

1. Kinect for Windows is now Available! | Kinect for Windows blog.
2. Dudley, Brier. E3: Microsoft Xbox throws down gauntlet with «Natal» controller, The Seattle Times (1 июня 2009).
3. http://helixtoolkit.codeplex.com/
4. http://www.emgu.com/
5. www.kinectforwindows.org
6. http://www.brekel.com/?page_id=671
7. candescentnui.codeplex.com

USABILITY В ПРОЕКТЕ QREAL: ROBOTS

Н. А. Соковикова

студентка 3 курса кафедры Системного программирования;
snnatalies@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Юзабилити — это качественный признак, который определяет, насколько интерфейс пользователя легок в использовании. Слово «юзабилити» также обозначает набор методов, служащих для улучшения интерфейса во время процесса его разработки.

Введение

Программирование — важный предмет школьной программы. Детям сложно его изучать без специальных программ и приложений, так как они не могут видеть работу своей программы, могут только абстрактно представить себе ход ее выполнения, и получить некоторый результат, по которому часто нельзя судить о правильности работы программы. Одним из первых проблему нематериальности программирования увидел информатик-психолог из MIT Сеймур Пейперт. Для ее решения он предложил язык LOGO. Результаты программирования на нем — это движение черепашки по экрану. Но даже красиво отображающийся на экране исполнитель недостаточно нагляден.

Современный уровень развития вычислительной техники позволяет создать недорогие устройства, как исполняющие внутреннюю программу, так и управляемые по беспроводной связи с персонального компьютера, и средства программирования для них. Самые распространенные на данный момент среды программирования роботов –NXT-G, входящая в стандартный комплект конструктора, и Robolab. Однако у обеих есть существенные недостатки. Так, NXT-G недостаточно функциональна и подходит только для начальных этапов обучения, а Robolab не позволяет переходить от графического представления к текстовому без смены инструментария, и, к тому же, имеет довольно высокую для российских школ стоимость.

Среда программирования роботов QReal: Robots разрабатывается на основе CASE-системы QReal, предназначенной для создания специализированных сред визуального программирования, на кафедре системного программирования СПбГУ. Она имеет ряд преимуществ перед NXT-G и Robolab»ом. QReal: Robots предоставляет возможность работать с визуальным языком программирования и просматривать сгенерированный по блок-схеме код на языке C в режиме текстового редактора. Робот может непосредственно

управляться с компьютера через Bluetooth, или на него может быть залита готовая программа. Еще одна особенность QReal: Robots — возможность выполнить программу на аналоге традиционного исполнителя — Logo-черепашки — прямо на экране компьютера. Моделируется трёхколёсная тележка с двумя ведущими колёсами. Это позволяет отлаживать программы без доступа к реальному устройству. Реализовано моделирование окружающей реальности робота, что дает возможность отлаживать с использованием модели довольно сложные программы. Однако наряду с функциональностью следует так же уделить внимание юзабилити приложения.

Юзабилити — это понятие, обозначающее итоговый уровень удобности чего-либо для использования в заявленных целях. Речь может идти о приложении, вебсайте, книге, инструменте — о чем угодно, созданном человеком. Юзабилити включает в себя анализ потребностей пользователей и исследование принципов, по которым объект воспринимается эффективным или приятным на вид.

Сложные программные продукты находят свое место в жизни людей, и в то же время, рынок заполнен соревнующимися брендами. Это сделало юзабилити более популярным и признаваемым направлением, поскольку компании видят преимущества разработки, ориентированной на удобство пользователя.

Для проекта QReal: Robots вопрос юзабилити еще более актуален, чем обычно, так как это среда для обучения, и удобство ее использования может формировать у детей мнение об изучаемом предмете и в дальнейшем повлиять на выбор профессии. В то же время, дети более требовательны, чем взрослые, поэтому чтобы их заинтересовать новым, пока еще им непривычным средством программирования, необходимо, чтобы его интерфейс был понятен при минимальном времени, потраченном на его изучение.

Таким образом, исследование и улучшение юзабилити увеличивает шанс программы на успех.

Оценка юзабилити

Чтобы улучшить юзабилити текущей версии программы, нужно в первую очередь оценить ее и найти недочеты. Фактически, нужно было определить, насколько программа соответствует следующим критериям:

1. обучаемость — насколько просто пользователям освоить интерфейс;
2. эффективность — как быстро они смогут выполнять различные задания, после знакомства с интерфейсом;
3. как легко пользователи могут восстановить свои навыки работы с приложением, не используя его некоторое время;
4. ошибки: как много делают ошибок, насколько те серьезны и насколько сложно их исправить;
5. удовлетворенность: насколько приятно использовать данный интерфейс.

Существует множество методов изучения юзабилити, но самый основной из них — это метод тестирования дизайна пользователями.

Он включает следующие компоненты:

- выбор пользователей, наиболее характерных для приложения (в случае QReal: Robots это школьники и их преподаватели информатики);
- выполнение пользователями наиболее типичных базовых задач;
- важно, чтобы задания выполнялись без подсказок или наводящих вопросов;
- самое важное — проводить тестирование отдельно с каждым пользователем;

Каждый пользователь должен сам решать ставшие перед ним проблемы, так как помогая или привлекая внимание к какому-то определенному элементу на экране, можно серьезно исказить результаты теста.

Существует большое разнообразие методов тестирования. Некоторые из них предусматривают слежение за зрачками пользователя во время выполнения задания. По результатам наблюдений составляются тепловые или туманные карты экрана. На тепловых картах самыми «горячими» цветами (красный, оранжевый) выделяются области, на которых дольше всего задерживается взгляд пользователей. На туманных картах участки экрана, на которые не обратили внимания, видны как бы через плотный туман, в то время как участки, которые привлекли к себе взгляды, видно без всякой дымки. Другой пример -«mouse tracking» — слежение за передвижением курсора по экрану. Однако по рекомендации специалиста по инженерной психологии был выбран метод обычного тестирования.

После выполнения задания пользователю предлагается ответить на вопросы анкеты. Часто используются следующие опросники:

1. QUIS (Questionnaire for User Interface Satisfaction) — анкета, разработанная в университете Мэриленда и содержащая 27 вопросов (например, «Ваша реакция на приложение в целом»). На каждый вопрос нужно ответить числом от 1 («Ужасно!») до 10 («Великолепно!»);
2. Words — это список из 118 слов, напротив каждого из которых оставлено место для метки. Пользователя просят отметить слова, наиболее точно описывающие его впечатление от работы с приложением. Можно выбрать любое количество слов;
3. SUS (System Usability Scale) — этот опросник состоит из десяти утверждений, каждое из которых нужно оценить от 1 до 5, где 5 означает полное согласие с утверждением, а 1 — его совершенное отрицание.

Чтобы не утомить детей длинным списком вопросов, была выбрана третья анкета.

Тестирование проводилось на открытых соревнованиях Санкт-Петербурга по робототехнике. По проекту был сделан стендовый доклад и некоторых заинтересовавшихся им участников и гостей соревнований просили решить несложную задачу с помощью приложения (например, составить программу,

чтобы нарисованный робот доехал до стенки и остановился), после чего пройти анкетирование и рассказать о том, что понравилось, что не понравилось, что вызвало затруднения и пр. Все участники тестирования справились с задачей без больших проблем, и в целом неплохо отзывались о QReal: Robots. Всего было опрошено 8 человек. Средние результаты анкеты — 76 баллов. Это лучше среднего (68 баллов), но «хороший» интерфейс — это 85 баллов и более.

Тестирование помогло выявить некоторые мелкие замечания к интерфейсу, работа над которыми должна хорошо сказаться на юзабилити проекта.

Как было сказано выше, QReal: Robots разрабатывается на основе CASE-средства QReal. Уже после проведения тестирования в него были внесены обновления из QReal»а. Такие изменения могут сказаться на юзабилити. Специалисты утверждают, что вместо одного крупного исследования лучше провести несколько мелких тестов и вносить изменения в дизайн после каждого из них: так можно сразу исправлять обнаруженные ошибки, поэтому в ближайшее время планируется вновь оценить наше приложение.

Заключение

Среда визуального программирования роботов QReal: Robots, разрабатываемая на основе CASE-средства QReal, имеет ряд преимущест перед своими ранее появившимися конкурентами, такими как NXT-G, RoboLab и RobotC. Однако, это новый и непривычный продукт, поэтому чтобы QReal: Robots занял свое место в мире робототехники, необходимо сделать его одновременно эффективным и удобным в использовании, создать интерфейс, понятный практически интуитивно. Все эти показатели включает в себя юзабилити. Чтобы их улучшить, нужно для начала выявить недостатки существующего решения. Выбрав подходящую методику оценки и анкету, мы провели юзабилити-тестирование на городских соревнованиях по робототехнике. С помощью тестирования удалось не только оценить юзабилити текущего интерфейса, но и выявить некоторые замечания, работа над которыми должна улучшить ее. Чтобы увидеть результат этой работы и внесения изменений из QReal, планируется провести новое тестирование в ближайшее время.

Таким образом, было проведено тестирование, по результатам которого были составлены некоторые задачи для улучшения интерфейса. Планируется продолжать действовать таким образом (тестирование, затем работа над полученными задачами) для улучшения юзабилити конечного продукта.

Литература

1. *Брыксин Т. А., Литвинов Ю. В.* Среда визуального программирования роботов QReal: Robots // Материалы международной конференции «Информационные технологии в образовании и науке». Самара. 2011. С. 332–334.
2. *Нильсен Я.* «Элементарные основы юзабилити».

СРЕДА ПРОГРАММИРОВАНИЯ QREAL: ROBOTS

М. В. Тихонова

студентка 245 группы математико-механического факультета;
maria_tikhonova@list.ru

Литвинов Ю. В.

ст. преп. кафедры системного программирования

Санкт-Петербургский государственный университет

Аннотация: В данной статье приводится обзор существующих сред программирования роботов и описывается разрабатываемая среда QReal: Robots. В ней рассматриваются задачи, возникшие при апробации среды в реальных условиях, и их решения.

Введение

QReal: Robots — среда программирования роботов, система для обучения основам программирования и кибернетики. Она базируется на CASE-системе QReal, которая является средством создания визуальных языков. QReal позволяет быстро создавать предметно-ориентированные визуальные языки, задавать форму используемых фигур и задавать правила генерации кода по диаграмме. Было решено создать визуальный язык программирования роботов, для чего удобно было использовать технологию QReal. QReal: Robots — это одно из применений QReal.

Теперь следует пояснить, как системы программирования роботов применяются в образовании. Идея использования реального исполнителя в обучении программированию и кибернетике возникла довольно давно. Обусловлено это тем, что изучение программирования сложно именно из-за того, что объекты, с которыми приходится иметь дело обучающемуся, абстрактны и нематериальны, следовательно, их трудно сравнивать и оценивать. Но нематериальный исполнитель, даже если он отображается на экране, все еще недостаточно нагляден. К тому же, роботы интересны школьникам: процесс сборки робота очень занимателен и увлекает их, им интересно наблюдать, как реальный исполнитель выполняет заданные команды (едет вперед, останавливается, поворачивает). А материал лучше усваивается, когда обучающиеся увлечены процессом; таким образом, понятна польза от применения робототехники в обучении программированию.

Одним из первых, кто оценил проблему сложности работы с нематериальным исполнителем, был Сеймур Пейперт, психолог-информатик из MIT. Он разработал язык программирования LOGO, который позволял видеть результаты работы программы непосредственно. На экране рисовался не-

материальный исполнитель — черепашка; она передвигалась по экрану в соответствии с заданными ей командами. Кроме того, Пейперт изначально использовал в экспериментах механическую черепашку, которая позже была заменена на рисованную.

Что касается отечественного подхода к обучению информатике, здесь основоположником считается академик Ершов А. П., вместе с Г. А. Звенигородским и Н. А. Юнерман создавший методики и инструментальные средства, в которых нематериальный исполнитель выполнял команды, заданные пользователем (например, Робик и Рапира).

На сегодняшний день уровень развития вычислительной техники позволяет создавать устройства, которые могут как исполнять внутреннюю программу, так и управляться с компьютера по беспроводной связи. Конструкторы на основе контроллеров, поддерживающих эту функциональность, давно внедряются в школьное образование. В частности, одним из наиболее часто используемых конструкторов для обучения робототехнике является Lego Mindstorms.

В стандартном комплекте конструктора поставляется среда программирования роботов NXT-G, однако у нее имеется ряд недостатков, главным из которых является ее ограниченная функциональность (не поддерживается сложная математика). Поэтому преподаватели отдают предпочтение среде программирования Robolab, которая позволяет создавать гораздо более сложные программы, содержащие нетривиальные математические выражения. Но в Robolab недостаточно полный и качественный перевод на русский язык, он обладает устаревшим и недружественным пользовательским интерфейсом. К тому же, Robolab является платной средой программирования.

QReal: Robots является свободно распространяемым средством программирования роботов. В нем есть такие особенности, как пошаговая отладка программы и переход от диаграммы поведения робота к текстовому представлению, что актуально для тех, кто больше интересуется программированием, чем кибернетикой. На момент начала нашей работы над этим продуктом большая часть функциональности уже была реализована, но была не до конца доработана и содержала большое количество ошибок. Наша задача состояла в том, чтобы исправить эти ошибки и довести продукт до состояния, в котором его можно было бы внедрять в школах. Для этого были собраны и проанализированы требования к продукту, выдвинутые руководителем робототехнического кружка в лицее №239, школьниками, обучающимися там, и участниками городских соревнований по робототехнике. Эта статья описывает текущие возможности среды QReal: Robots, проблемы, возникшие при ее внедрении, и их решения.

Обзор существующих сред программирования

На сегодняшний день существует множество средств программирования роботов. Они делятся на графические (Robolab, NXT-G) и текстовые, основанные на существующих языках программирования (RobotC, NXC). Наиболее часто используемыми системами для программирования роботов являются Robolab, NXT-G и RobotC.

NXT-G — графическая среда программирования роботов. Создание программы для управления поведением робота в ней похоже на рисование блок-схемы: с помощью имеющихся блоков рисуется диаграмма и задаются характеристики поведения робота. NXT-G очень проста для освоения и использования из-за своей наглядности, но ее существенным недостатком является ее недостаточная функциональность, например, отсутствие поддержки сложных математических выражений.

Robolab — тоже графическая среда программирования, самая распространенная в школах и вузах для преподавания кибернетики. С ее помощью можно создавать гораздо более сложные программы с использованием нетривиальных математических выражений, но она имеет ряд недостатков, таких как отсутствие пошаговой отладки и значительная стоимость.

RobotC — текстовая среда программирования, позволяющая создавать программы для управления поведением робота с помощью языка программирования C. У нее есть два режима работы: базовый режим и режим для специалистов (в базовом режиме некоторая функциональность отсутствует). RobotC имеет мощный интерактивный отладчик в режиме реального времени; с помощью этой среды можно создавать сложные и эффективные программы, но текстовое программирование может оказаться недостаточно наглядным для школьников.

QReal: Robots

QReal: Robots разрабатывается силами сотрудников и студентов кафедры системного программирования. Система постоянно развивается с учетом замечаний и пожеланий преподавателей школ и вузов. На момент начала нашей работы над средой QReal: Robots в ней уже была реализована основная функциональность для управления роботом. Система позволяла создавать графические программы в виде последовательности блоков, соединенных линиями управления, и исполнять эти программы на компьютере, посылая роботу команды через Bluetooth-интерфейс. Были реализованы блоки, отвечающие за аппаратную часть — управление датчиками цвета и нажатия, ультразвуковым датчиком расстояния, управление моторами и динамиками, а также блоки, задающие логику работы программы — условные операторы, проверяющие показания датчиков или значения переменных, циклы и средства для работы с таймерами. Была осуществлена поддержка сложных мате-

матических выражений, состоящих из переменных, констант, арифметических операций, тригонометрических функций и значений датчиков.

Кроме того, важной особенностью QReal: Robots является наличие двухмерной модели робота, способной исполнять те же программы, что и реальный робот. Она может помочь при отладке, позволяя отлаживать программу без доступа к реальному роботу. Двухмерная модель в QReal: Robots, по сути, повторяет функциональность черепашки LOGO, но, кроме этого, она поддерживает сенсоры, и в этом заключается ее существенное преимущество.

Возникшие задачи

Тем не менее, чтобы начать внедрять QReal: Robots, требовалось решить целый ряд задач, о которых и пойдет речь далее. Задачи, в основном, были связаны с недостаточностью поддержки некоторых аппаратных компонентов робототехнического конструктора, поскольку на первых этапах разработки задача поддержки всех видов датчиков для NXT во всех режимах и не ставилась. Робот должен был устойчиво функционировать с теми датчиками, что входили в стандартную поставку конструктора NXT 2.0, однако же, для решения реальных задач этого оказалось мало. Кроме того, при столкновении с реальными задачами выявились и другие слабые места продукта — некоторые неточности и концептуальные ошибки, допускаемые генератором кода для ОС nxtOSEK, связанные, опять же, с необходимостью быстро представить работающее на некоторых частных случаях решение. Таким образом, наша задача состояла в том, чтобы поддержать необходимые для решения реальных задач датчики — например, для реализации алгоритма движения по линии нам понадобились датчики цвета. Оказалось, что nxtOSEK не поддерживает датчики цвета, и пришлось использовать вместо них датчики света, не умеющие распознавать цвета, а распознающие только интенсивность света, но поддерживаемые nxtOSEK. Поддержку датчиков света в QReal: Robots тоже надо было реализовать. Также нужно было исправить ошибки в генерации кода; в частности, нужно было поддержать генерацию функций-инициализаторов (функций, в которых инициализируются сенсоры), генерацию переменных в начале программы и сделать так, чтобы код генерировался в соответствии с инициализированными сенсорами. К тому же, выяснилось, что заливка программы на робота по USB для выполнения непосредственно на роботе не всегда работает корректно, потребовалось поправить и это.

Городские соревнования по робототехнике и фестиваль в Москве

Для апробации разрабатываемого нами продукта мы также приняли участие в городских соревнованиях по робототехнике. Именно здесь пришлось столкнуться с реальными задачами, такими, как реализация алгоритма

для движения по линии. Движение по линии — один из видов соревнований; в нем робот должен проехать по кривой линии за возможно меньшее время. Для участия в этом соревновании мы должны были собрать робота и реализовать для него алгоритм пропорционально-дифференциального регулятора. Пропорционально-дифференциальный регулятор (ПД-регулятор) — алгоритм, считывающий показания с двух сенсоров света и считающий две величины: во-первых, разницу между показаниями сенсоров, чтобы узнать, насколько робот отклонился от линии, во-вторых, разницу между текущим показанием и предыдущим. ПД-регулятор гораздо лучше обычного пропорционального регулятора, который использовали многие, но наш робот, собранный из конструктора LEGO NXT, едва ли мог соперничать с роботами, специально сконструированными для езды по линии.

Также проект QReal: Robots был представлен в творческой категории, где занял второе место. Проектом многие заинтересовались, что, в конечном счете, и было целью нашего выступления на соревнованиях.

Кроме того, мы съездили на Робототехнический фестиваль в Москве, где представили свой проект в творческой категории. Для того, чтобы сделать представление проекта более интересным, было принято решение продемонстрировать реализацию алгоритма балансировки сегвея. Сама задача балансировки сегвея является хоть и стандартной в робототехнике, но тем не менее труднореализуемой. Трудность заключается в том, что, во-первых, алгоритм достаточно сложен, а во-вторых, программу нельзя отлаживать из-за того, что данные по Bluetooth и по USB передаются с задержкой, поэтому робот не успевает реагировать на изменение наклона и не балансирует. Нам нужно было реализовать алгоритм в QReal: Robots, для чего потребовалось поддержать новые блоки (в частности, блоки инициализации и балансировки). Получившаяся в результате диаграмма поведения робота достаточно проста, таким образом, было показано, что даже столь сложные задачи, как балансировка сегвея, с помощью QReal: Robots могут быть довольно просто реализованы.

Также мы приняли участие в соревнованиях по движению по линии. Сделано это было для того, чтобы продемонстрировать, что можно использовать QReal: Robots для написания алгоритмов, требующихся школьникам на соревнованиях, и что создание программы настолько наглядно, что даже неспециалисты могут легко справиться с этой задачей.

Заключение

Результатом нашей работы стало улучшение системы QReal: Robots и ее апробация на реальных задачах. Мы представили проект на городских соревнованиях и всероссийском робототехническом фестивале в Москве. Мы посетили робототехнический кружок в лицее 239 и выяснили потребности занимающихся там школьников. В ходе работы были решены такие задачи,

как совершенствование генератора кода по диаграммам, исправление ряда ошибок, связанных с заливкой программ на робота, добавление новых блоков, потребовавшихся для реализации алгоритма балансировки сегвея. По результатам работы будет выпущена новая версия.

Литература

1. *Брыксин Т. А., Литвинов Ю. В.* Среда визуального программирования роботов QReal: Robots // Материалы международной конференции «Информационные технологии в образовании и науке». Самара, 2011. С. 332–334.
2. *С. А. Филиппов*. Робототехника для детей и родителей. СПб.: Наука, 2010.
3. QReal: Robots, URL: http://se.math.spbu.ru/SE/qreal

ИДЕНТИФИКАЦИЯ ПРОЦЕССОВ ПРОГРАММ ПО ВНЕШНИМ ПРИЗНАКАМ

А. Р. Ханов

awengar@gmail.com

Руководитель: ***М. В. Баклановский***

Санкт-Петербургский государственный университет

Анализ программ является важнейшей областью исследования. Он включает в себя получение различной информации из кода программы, из самой программы в процессе ее работы. Такой анализ производит как сама операционная система, например, для планирования потоков или свопа, так и сторонние программы для поиска ошибок, анализа производительности, тестирования, для опознавания вредоносного кода.

Существуют методы построения моделей программ по исходному коду [1], предназначенные для проверки эквивалентности программ, но эта задача все еще остается нерешенной. В исследованиях, посвященных опознанию вредоносных кодов, строятся модели программ по их поведению в системе для проверки на наличие в них угрозы безопасности. Так в [2] строятся признаки для опознавания вредоносных программ по набору API-вызовов, производимых в процессе ее работы.

Нашей целью является идентификация произвольного процесса на основе его внешних взаимодействий. Для работы были взяты списки системных вызовов, которые производит каждый поток процесса. Возможность идентификации процесса лишь по такой информации не очевидна и требует детальной проработки алгоритма и его проверки на реальных данных.

Для перехвата номеров системных вызовов, производимых программой в процессе работы, был использован метод, описанный в [3]. За секунду программа может производить от нескольких сотен до тысяч таких вызовов. Их последовательность обладает высокой статической энтропией (от 4 до 10), значит, мы имеем данные с высокой избыточностью и хорошей сжимаемостью. По методу PPM была посчитана энтропия каждого вызова в некотором контексте фиксированной длины. Было установлено, что вызовы являются достаточно предсказуемыми по контексту, то есть по окружающим их вызовам.

Все это указывает на существование последовательностей, названных нами термы, которые повторяются в потоках программ много раз. Они являются некоторыми смысловыми единицами длиной большей, чем один вызов. Термы могут быть как очень длинными, порядка нескольких тысяч вызовов, так и очень короткими, одни из них могут целиком включаться в другие, они накладываются друг на друга, на самих себя, они могут быть степенями более коротких термов. Из всего множества термов необходимо выделить те, которые могут помочь распознать поток. Это должны быть достаточно длинные

и частые термы. Здесь можно строить алгоритмы, с большей или меньшей эффективностью подходящие для решения задачи. Был использован следующий алгоритм: среди всех термов с уникальным числом встречаемости в потоке не меньше, чем фиксированное число раз, выбираем самые длинные. При этом мы заранее можем указать максимальное число необходимых нам термов и их минимальную длину. Для каждого терма, полученного таким образом, фиксируем позиции, на которых он встретился, и получаем средний и максимальный период его появления.

Теперь, получив очередной список системных вызовов одного потока, мы сможем, проведя поиск термов, установить, был ли этот поток изучен нами ранее. Количество вызовов, необходимое для опознания, можно получить из периода встречаемости термов. Имея множество термов для некоторого набора потоков программ, мы можем отнести распознаваемый список системных вызовов потока к той программе, чьи термы в нем встречаются в большем количестве и чаще.

В результате тестов было выявлено, что такие термы существуют, и они позволяют идентифицировать процессы. Для 12 типичных оконных программ операционной системы Windows 7 были получены последовательности системных вызовов, сделанных ими за период в несколько сотен секунд. Было извлечено до 2000 термов для самого длинного потока каждого процесса. Длины этих потоков, были порядка миллиона вызовов. С помощью этих последовательностей были распознаны списки системных вызовов тех же программ, записанных позднее на новых экземплярах процессов. Из каждого списка были взяты по 30 000 начальных вызовов. Часть результатов показана в таблице.

Процесс, термы	Calc	Devenv	Explorer	IE	Mines	Mozilla	Notepad
CALC	964 [360]	103 [93]	37 [11]	77 [32]	61 [134]	32 [11]	86 [103]
DEVENV	3 [62]	145 [585]	2 [45]	3 [47]	6 [71]	0 [0]	23 [98]
EXPLORER	105 [55]	192 [49]	85 [28]	176 [45]	62 [23]	42 [21]	87 [41]
IE	3 [29]	8 [34]	1 [27]	322 [110]	1 [23]	0 [0]	0 [0]
MINES	1 [71]	0 [0]	0 [0]	0 [0]	400 [722]	0 [0]	0 [0]
Mozilla	0 [0]	0 [0]	0 [0]	0 [0]	0 [0]	174 [1426]	0 [0]
NOTEPAD	105 [80]	148 [65]	14 [47]	50 [39]	45 [60]	13 [18]	224 [165]

В левой колонке указаны процессы, по которым были получены термы. Для каждого из них указано, сколько термов встречается хоть раз в вызовах самых длинных потоков программ справа, в скобках указан размер самого длинного из таких термов. В ситуациях, когда два процесса плохо распознаются по количеству термов, разрешить ситуацию помогает наибольшая длина терма или общее число встречаемости. Так для devenv правильное опозна-

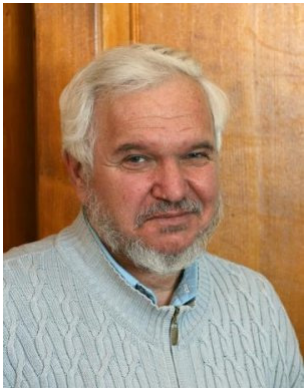
вание не происходило исходя лишь из количества термов, так как в explorer было большее число его термов. В таком случае мы можем рассмотреть также максимальную длину встретившегося терма, а для devenv она была наибольшей. Если мы возьмем не 30000, а 60000 вызовов из потоков для их опознания, мы сможем различить эти два процесса только по числувстретившихся термов. Мы можем изменять критерий распознавания, основываясь на количестве появлений, числе термов в потоках, максимальной длине терма. Но в любом случае это показывает, что опознавание процессов по спискам системных вызовов самых длинных потоков возможно и полученные нами цепочки, термы, позволяют это сделать. Кроме того мы используем информацию, которую можем извлекать с большой скоростью и не используем алгоритмов высокой сложности.

Идентификация процессов может иметь широкое применение. Например, можно зафиксировать все процессы, которые могут быть запущены в системе или выделить те, которые нужно запретить. Само множество термов в дальнейшем будет являться объектом изучения, так как оно фактически составляет смысловые единицы работы программы.

Литература

1. *Р. И. Подловченко, Н. Н. Кузюрин, В. С. Щербина, В. А. Захаров*. Использование алгебраических моделей программ для обнаружения метаморфного вредоносного кода.
2. *M. Ahmadi, A. Sami, H. Rahimi, B. Yadegari*. Iterative System Call Patterns Blow the Malware Cover // Security for The Next Generation 2011.
3. *Одеров Р. С., Тенсин Е. Д.* Способы размещения своего кода в ядре ОС Microsoft Windows Server 2008 // Сборник трудов межвузовской научно-практической конференции «Актуальные проблемы организации и технологии защиты информации», СПб, 2011 год. Стр. 100–102.

Фундаментальная информатика



Косовский
Николай Кириллович

д.ф.-м.н., профессор
заведующий кафедрой информатики СПбГУ

Герасимов
Михаил Александрович

к.ф.-м.н., доцент кафедры информатики СПбГУ

ПОЛИНОМИАЛЬНЫЙ ПО ВРЕМЕНИ МЕТОД НАХОЖДЕНИЯ ПРИБЛИЖЕННОГО РЕШЕНИЯ ЗАДАЧИ О РАЗБИЕНИИ С ГАРАНТИРОВАННОЙ ОЦЕНКОЙ ТОЧНОСТИ

М. А. Герасимов

доцент кафедры Информатики;
ge@star.math.spbu.ru

Санкт-Петербургский государственный университет

Аннотация: Рассматривается полиномиальный по времени четырехфазный алгоритм приближенного поиска разбиения множества целых неотрицательных чисел на два приблизительно равных по весу подмножества. Предлагаемый алгоритм является в некотором смысле продолжением таких алгоритмов, как алгоритм Карлмаркара — Карпа [5] и реализует поиск приближенного решения для задачи о разбиении множества неотрицательных целых чисел [2].

Детерминированная машина Тьюринга

Для анализа алгоритма рассматривается одноленточная, одноголовочная машина Тьюринга с входной и выходной лентами [1]. Предполагается, что входные данные (рассматриваемое множество натуральных чисел) записаны на одной ленте, обрабатываются на рабочей ленте и результат записывается на выходной ленте. При работе машина Тьюринга использует алфавит, состоящий их четырех символов {#,b,0,1}. Результатом работы алгоритма считается битовая последовательность, кодирующая полученное разбиение исходного множества натуральных чисел. В дальнейшем будем считать, что входные данные (множество натуральных чисел) записаны в виде битовой последовательности на входной ленте между двумя маркерами «#». В качестве разделителя чисел используется пустой символ «b». Считывание второго маркера означает конец цепочки входных данных. Входная лента позволяет считывать входные данные произвольное количество раз. Выходная лента позволяет только записывать полученный результат вычисления в виде последовательности символов рабочего алфавита. Каждый символ выходной цепочки записывается только один раз и больше не изменяется.

Задача о разбиении

В теории алгоритмов задача о разбиении относится к классу NP-полных [2]. В самой простой формулировке она сводится к вопросу: может ли данное множество неотрицательных целых чисел быть разбито на два под-

множества таким образом, что сумма всех чисел одного подмножества равна сумме всех чисел другого. Имеются и другие эквивалентные формулировки данной задачи, например, можно ли из заданного множества неотрицательных целых чисел выделить такое подмножество, чтобы сумма составляющих его элементов была равна точно половине суммы всех элементов исходного множества. Эта задача — упрощенная версия хорошо известной, также NP-полной задачи о рюкзаке. Несмотря на то, что обе эти задачи имеют псевдополиномиальную временную сложность, настоящего полиномиального алгоритма для них до сих пор не известно.

Наряду с данной широко известной формулировкой задачи о разбиении, существуют разнообразные обобщения задачи. Одно из этих обобщений, практически важное для задач планирования, заключается в том, что исходное множество разбивается не на два подмножества, а на большее их число (но не менее двух). В этом случае, исходное множество требуется разбить таким образом, чтобы сумма чисел каждого подмножества была равна одной и той же величине. Эта задача также может быть сведена к поиску такого подмножества исходного множества, чтобы сумма входящих в него чисел была равна одной K-той части от суммы всех чисел исходного множества. В отличие от классического случая задачи, этот случай даже не псевдополиномиален.

Другим направлением обобщения задачи о разбиении является задача нахождения приближенного разбиения исходного множества. Это значит, что нужно найти такие два подмножества исходного множества, объединение которых давало бы исходное множество, а разность сумм входящих туда чисел была бы минимальна. Если разбиение есть, то оно и будет одним из решений этой задачи. Но, если разбиения (в классическом виде) нет, то следует найти точное приближение, т. е. такое, которое будет давать минимальную разность между суммами подмножеств.

Формализация постановки задачи

Рассматриваемая реализация алгоритма поиска разбиения использует список входных элементов, занумерованных натуральными числами $X = \{x_1, x_2, \ldots, x_M\}$ и имеющих положительные веса $\{w_1, w_2, \ldots, w_M\}$ соответственно. Обозначим через $W(x_i)$ вес i-го элемента, а через $W(X)$ суммарный вес элементов множества X:

$$W(x_i) = w_i,$$

$$W(X) = \sum_{i \in (1:M)} w_i.$$

Пусть множество X разбито на два дизъюнктных подмножества X_1 и X_2:

$$X_1 \subset X, \quad X_2 \subset X,$$

$$X_1 \cap X_2 = \varnothing, \quad X_1 \cup X_2 = X.$$

Согласно [2] весом разбиения назовем величину

$$F(X_1, X_2) = \max \{W(X_1), W(X_2)\}$$

В случае идеального приближения, требуется найти такое разбиение множества X на два подмножества X_1^* и X_2^*, что $F(X_1^*, X_2^*) = \min\limits_{\{X_1, X_2\}} \{F(X_1, X_2)\}$.

Понятно, что при $F(X_1^*, X_2^*) = 0$ разбиение X_1^* и X_2^* точное, а при $F(X_1^*, X_2^*) > 0$ – приближенное. Нахождение идеального приближения $\{X_1^*, X_2^*\}$ для данного множества X и величины $F(X_1^*, X_2^*)$ – NP-полная задача [3]. Рассматриваемая работа посвящена нахождению некоторого разбиения X_1, X_2 и значения $\Delta = F(X_1, X_2)$ достаточно близкого к $F(X_1^*, X_2^*)$ за полиномиальное число шагов на описанной машине Тьюринга.

Алгоритм

Предлагаемый алгоритм нахождения разбиения работает в четыре этапа.

На первом этапе происходит сортировка исходного множества элементов в соответствии с их весами в порядке их убывания, а также проверка задачи на тривиальность (проверяется, что вес первого элемента меньше суммарного веса остальных элементов).

На втором этапе происходит построение списка «дополняющих» элементов по следующему алгоритму.

Шаг 1. Если число элементов текущего списка четное, то переходим к шагу 2. Если число элементов нечетное, то добавляем к списку один элемент, с весом, равным весу наименьшего в списке и помечаем его как «дополняющий».

Шаг 2. Строим новый список элементов по следующему правилу: каждый новый элемент x_i' получается из исходного элемента x_i путем уменьшения его веса на минимальный в исходном списке. Бывший минимальным по весу элемент x_M исключается из списка. Таким образом, получаем список, в котором по крайней мере на один элемент меньше. Переходим к шагу 3.

Шаг 3. Если в списке не осталось элементов, то заканчиваем первый этап. В противном случае переходим к шагу 1.

На следующем этапе, на основании имеющихся данных, строим точное разбиение объединенного множества элементов (включая «дополняющие»). При этом, элементы с нечетными номерами попадают в левое подмножество, а элементы с четными в правое.

На четвертом этапе находим такой элемент левого подмножества, вес которого равен половине суммы всех «дополняющих» элементов. Если это

удается, то помещаем этот элемент в правое подмножество, а «дополняющие элементы» — в левое. Выбрасываем все «дополняющие элементы». Полученное разбиение и будет приближением к идеальному разбиению.

Теорема 1. О сложности обратного алгоритма. Описанный алгоритм может быть реализован на одноленточной, одноголовочной машине Тьюринга с входом и выходом за $O(n^4)$ число шагов, где n — длина входных данных.

Теорема 2. Оценка погрешности. Сумма всех отброшенных «дополняющих» элементов рассмотренного четырехфазного алгоритма Δ не превосходит величины

$$\max_{1 \le i \le M} \{w_1, ..., w_M\} \le \frac{W}{4}.$$

Замечание. В общем случае, когда веса элементов распределены достаточно случайно, соотношение $\frac{m}{N}$ максимальной длины битового представления m и количества входных элементов N близко или меньше 1, $\max_{1 \le i \le M} \{w_1, ..., w_N\} \le 2^m \le 2^{\sqrt{n}-1}$, где n — длина входных данных.

В частном случае, когда все веса ограничены некоторой константой A, погрешность алгоритма также ограничена этой константой.

Литература

1. *Минский М.* Вычисления и автоматы. М: Мир, 1971.
2. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М: Мир, 1982.
3. *Fischetti M., Martello S.* Worst-case analysis of the differencing method for the partition problem // Math. Programming. 1987. V. 37, No. 1. Pp. 117–120.
4. *Boettcher S.* and *Mertens S.* Analysis of the Karmarkar-Karp differencing algorithm // Tech. report UCB/CSD 82/113 Comp. Sci. Div. Univ. of Calif., 1982.
5. *Hayes B.* The easiest hard problem //American Scientist 90–113, 2002.

ИЗМЕНЕНИЕ ХАРАКТЕРИСТИК ИНФОРМАЦИОННОЙ СИСТЕМЫ НА ОСНОВЕ СУБД MYSQL 6.0 ПРИ РАЗЛИЧНЫХ ВЕРОЯТНОСТНЫХ РАСПРЕДЕЛЕНИЯХ ПОТОКА ЗАПРОСОВ

А. В. Сигаль

студент 532 группы; alex.sigal@mail.ru

Аннотация: В работе рассматриваются различные методы моделирования информационной системы, моделирование ее входных потоков заявок. Исследуется поведение системы в зависимости от различных распределений входных потоков, используются алгоритмы аппроксимации полученных данных.

Введение

В современных условиях трудно представить работу крупной компании без использования информационной системы. Ее преимущества касаются таких аспектов деятельности фирмы, как отслеживание состояния текущих проектов, контроль над документооборотом, хранение и обработка информации.

В 2011 году в компании Konecranes Finland Corporation — одном из мировых лидеров в производстве крупного грузоподъёмного оборудования — было принято решение о переходе на новую информационную систему на базе SAP. Стало понятно, что этот процесс будет сильно растянут во времени — Konecranes Finland Corporation имеет представительства более чем в 50 странах мира (в том числе и в России, которая, к сожалению, имеет низкий приоритет очередности внедрения системы). Тогда-то и родилась идея дипломной работы — написать с нуля информационную систему и проанализировать ее поведение при разных распределениях входных потоков заявок. Это стало возможно еще и потому, что старая информационная система уже почти не поддерживается, а новая будет доступна нескоро; в такой ситуации руководство сочло логичным создать собственную мини-систему для одного отдела, которую впоследствии можно будет использовать как готовую внедренческую модель.

Объектом исследования является распределенная клиент-серверная информационная система, имеющая трехуровневую архитектуру: клиентские классы, серверная часть, сервер базы данных. Система реализована на языке Java на основе технологии Google Web Toolkit (GWT), позволяющей ком-

пилировать код на Java в JavaScript. Для работы с базой данных MySQL 6.0 используется Spring Framework. Проект скомпилирован и развернут на вебсервере Tomcat 7.0. Моделирование происходит на отдельном, вручную написанном Java-сервере с использованием сокетов; для взаимодействия с базой данных используется JDBC.

Основная задача работы — создать полноценную информационную систему и исследовать поведение базы данных при различных вероятностных распределениях потоков заявок от пользователей.

1. Этапы решения задачи

Этапы решения поставленной задачи подразделяются на две части: практическую и теоретическую.

1. Практическая часть:
   1.1. Построение модели информационной системы.
   1.2. Разработка и реализация бизнес-логики.
   1.3. Тестирование и внедрение системы.
2. Теоретическая часть:
   2.1. Реализация алгоритмов распределений случайных величин
   2.2. Моделирование поведения системы на протяжении года в соответствии с различными распределениями потока заявок
   2.3. Анализ полученных результатов

В рамках данной работы мы не будем делать подробный обзор используемых технологий, не будем углубляться в практические аспекты. Поэтому перейдем сразу к полученным результатам.

2. Основные результаты

Была создана информационная система, составляющими которой являются следующие компоненты, отвечающие за:

1. Хранение информации (поставщики, покупатели, договора и т. д.).
2. Отслеживание статуса текущих проектов.
3. Контроль над документооборотом.
4. Отправка писем-напоминаний менеджерам проектов.
5. Сбор статистической информации.

Моделирование распределений происходило с помощью метода обратной функции, ЦПТ. Пуассоновское распределение было смоделировано с помощью предельной теоремы, согласно которой распределение биномиальной случайной величины $B_{n,p}$ при больших n и малых p хорошо аппроксимирует распределение Пуассона с параметром $a = np$.

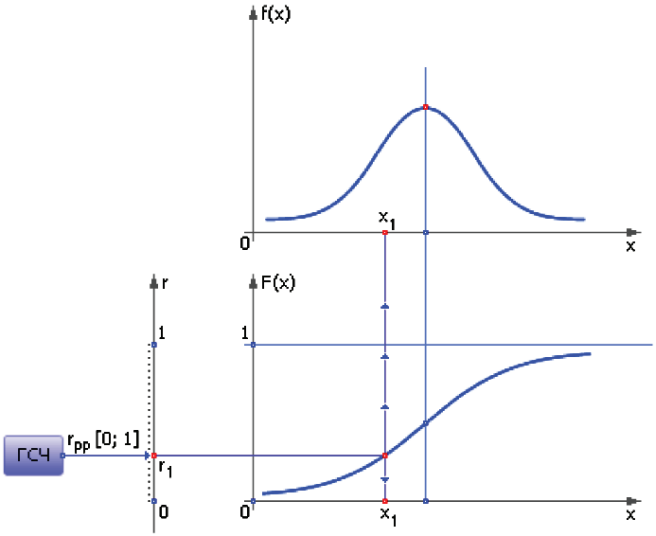


Рис. 1. Иллюстрация метода обратной функции

Итоговый алгоритм выглядит следующим образом:

1) Запускаем сервер, который создает поток, реагирующий на изменение размера базы и записывающий этот размер в список. Потоку задается интервал дискретизации.
2) Выбираем характеристику информационной системы (в нашем случае это интервал между заявками или количество заявок в день).
3) Запускаем несколько клиентов с одним типом распределения. Последний клиент в конце работы посылает серверу «exit».
4) При получении строки «exit» сервер выгружает данные из списка размеров в Excel, где по ним строится график роста базы во времени.

3. Анализ результатов

В ходе исследования были проанализированы три алгоритма аппроксимации: метод наименьших квадратов (МНК), метод последовательных приближений и метод *k*-средних. Самым простым является метод наименьших квадратов, единственным серьезным недостатком которого является высокая зависимость от начальных данных и, как следствие, неустойчивость к наличию ошибочных точек. В нашем исследовании не было замечено ошибочных точек, поэтому было принято решение воспользоваться МНК, благо он используется по умолчанию в Excel.

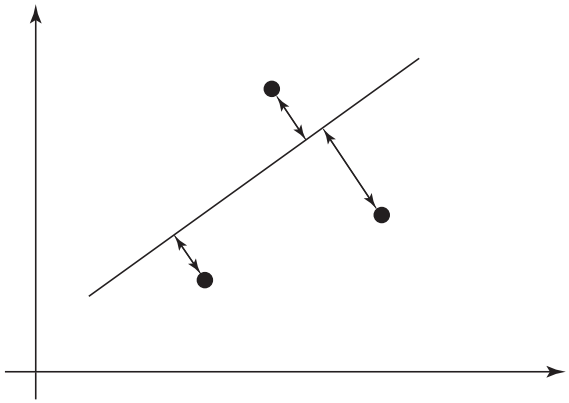


Рис. 2. Иллюстрация метода в случае линейной аппроксимации — минимизация функции $\sum (ax_i + by_i + c)^2$ при условии $a^2 + b^2 = 1$

При запуске алгоритма на одном потоке при всех распределениях рост базы происходил линейно. Однако при увеличении числа потоков график становится более выпуклым. В качестве иллюстрации возьмем экспоненциальное распределение.

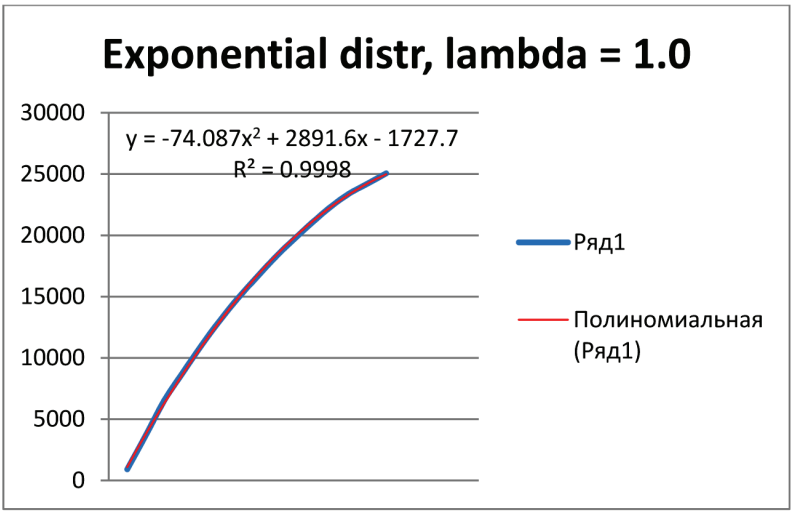


Рис. 3. Рост базы при экспоненциальном распределении

Вид полученного уравнения нельзя использовать для прогнозирования поведения функции на длительном временном отрезке, так как функция, очевидно, возрастающая и не может быть описана уравнением параболы с от-

рицательным старшим коэффициентом; однако на полученном промежутке полиномиальная аппроксимация оказывается наиболее точной.

Заключение

Полученные результаты показали, что при выбранном методе измерения размера базы сервер справляется с нагрузкой и не происходит никаких сбоев. Также заметно, что при текущей нагрузке размер базы через год достигнет всего 25 мегабайт, что говорит о том, что система будет нормально функционировать в течение длительного временного промежутка. В заключение приведем сводную таблицу, демонстрирующую различные функции роста базы в зависимости от распределений входных потоков с разными параметрами.

Тип распределения	Число потоков	Функция роста
Равномерное (3; 5)	1	$y = 170{,}1x - 55{,}842$
Равномерное (3; 10)	1	$y = 153{,}45x + 67{,}927$
Экспоненциальное (1)	1	$y = 807{,}07x - 592{,}44$
Экспоненциальное (1)	4	$y = -74{,}087x^2 + 2891{,}6x - 172{,}7$
Экспоненциальное (0.3)	1	$y = 111{,}3x - 187{,}26$
Нормальное (3; 0.1)	1	$y = 87{,}178x - 130{,}67$
Нормальное (3; 0.1)	4	$y = -0{,}228x^2 + 125{,}3x - 575{,}09$

Литература

1. Java JDBC Tutorial. URL: http://www.jdbc-tutorial.com/
2. *Мухин О.* Моделирование систем. Лекция 24. URL: http://www.stratum.ac.ru/textbooks/modelir/lection24.html
3. *Большаков С.* ALGLIB Project. Аппроксимация линейным или нелинейным МНК. URL: http://alglib.sources.ru/interpolation/leastsquares.php

ОБРАТНЫЙ МЕТОД ДЛЯ РЕШЕНИЯ ЗАДАЧ ЛОГИКО-ПРЕДМЕТНОГО РАСПОЗНАВАНИЯ ОБРАЗОВ И ОЦЕНКИ ЧИСЛА ШАГОВ ЕГО РАБОТЫ

Н. Петухова

Введение

Решение многих задач искусственного интеллекта, допускающих формализацию средствами языка исчисления предикатов, сводится в доказательству формул вида:

$$S(\omega) \Rightarrow \exists \overline{x}_{\neq} A(\overline{x}),$$

где $S(\omega)$ — набор постоянных атомарных формул или их отрицаний, $A(\overline{x})$ — элементарная конъюнкция [1].

При рассмотрении задач поиска логического вывода в исчислении предикатов одним из наиболее продуманных, с точки зрения сокращения перебора, методов является обратный метод, предложенный С. Ю. Масловым.

Доказательство $S(\omega) \Rightarrow \exists \overline{x}_{\neq} A(\overline{x})$ равносильно доказательству формулы:

$$\left(\&S(\omega)\right) \rightarrow \exists \overline{x}_{\neq} A(\overline{x})$$

при любых наборах значений ω, которую, используя равносильные преобразования аппарата математической логики, можно свести к формуле вида:

$$\forall a_1, ..., a_k \exists x_1, ..., x_{n_{\neq}} \left(\mathop{\&}_{i=1}^{\alpha} D_i(a_1, ..., a_k, x_1, ..., x_n) \right), \qquad (1)$$

где D_i имеет вид: $\vee \neg S(\omega) \vee P_{k_i}(\overline{x})$.

В работе предлагается модификация обратного метода для доказательства формул именно такого вида.

Описание метода

Сформулируем обратный метод Маслова для формул вида (1). Переменные $a_1, ..., a_k$ под квантором всеобщности будем рассматривать как константы.

Определение 1: Любой список формул Г вида $D_i(a_1, ..., a_k, c_1, ..., c_n)$ является **допустимым** для формул вида (1).

Определение 2: Любой список Г формул вида $D_i(a_1, ..., a_k, c_1, ..., c_n)$ является ***F*-набором** для формул вида (1), если формулы в этом списке не повторяются.

Определение 3: Пусть $\alpha_1, ..., \alpha_l$ — список констант из списка $a_1, ..., a_k$. $\beta_1, ..., \beta_l$ — список переменных и констант из списка $a_1, ..., a_k$. Рассмотрим систему равенств:

$$\begin{cases} \alpha_1 = \beta_1 \\ \quad ... \quad . \\ \alpha_l = \beta_l \end{cases} \qquad (2)$$

Пусть $u_1, ..., u_p$ список без повторений всех переменных. Входящих в равенства системы (2). Систему (2) будем называть **системой уравнений в переменных** $u_1, ..., u_p$. **Решением системы уравнений** (2) будем называть всякий набор значений констант $(\gamma_1, ..., \gamma_p)$ из списка $a_1, ..., a_k$ такой, что в результате одновременной замены переменных $u_1, ..., u_p$ на их значения в решении $(\gamma_1, ..., \gamma_p)$ левые и правые части каждого равенства системы совпадут. **Система уравнений (2) не имеет решений**, если в списке $(\gamma_1, ..., \gamma_p)$ есть повторения, так как согласно постановке задачи ищутся различные значения переменных.

Процедура 1 (отождествления переменных с константами): Пусть Г — список формул вида $D_i(a_1, ..., a_k, u_1, ..., u_n)$; S — система уравнений в переменных $u_1, ..., u_p$; σ — решение этой системы. Процедура отождествления переменных с константами в списке Г согласно системе S состоит в замене во всех формулах списка Г переменных $u_1, ..., u_p$ на их значения в решении σ.

Процедура 2 (преобразования списков формул в *F*-наборы): Пусть Г — список формул вида $D_i(a_1, ..., a_k, u_1, ..., u_n)$. Процедура преобразования списка Г в F-набор состоит в удалении повторений формул в этом списке.

Процедура 3 (построения замкнутого *F*-набора для формул вида (1)): Пусть $D_i(a_1, ..., a_k, t_1, ..., t_n)$ и $D_r(a_1, ..., a_k, v_1, ..., v_n)$ — формулы вида (1), где $t_1, ..., t_p$ — переменные, а $a_1, ..., a_k$ — константы. Обозначим первую формулу посредством A, а вторую — B. Предположим, что в A входит в качестве дизъюнкции атомарная формула $P(t_1, ..., t_s)$, а в B — отрицание формулы $P(a_1, ..., a_s)$, где P — s-местный предикат. Процедура построения замкнутого F-набора по парам формул A, $P(t_1, ..., t_s)$ и B, $\neg P(a_1, ..., a_s)$ состоит в последовательном выполнении следующих действий:

Применить к списку формул $[A, B]$ процедуру отождествления переменных с константами согласно системе уравнений в переменных

$$\begin{cases} t_1 = a_1 \\ \quad ... \quad . \\ t_s = a_s \end{cases}$$

В случае успешного завершения процедуры отождествления, применить процедуру преобразования полученного списка формул в F-набор.

Определение 4: *F*-набор будем называть замкнутым, если он может быть получен в результате применения процедуры построения замкнутого *F*-набора к подходящим парам формул A, $P(t_1, ..., t_s)$ и B, $\neg P(a_1, ..., a_s)$.

Процедура 4 применения правила Б к *F*-наборам: рассмотрим δ *F*-наборов

$$\begin{cases} \tilde{A}_1, & D_1\left(a_1, ..., a_k, \tilde{n}_1^1, ..., c_n^1\right), \\ & \quad ... \\ \tilde{A}_\delta, & D_\delta\left(a_1, ..., a_k, \tilde{n}_1^\delta, ..., c_n^\delta\right), \end{cases} \qquad (3)$$

где $\Gamma_1, ..., \Gamma_\delta$ — списки формул вида $D_i(a_1, ..., a_k, u_1, ..., u_n)$. Если какие-нибудь два из этих наборов содержат общую переменную, переименуем её на новую переменную. Таким способом добьемся, чтобы *F*-наборы (3) не содержали общих переменных. Применим к списку $\Gamma_1, ..., \Gamma_\delta$ процедуру отождествления переменных согласно системе уравнений в переменных:

$$\begin{cases} c_1^1 = c_1^2 = ... = c_1^\delta, \\ \quad ... \\ c_n^1 = c_n^2 = ... = c_n^\delta, \end{cases} \qquad (4)$$

а затем к полученному списку — процедуру преобразования списков формул в *F*-наборы. Построенный таким образом *F*-набор $\sum$ является результатом применения правила Б к *F*-наборам (3).

Так как перед применением этого правила мы переименовали все переменные, то все переменные в системе уравнений (4) различны и противоречащего условию о различности искомых переменных присвоения значений переменным в ходе решения этой системы не возникает. Поскольку при решении системы уравнений (4) переменные снова переименовываются на исходные имена, а δ *F*-наборов объединяются в один, применение правила Б к рассматриваемым формулам можно свести к простому объединению δ одночленных *F*-наборов в один δ-членный. Для простоты именно таким образом мы и будем применять эту процедуру: простым переписыванием вместо δ одночленных *F*-наборов одного δ-членного.

Теорема: *Для того, чтобы формула F была доказуема в исчислении предикатов, необходимо и достаточно вывести пустой F-набор □ в исчислении благоприятных наборов.*

Это исчисление задается приведенными ниже ***правилами А, Б и правилом перестановки***.

Правило А: *Замкнутые F-наборы являются благоприятными.*

Правило Б: *F-наборы, на которых процедура применения правила Б заканчивается успешно являются благоприятными.*

Правило перестановки: *Перестановка формул в благоприятном F-наборе также является благоприятным F-набором.*

Теперь, на основе приведенных процедур и правил можно сформулировать алгоритм поиска вывода для формул вида (1), использующий тактику обратного метода, и сравнить его с алгоритмом поиска вывода, использующим тактику метода резолюции.

Алгоритм и оценка числа шагов его работы

Определение 5: *F*-набор будем называть **пустым**, если все формулы, входящие в него не имеют переменных и тавтологичны.

Определение 6: *F*-набор будем называть **тупиковым**, если в него входит хотя бы одна формула, не имеющая переменных и являющаяся ложной или не являющаяся ни тавтологией ни противоречием.

1. По исходной формуле строим α *F*-наборов.
2. По имеющимся α *F*-наборам, согласно правилу Б, строим *F*-набор, формулы в котором не повторяются, (проще говоря, он получается простым переписыванием без конъюнкций всех дизъюнктов D_i).
3. Применяем процедуру построения замкнутого *F*-набора следующим образом:
   - 3.1. Ищем среди формул D_i рассматриваемого *F*-набора формулы D_j и D_r, содержащие *s*-местный предикатный символ $P(t_1, ..., t_s)$ и его отрицание (при другом наборе переменных или констант) $\neg P(v_1, ..., v_s)$, причем один из списков $t_1, ..., t_s$ или $v_1, ..., v_s$ должен быть списком только констант. Если и второй список имеет константы, то в двух списках на одних и тех же позициях должны находиться одни и те же константы: например, если первый список имеет вид: $t_1, ..., t_s = \{a_2, \boldsymbol{a_4}, \boldsymbol{a_3}, a_1\}$, то второй список должен иметь вид*:* $v_1, ..., v_s = \{x_2, \boldsymbol{a_4}, \boldsymbol{a_3}, x_1\}$.
   - 3.2. Решаем систему уравнений, отождествляющую списки переменных и констант $t_1, ..., t_s$, и $v_1, ..., v_s$. В случае, если эта система имеет решение, перейти к пункту 3.3.
   - 3.3. В полученном *F*-наборе удаляем повторения формул, если они есть.
   - 3.4. Проверяем, получился ли пустой набор, если получился: алгоритм заканчивает работу, если нет: если возможно дальнейшее применение пункта 3., то есть если в *F*-наборе существуют формулы, к которым возможно применение правила построения замкнутого *F*-набора — перейти к пункту 3, если получился тупиковый набор — перейти к пункту 4.

4. Отменяем одно действие и выполняем пункт 3. следующим образом: ищем другой вариант, подходящий для замыкания рассматриваемой формулы или новую подходящую для замыкания формулу. Если применение пункта 3. не дает нового присвоения значений переменным, применяем пункт 4. еще раз. Применяем правило 4. до тех пор, пока не выведется пустой набор или не закончатся возможности для отмены действий.
5. Если комбинации для замыкания F-наборов закончились, а результат не получен, значит, формула не выводима.

Пусть $\forall a_1, ..., a_k \exists (x_1, ..., x_n)_{\neq} \left(\&_{i=1}^{\delta} D_i (a_1, ..., a_k, x_1, ..., x_n) \right)$ – рассматриваемая формула, где: k — количество констант; n — количество переменных; ($k \geq n$, так как переменные не должны совпадать).

Одним ***шагом*** работы алгоритма считаем присвоение переменной значения (решение одного уравнения вида $x = a$), проверку на графическое совпадение атомарных формул или подстановку значений переменных в формулы, имеющие вхождения данных переменных.

l — наибольшее количество аргументов в атомарной формуле;

$\tilde{s}$ — максимальное количество вхождений одного и того же предиката (только без отрицаний, или только с отрицаниями) в описание объекта.

Нижняя оценка работы алгоритма. Количество шагов решения любой из задач распознавания образов при использовании алгоритма, основанного на тактиках обратного метода не менее $O(\tilde{s}l)$.

Верхняя оценка работы алгоритма. *Количество шагов, затрачиваемых на решение любой из задач распознавания образов с помощью исчисления благоприятных наборов, а так же нахождения значений для переменных, существование которых утверждается в антецеденте задачи не превосходит* $O(\delta\tilde{s}^{\delta}l)$.

Литература

1. *Kosovskaya T.* Discrete Artificial Intelligence Problems and Number of Steps of their Solution // International Journal on Information Theories and Applications, Vol. 18, No. 1, 2011. P. 93–99.

Технологии
и инструментальные средства
разработки программ

Сафонов
Владимир Олегович

д.т.н.
профессор кафедры информатики СПбГУ
член-корреспондент РАЕН

ИЗБАВЛЕНИЕ ОТ ШАБЛОННОГО КОДА В JAVA С ПОМОЩЬЮ ПРОЕКТА LOMBOK

Кондратьев А. Е.

anatoly.kondratyev@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Lombok — библиотека, предназначенная для сокращения шаблонного кода Java с помощью аннотаций. Это происходят за счет трансформации классов во время компиляции. Вместе с библиотекой предоставляются несколько реализованных трансформаций, но у пользователя может возникнуть желание реализовать собственное преобразование кода.

В этой статье рассматривается технология Lombok, сложности, возникающие при создании собственных трансформаций и описываются механизмы, упрощающие этот процесс.

Введение

«Шаблонный код» — это код, который повторяется во многих частях приложения с минимальными изменениями. Так, автоматически генерируемые get (), set () методы для полей в объектно-ориентированном программирование являются ярким примером шаблонного кода. Целью проекта Lombok было избавление от наиболее часто дублируемого кода в программах на языке Java, путем его замены на небольшой комплект аннотаций.

Использование аннотаций в качестве маркера, для осуществления привязок и даже для генерирования кода для использования во фрэймворках не является редкостью, но аннотации, как правило, не применяются для генерации кода, используемого непосредственно самим приложением. Проект Lombok, напротив, обрабатывает аннотации и модифицирует код непосредственно во время разработки, а благодаря интеграции в IDE (Integrated Development Environment — Интегрированная среда разработки) генерируемый код становится доступен через такие инструменты как автодополнение кода, иерархия методов класса и т.д. Для того, чтобы лучше понимать, как работает Lombok, ниже приведен пример использования:

Код с использованием Lombok	Обычный Java-код

`public class Example{`
`  @Getter @Setter`
`  private int age = 10;`
`}`
	`  public class Example {`
	`  private int age = 10;`
	`  public int getAge() {`
	`  return age;`
	`  }`
	`  public void setAge(int age){`
	`  this.age = age;`
	`  }`
	`  }`

Аннотации, позволяющие избавиться от шаблонных get-, set-методов, toString (), hashCode () и прочих, в рамках проекта называются трансформациями, поскольку изменяет код классов. В данный момент в составе самого проекта Lombok распространяется 12 трансформаций, включающие, помимо вышеперечисленных, трансформации конструкторов, трансформации работы с потоками, исключениями и логированием [1]. Несмотря на то, что многие популярные трансформации реализованы, возможна ситуация, когда разработчик захочет реализовать новую трансформацию. Так, в рамках изучения и улучшения этого проекта автором была добавлена трансформация, генерирующая стандартный паттерн проектирования «Строитель» [2], а так же проведены некоторые начальные исследования для реализации ряда других трансформаций.

Механизм работы проекта Lombok

Для дальнейшего обсуждения проекта необходимо познакомиться с механизмом его работы. Проект Lombok действует как процессор аннотаций [3], перехватывая попытки обработать «свои» аннотации, делегируя это действие собственным обработчикам. Каждый обработчик связывается только с одной обрабатываемой аннотацией. При обращении к подобному обработчику процессор аннотаций предоставляет объект, описывающий абстрактное синтаксическое дерево (Abstract Syntax Tree, в дальнейшем — AST) аннотированного узла. Обработчик аннотаций имеет возможность модифицировать AST, вставляя новые выражения, поля и методы, как, например, set-, get-методы из введения. То есть, Lombok, встретив свою аннотацию в AST, определенным образом модифицирует соответствующий узел. Из вышесказанного понятно, что для корректной работы Java-кода, написанного с использованием этого проекта, достаточно поместить Lombok-библиотеку в зону видимости компилятора.

Одна из основных проблем при использовании Lombok — это необходимость реализовывать разные обработчики для всех используемых компиляторов [4]. К примеру, если нужно, чтобы Lombok корректно работал в Eclipse, нужно реализовать обработчик для ECJ (Eclipse Compiler for Java), а для работы в NetBeansIDE — обработчик для javac. Поскольку AST в обоих случаях являются разными объектами, с разными методами взаимодействия, эта задача становится довольно сложной и нуждающейся в решении.

Суммируя действия, которые необходимо выполнить пользователю, решившему реализовать новую трансформацию в Lombok, получаем:

1. необходимо реализовать класс, описывающий аннотацию;
2. необходимо реализовать обработчики аннотации для тех компиляторов, с которыми Lombok будет использоваться:
   2.1. реализовав обработчик для javac получим поддержку в большинстве средств автоматической сборки (Apache Ant, Apache Maven), а так же в NetBeansIDE;

2.2. для использования проекта Lombok в Eclipse необходимо реализовать обработчик для ECJ.

Как можно видеть, создание инструмента, который бы обеспечивал взаимные трансформации между обработчиками для разных компиляторов, значительно упростило бы задачу.

Решением этой задачи должна была стать специальная библиотека lombok.ast, представляющая из себя отдельное представления для AST. Идеология работы этой библиотеки продемонстрирована на следующей диаграмме (см. рис. 1):

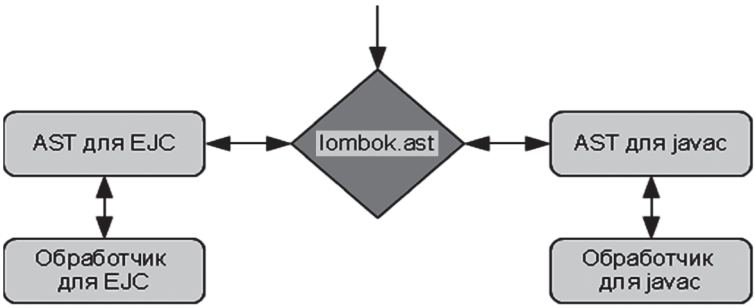


Рис. 1. Диаграмма работы библиотеки lombok.ast

Из этой диаграммы понятно, что трансформации между AST от разных компиляторов производятся через промежуточное представление, lombok.ast, которое должно обладать всей необходимой информацией. То есть, разработчику нужно реализовать трансформацию для lombok.ast, а для используемых компиляторов новые узлы будут сгенерированы с помощью библиотеки, что уменьшает затраты на реализацию новой функциональности в Lombok практически вдвое.

К сожалению, данная библиотека не развивается активно интернет-сообществом, тем не менее, в рамках выполнения магистерской работы автором этой статьи был внесен значительный вклад в её улучшение, что позволяет часть типовых трансформаций, таких как описание переменной, описание простого выражения, описание простого метода и прочее, реализовывать через неё.

Заключение

В документе был представлен проект Lombok, кратко продемонстрированы возможности его применения и описана главная идея его механизма работы. Так же проакцентированы некоторые сложности возникающие при создании новых трансформаций и приведен пример библиотеки, эти сложности уменьшающей.

Литература

1. *Ван дер Хель Д., Ейхорн Ф., Швитцерлут Р., Грутьянс Р. Ж., Спилкер Р., Конинг С.* Описание проекта Lombok. 2011. URL: http://projectlombok.org/features/index.html [дата просмотра: 07.04.2012]
2. *Фаулер М.* Шаблоны корпоративных приложений. М.: Вильямс, 2009. С. 544.
3. *Кимберлин М.* Reducing Boilerplate Code with Project Lombok. URL: http://jnb.ociweb.com/jnb/jnbJan2010.html [дата просмотра 18.04.2012].
4. *Найл Д.* Project Lombok: Creating Custom Transformations. 2011. URL: http://notatube.blogspot.com/2010/12/project-lombok-creating-custom.html [дата просмотра: 07.04.2012].

АСПЕКТНО-ОРИЕНТИРОВАННАЯ РАЗРАБОТКА ОБЛАЧНЫХ ПРИЛОЖЕНИЙ С ПОМОЩЬЮ СИСТЕМЫ ASPECT.NET

А. В. Григорьева

асп. кафедры информатики; nastya001@mail.ru

Д. А. Григорьев

к.ф.-м.н., лаб. Java-технологии СПбГУ; gridmer@mail.ru

В. О. Сафонов

д.т.н., проф., каф. Информатики, ММФ; v_o_safonov@mail.ru

Санкт-Петербургский государственный университет

Аннотация: Облачная платформа MS Azure предоставляет множество сервисов, которые позволяют разрабатывать производительные и надежные приложения. Управление этими сервисами производится из исходного кода облачного приложения, что приводит к проблеме «сквозной функциональности», когда реализация одной функции распределена по всему коду.

Данная работа предлагает подход, который позволяет устранить этот недостаток.

Введение

Облачная платформа MS Azure [1] предоставляет множество сервисов, которые позволяют разрабатывать производительные и надежные приложения. К таким сервисам относятся СУБД MS SQL Azure, сбор диагностической информации, управление своей конфигурацией, разделяемый кэш и пр. В зависимости от фактической или предполагаемой нагрузки, приложение может увеличивать или уменьшать количество своих одновременно запущенных экземпляров (instances). При этом все сложности взаимодействия с аппаратной инфраструктурой скрыты от программиста. Сервисные компоненты Azure не зависят от вызывающего их контекста и, тем самым, предназначены для гибкого повторного использования в различном окружении.

Однако при этом возникает проблема «сквозной функциональности» (cross-cutting concerns), когда для реализации какой-либо функции с помощью сервисов Azure, требуется вносить изменения во множество разных точек исходного кода.

Подобные ситуации возникают в ряде таких задач, как протоколирование, авторизация, проверка контрактов, профилирование и т.п. Сквозная функциональность затрудняет сопровождение проекта, ведь при смене способа решения задачи, напр. авторизации, необходимо внести изменения во множество точек исходного кода. Если же на определенном этапе потребуется

вообще удалить все, относящееся к определенной функции, напр. профилированию, то исходный код снова поменяется на множестве уровней.

Аспектно-ориентированное программирование (АОП) предлагает решать проблему «запутанной функциональности» вынесением каждой задачи в свой модуль — аспект. Затем на этапе компиляции или в процессе выполнения целевой программы, действия аспекта вставляются в ее заданные точки, обеспечивая нужное поведение.

В рамках данной работы был предложен подход к устранению сквозной функциональности в облачных приложениях с помощью аспектно-ориентированного программирования и инструмента Aspect.NET [2].

Технология Aspect.NET

Aspect.NET — это инструментарий АОП для платформы.NET, разработанный в лаборатории Java-технологии мат.-мех. факультета СПбГУ под научным руководством профессора В. О. Сафонова. С помощью Aspect.NET можно определять аспекты в отдельных библиотеках классов, а затем вплетать вызовы их методов в заданные места целевой сборки. Определения аспектов не зависят от конкретного языка, а разрабатывать их можно в любой среде разработки, поддерживающей платформу.NET.

Aspect.NET вплетает действия на уровне MSIL-инструкций после этапа компиляции целевой сборки, что влечет повышение производительности целевого приложения. Более того, такая «пост-обработка» дает возможность выбирать конкретные места применения действий аспектов.

Аспектом может быть любой класс, производный от класса Aspect (предопределенного в библиотеке Aspect.NET [3]). Реализация аспекта осуществляется статическими методами («действиями»), которые затем будут вставлены компоновщиком в заданные точки внедрения (joinpoints) в сборке целевого приложения. Требуемое множество точек внедрения задается в пользовательском атрибуте AspectAction () своего действия. Любое действие можно вставлять перед (ключевое слово %before), после (%after) или вместо (%instead) вызова заданного целевого метода. Название целевого метода задается с помощью регулярных выражений относительно его сигнатуры. Местонахождение точки внедрения внутри конкретного метода или класса можно фильтровать по ключевому слову %within.

Внутри действий можно использовать свойства базового класса, предоставляющие доступ к контексту точки внедрения, например, «this» и «target» объектам целевого метода, его метаданным (типа MethodInfo), отладочной информации и пр. Аргументы целевого метода передаются через аргументы действия.

Компоновщик аспектов — это отдельное консольное приложение. Его параметрами являются пути к сборкам аспектов и целевого приложения. При работе в MS Visual Studio требуется в свойствах проекта аспекта включить его вызов в события пост-компиляции (post-build events).

Каталог аспектов

В рамках данной работы предложены решения в виде аспектов Aspect.NET для следующих задач в рамках предметной области:

1. Протоколирование. Этот аспект перехватывает в целевом коде все вызовы метода System.TraceInformation () и перенаправляет их данные в хранилище диагностической информации Azure. Это позволяет просматривать и обрабатывать ее с помощью специальных инструментальных средств.
2. Прозрачная интеграция сторонних инструментов. Для платформы.NET разработано множество библиотек, каркасов и инструментов, таких как PostSharp [4], MS Code Contracts [5], MS EL [6] и пр., которые предоставляют полезные технические сервисы и службы. Однако их интеграция в целевое приложение приводит к изменениям в исходном коде, что может быть нежелательно при быстром прототипировании, когда необходимо удалить или заменить данную подсистему.
3. Обработка исключений. Если при обращении к некоторому веб-сервису возникает исключительная ситуация, аспект посылает ему еще несколько запросов по таймауту. В случае неудачи данные запрашиваются у запасного веб-сервиса. Вся служебная информация, в том числе стек, отсылается в службу протоколирования Azure.
4. Кэширование. Предложен аспект, действия которого подменяют целевой метод, загружающий данные из MS SQL Azure и сохраняют их в разделяемом кэше Microsoft.ApplicationServer.Caching.DataCacheFactory. В целевое приложение можно встроить управляющий кэшем веб-компонент, который также содержится в сборке этого аспекта.
5. Управление конфигурацией MS Azure. С помощью счетчиков производительности платформы аспект постоянно отслеживает текущую загрузку и по необходимости подключает или отключает новые вычислительные узлы, тем самым экономя денежные затраты клиента.

Для оценки полученных результатов использовалась интегральная метрика Maintainability Index, которая встроена в MS Visual Studio 2010 и определяет удобство сопровождения исходного кода отдельных классов и проекта в целом. Так, например, выделение из целевых классов приложения [7] функции кэширования в отдельный аспект позволило увеличить для них этот индекс на 33%. Для целевых классов другого приложения [8] аспектное управление конфигурацией дало прирост этого индекса на 24%.

Заключение

Таким образом, применение АОП и Aspect.NET в разработке облачного приложения позволяет повысить легкость его сопровождения, увеличить скорость разработки и снизить затраты за счет повторного использования универсальных аспектов. Все упомянутые аспекты доступны на сайте проекта Aspect.NET [8].

Литература

1. Сайт проекта MS Windows Azure // http://www.windowsazure.com
2. *Safonov V. O.* Using aspect-oriented programming for trustworthy software development. Wiley Interscience. John Wiley & Sons, 2008. 338 p.
3. *Григорьев Д. А.* Реализация и практическое применение аспектно-ориентированной среды программирования для Microsoft.NET // СПб.: Научно-технические ведомости, СПбГПУ. 2009. №3. С. 225–232.
4. Сайт проекта PostSharp // http://www.sharpcrafters.com/
5. Сайт проекта Code Contracts // http://msdn.microsoft.com/en-us/devlabs/dd491992
6. Сайт проекта MS Enterprise Library // http://wag.codeplex.com/
7. Windows Azure Training Course: Caching Data with Windows Azure Caching // http://msdn.microsoft.com/en-us/gg457898
8. J. Fultz «Performance-Based Scaling in Windows Azure» // http://msdn.microsoft.com/en-us/magazine/gg232759.aspx
9. Сайт проекта Aspect.NET // http://aspectdotnet.org/

РЕФАКТОРИНГ ДОКУМЕНТАЦИИ: ПОИСК КЛОНОВ И ОПЕРАЦИЯ DIFF

А. В. Шутак

студент кафедры информатики; artem.shutak@gmail.com

М. А. Смажевский

студент кафедры информатики СПБГУ; mihail.smazhevsky@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Данная работа проходит в рамках технологии DocLine, предназначенной для разработки документации семейств программных продуктов. Представлены новые операции рефакторинга документации, основанные на поиске клонов и операции diff. Эти операции предназначены для оптимизации работы (по скорости и качеству) технического писателя.

Введение

Техническая документация — важная составляющая современного промышленного ПО. Документация изменчива и структурно сложна. Кроме того, документация часто существует на разных уровнях детализации и в различных форматах: электронная (HTML), печатная (PDF), и другая.

Разработка семейств программных продуктов (product lines, далее СПП) — это популярный подход к промышленной разработке ПО, Создаваемые продукты обладают схожей функциональностью, направлены на один сегмент рынка и разрабатываются на основе общих активов с использованием заранее определенного метода [1]. Возможные повторно используемые активы: программные компоненты, архитектуры, процесс разработки, документация, и т.п. Документация СПП имеет широкие возможности для повторного использования.

Данная работа является частью исследовательского проекта DocLine [7]. DocLine — это метод разработки документации СПП на основе повторного использования. Данный подход включает в себя процесс разработки документации, язык разметки документации (DRL), имеющий текстовую и графическую нотации, а также пакет инструментальных средств.

Здесь мы представляем операции рефакторинга, носящие не только функциональный характер (как в [8]), но и рекомендательный. Они должны сильно упростить обработку больших объемов документации.

Обзор литературы

Технология DocLine и рефакторинг документации

Разработка семейств программных продуктов позволяет эффективно реализовывать повторное использование различных активов. Преимуществами

данного метода являются: повышение производительности труда, повышение качества продуктов, уменьшение времени на разработку конкретного продукта.

Метод разработки документации семейств программных продуктов DocLine [7, 8, 9] предложен на кафедре системного программирования математико-механического факультета СПбГУ. Он предназначен для разработки и сопровождения технической документации. Одна из отличительных характеристик такой документации — наличие большого количества возможностей повторного использования.

Метод DocLine состоит из следующих частей:

- процесс разработки документации;
- язык разработки документации DRL (Document Reuse Language) [9], включающий текстовую и графическую нотации. Фактически DRL расширяет DocBook [15] (который является стандартом де-факто в разработке документации для Linux/Unix-систем) механизмом адаптивного повторного использования.
- пакет инструментальных средств, включающий в том числе автоматизированные средства рефакторинга [8].

Обзор инструментов по поиску клонов

Задача поиска клонов не является новой и достаточно хорошо исследована. Этому свидетельствует множество работ в данной области. В [16] можно найти обширный перечень статей и инструментов в данной области (с кратким описанием).

Можно выделить несколько категорий, характеризующих инструмент:

- по способу использования. Автономные или standalone tool (CloneDR [10], Duplo [11]). И встраиваемые в среды разработки, такие как Eclipse и Visual Studio (SDD [12], CloneDifferentiator [13]);
- учитывают ли структуру документа. Здесь особое место занимают инструменты, специализированные на поиске клонов в языках программирования. С точки зрения автора, такой вид инструментов преобладает в данной области, но как правило, они либо вообще не поддерживают поиск в плоском тексте, либо результаты для него не удовлетворительны;
- по качеству найденных клонов. Четкие клоны — результаты в одной группе клонов полностью совпадают. И нечеткие клоны — результаты, находящиеся в одной группе могут совпадать не полностью, но близки по какой-то эвристике (SDD [12], CloneDR [10]);
- по виду лицензии на продукт. Свободно распространяемые и коммерческие.

Clone Miner

Исходя из необходимости дальнейшего встраивания выбранного инструмента в уже существующие средства DocLine, структуры DRL и необходи-

мости предсказуемости, возвращаемых им результатов, был сделан вывод, что используемый инструмент должен быть автономным, хорошо обрабатывать плоский текст и возвращать только четкие клоны.

Clone Miner tool [14] представляет из себя инструмент для поиска клонов высокого уровня в ПО (Higher-Level clones in Software). Под клонами высокого уровня (или структурными клонами) авторы понимают логические группы простых клонов. Новизной данного подхода является концепция структурных клонов и применение интеллектуального анализа данных (data mining). Инструмент имеет простой интерфейс использования: файл с входными данными (указываются пути к файлам), файл с результатами поиска.

Хотя данный инструмент изначально не ориентирован на работу с плоским текстом, но после обсуждения данного вопроса с авторами Clone Miner»а и проведения ряда испытаний (с удовлетворительными результатами), было принято решение использовать Clone Miner tool.

Обзор diff-tools

В настоящее время во многих программных продуктах существуют средства для построчного текстового сравнения двух и более файлов, или, проще говоря, утилита diff. Для решения задачи сравнения элементов языка DRL стандартных возможностей утилиты вполне достаточно, поэтому потенциально для встраивания подходят многие инструменты. Ниже приведён краткий обзор нескольких из них:

1. java-diff [2]. Реализация утилиты на java, возвращающая список отличающихся объектов; каждый из которых содержит сведения о добавленном, удалённом или изменённом тексте по отношению к сравниваемому. Инструмент возвращает низкоуровневые результаты, к тому же написан на java, поэтому удобен для интеграции. Однако его существенным минусом является отсутствие визуализации.
2. DiffJ [3]. Расширение java-diff для работы с java-кодом. Игнорируются пробелы и комментарии. Ряд дополнительных возможностей. Интересная разработка, не имеющая серьёзных преимуществ перед java-diff для работы с xml и drl.
3. Eclipse Compare [4]. Базовый diff-плагин эклипса. Удобный и простой плагин с ясной визуализацией результатов сравнения. Не смотря на открытость библиотеки достаточно сложно интегрируем в случае параметризации. Тем не менее именно этот инструмент был выбран для встраивания в DocLine, ввиду его готовой визуализации, привычной пользователям eclipse.
4. External diff Tool [5]. Вспомогательный инструмент, позволяющий использовать визуализацию базового diff-плагина для отображения результатов работы внешних diff-tools. С помощью него в дальнейшем может быть заменена логика поиска различий.

5. GNU Diffutils [6]. Достаточно мощный, но тяжеловесный инструмент, содержащий набор команд для сравнения до трёх файлов и слияния результатов.

Выводы

DocLine имеет довольно обширный пакет инструментальных средств для рефакторинга документации. Тем не менее, для документации, которую необходимо импортировать из DocBook, а так же для поддержки уже существующей документации нужны операции, облегчающие и оптимизирующие работу технического писателя. Такими операциями могут стать операции нахождения клонов в имеющейся документации и операция diff, позволяющая выделить большие куски общих активов.

Проектирование новых операций рефакторинга

Сценарии рефакторинга с использованием поиска клонов

Данный сценарий позволяет обнаружить в информационном элементе группы точных клонов и обработать их. Возможны два подхода для обработки: добавление клона в словарь (для небольших клонов) и выделение клона в информационный элемент. Выбор осуществляется пользователем.

Следующие действия пошагово описывают процесс рефакторинга (см. рис. 1):

1. Пользователь выделяет информационыый элемент, в нём осуществляется поиск групп клонов с помощью инструмента (clone detection tool).
2. Группы найденных клонов отображаются в пользовательском окне. Каждая группа описывается в новой строчке. В описание входит текст клона (или его начальная часть, если весь текст слишком велик) и количество клонов в информационном элементе.
3. Пользователь выделяет одну из групп клонов, после чего в информационном элементе подсвечиваются все вхождения данной группы клонов.
4. После просмотра подсвеченных результатов пользователь может обработать каждую группу клонов одним из двух способов:
   a. Выделить клон в словарь и подставить вместо каждого вхождения клона в информационный элемент ссылку на элемент словаря. (Задействуется существующее средство Insert into Dictionary).
   b. Выделить клон в новый инф. элемент и подставить ссылку на него вместо каждого вхождения клона в исходный информационный элемент. (Задействуется существующее средство Extract inf Element).

Рис. 1. Сценарии рефакторинга документации на основе поиска клонов

Сценарии рефакторинга с использованием diff

Данный сценарий позволяет построчно сравнить два информационных элемента и, по выбору пользователя, выделять в общие активы новые информационные элементы с одинаковым текстом и различиями, вынесенными в точки расширения.

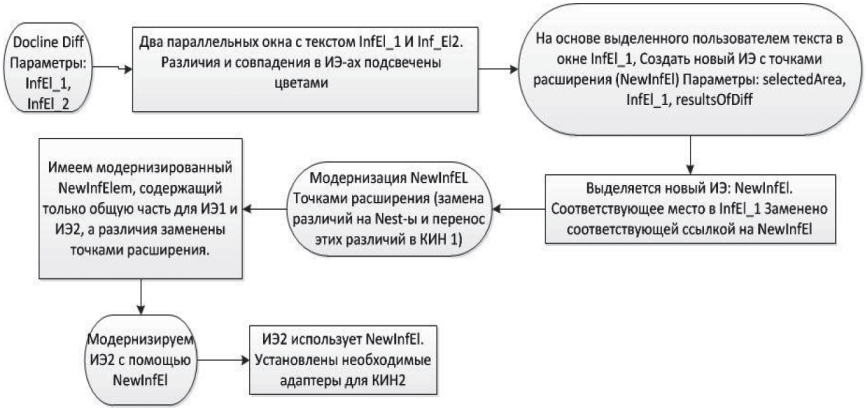


Рис. 2. Сценарии рефакторинга документации на основе diff

Следующие действия пошагово описывают процесс рефакторинга (см. рис. 2):

1. Применение diff-tool для двух информационных элементов (ИЭ). Результат: появляется два параллельных окна с текстом этих двух ИЭ. Различия и совпадения в ИЭ подсвечены цветами.
2. В окне ИЭ (одного из двух) на основе подсвеченных результатов diff-tool для двух ИЭ-ов пользователь выделяет ту часть, которую он хочет выделить в качестве нового информационного элемента с точками расширения (SelectedArea). Эта часть может захватывать в том числе и несовпадающие части для двух ИЭ. Может захватывать совпадающие части не полностью. С помощью средства рефакторинга Extract InfElement создаётся новый информационный элемент, содержащий выделенный текст. Результат: появляется новый ИЭ (NewInfEl) с текстом выделенной области (И все остальные изменения согласно Extract InfElement).
3. Модернизация NewInfEl точками расширения:
   a. В NewInfEl вместо каждого различия подставляем пустую точку расширения («Nest-метку» для будущей вставки).
   b. В конечный информационный продукт 1 (КИП1) для каждого различия создаем Adapter с Replace-Nest, в котором и содержится текст различия.
4. Модернизация ИЭ2 с использованием NewInfEl:
   a. Запоминаем различия в ИЭ2.
   b. Заменяем текст из ИЭ2, который соответствует выделенному тексту в ИЭ1, на ссылку на NewInfEl (применяем Replace With InfElemRef).
   c. В КИП2 для каждого различия, которое мы запомнили в шаге a), создаем Adapter с Replace-Nest, в котором и содержится текст различия.

Архитектура

Все операции встроены в структуру типовой операции рефакторинга DocLine [17]

Операции рефакторинга на основе поиска клонов

Были созданы модули отвечающие за следующее:

- работа с Clone Miner»ом из Java;
- разбор результатов работы Clone Miner»а и адаптация результатов;
- фильтрация и адаптация результатов поиска Clone Miner»а с учетом структуры DRL:
- графическое отображение результатов поиска в виде дерева, встроенное в Eclipse;
- выделение клона в общий актив (ИЭ или элемент словаря) и замена всей группы клонов ссылками на этот общий актив.

Первые 3 модуля запускают Clone Miner, обрабатывают его результаты и передают их на демонстрацию пользователю в графический модуль. Если

пользователь принимает решение, о необходимости применить к какой-либо группе клонов операцию рефакторинга, тогда срабатывает модуль выделения клонов в общие активы.

Операции рефакторинга на основе diff

Разрабатывая средство для переработки и переиспользования документации нельзя не думать о переиспользовании кода, поэтому новое средство рефакторинга необходимо было встроить, по возможности используя старые, в рамках существующей архитектуры. Для этого:

- был разработан модуль запуска и параметризации eclipse diff-tool
- был разработан модуль фильтрации и адаптации результатов diff с учетом структуры DRL
- на основе существующих операций рефакторинга (выделение в информационный элемент, выделение в точку расширения и другие), разработана операция рефакторинга;

По запросу пользователя срабатывает адаптированный eclipse diff-tool, его результаты адаптируются с учетом структуры DRL и отображаются пользователю. Далее по желанию пользователя для выбранного им фрагмента будет инициирована разработанная операция рефакторинга.

Заключение

В ходе данной работы были проанализированы инструменты по поиску клонов и diff-tools ([10], [11], [12], [13], [14], [2], [3], [4], [5], [6]), из них выбраны наиболее подходящие (Clone Miner [14] и Eclipse diff [4]). На их основе разработаны новые операции рефакторинга, позволяющие облегчить и оптимизировать работу технического писателя. Новые операции рефакторинга реализованы на языке Java в среде Eclipse и интегрированы в DocLine.

Литература

1. *Clements, P., Northrop, L.* Software Product Lines: Practices and Patterns. Boston, MA: Addison-Wesley, 2002. 608 p.
2. Утилита java-diff на сайте открытого программного обеспечения. http://www.incava.org/projects/java-diff/
3. Утилита DiffJ на сайте открытого программного обеспечения. http://www.incava.org/projects/java-diff/
4. Стандартный diff-плагин, встроенный в eclipse. http://www.eclipse.org/
5. Плагин, позволяющий запускать внешние, по отношению к eclipse, diff-tools. http://sourceforge.net/projects/externaldiff/
6. GNU Diffutils — пакет программ, связанных с поиском различий в файлах. http://www.gnu.org/software/diffutils/
7. *Романовский К. Ю.* Метод разработки документации семейств программных продуктов. // Системное программирование. Вып. 2. Сб. статей / Под ред. А. Н. Терехова, Д. Ю. Булычева. СПб.: Изд-во СПбГУ, 2007.

8. *Кознов Д. В., Романовский К. Ю.* Автоматизированный рефакторинг документации семейств программных продуктов // Системное программирование./ Вып. 4, под ред. А. Н. Терехова и Д. Ю. Булычева. СПб.: Изд. СПбГУ, 2009. С. 128–150.
9. *Романовский К. Ю., Кознов Д. В.* Язык DRL для проектирования и разработки документации семейств программных продуктов // Вестник СПб. ун-та. Сер. 10.2007. Вып. 4.
10. CloneDR tool на сайте Semantic Design. http://students.cis.uab.edu/tairasr/clones/literature/
11. Duplo tool на сайте Sourceforge. http://sourceforge.net/projects/duplo/
12. SDD tool на сайте Eclipse. http://wiki.eclipse.org/index.php/Duplicated_code_detection_tool_%28SDD%29
13. CloneDifferentiator tool на сайте Национального университета Сингапура. http://www.comp.nus.edu.sg/~yinxing/clonedifferentiator/
14. *Hamid Abdul Basit, Stan Jarzabek.* A Data Mining Approach for Detecting Higher-Level Clones in Software // IEEE Transactions on Software engineering. Vol. 35. No. 4, july/august 2009.
15. *Walsh N., Muellner L.* DocBook: The Definitive Guide. O»Reilly, 1999.
16. Code clone Literature on site of The University of Alabama at Birmingham. http://students.cis.uab.edu/tairasr/clones/literature/
17. *Минчин Л. А.* Разработка методов рефакторинга документации семейств программных чпродуктов, Дипломная работа, Санкт-Петербургский Государственный Университет, Математико-Механический факультет, кафедра системного программирования, 2008.

Рандомизированные алгоритмы в обработке данных и управлении

Граничин
Олег Николаевич

д.ф.-м.н.
профессор кафедры системного программирования СПбГУ
заведующий лабораторией
стохастических вычислительных систем НИИИТ СПбГУ

СОЗДАНИЕ КОМПЛЕКСА АВТОМАТИЧЕСКОЙ АЭРОФОТОСЪЁМКИ ДЛЯ ЛЁГКОГО БПЛА

И. Н. Калитеевский

i.kalit@yandex.ru

СПб НИУ ИТМО

Аннотация: Доступность, открытость, миниатюризация и мощность электронных встраиваемых систем открывают возможности для самого широкого применения.

В статье описана структурная схема комплекса автоматической аэрофотосъёмки, состоящего из беспилотного летательного аппарата, автопилота, мобильного телефона, сервера и клиентского компьютера.

Введение

Аэрофотосъёмка ведёт свою историю с момента появления первых самолётов. Уже в Первую мировую войну все противоборствующие стороны применяли её для фотографирования вражеских позиций. Картография, геологоразведка, охрана территорий и многое другое — сфера широкого применения аэрофотосъёмки, особенно в наше время, когда важна не только точность, но и постоянная поддержка актуальности информации о местности.

Однако, несовершенство элементной базы и отсутствие адекватных вычислительных мощностей делало процесс создания и эксплуатации БПЛА крайне дорогим и сложным, и долгое время это было прерогативой военной отрасли.

Технический прогресс открывает новые возможности. Миниатюризация, высокая надёжность и массовое производство открыли новую эру для беспилотной авиации, сделав процесс создания БПЛА более простым. Вместе с тем, он постоянно диктует новые требования:

- простота и низкая стоимость системы;
- быстрая подготовка к запуску;
- визуализация процесса в реальном времени;
- автоматическое распознавание необходимых объектов;
- автоматическая компенсация внешних воздействий: облачности, дождя и ветра;
- возможность вести слаженные групповые действия.

В настоящей статье рассмотрена проблема бесперебойной передачи фотоматериала с БПЛА на сервер и визуализация процесса на интерактивной карте местности.

Постановка задачи

Итак, имеется лёгкий БПЛА, оснащённый системой навигации, способной вести его по заранее заданному маршруту. Основные параметры БПЛА:

- размах крыла: 1,8 метра;
- масса: 2 кг;
- крейсерская скорость: 50–60 км/ч;
- дальность полёта: 100 км;
- время работы: 2 часа;
- допустимые условия: скорость ветра — до 5м/с.

Задача: произвести фотографирование определенных областей местности, отображая фотографии на карте на клиентском компьютере в местах съёмки в реальном времени. Качество фотографий должно быть пригодным для автоматического распознавания крупных объектов, таких как железнодорожный состав и большегрузная машина.

Существующие коммерческие БПЛА

Большинство отечественных невоенных разработок в области беспилотной авиации — это довольно большие летательные аппараты весом от нескольких десятков до нескольких сотен килограмм, оснащённые сложной дорогой аппаратурой и требующие для транспортировки грузовик, для запуска катапульту или даже взлётную-посадочную полосу (ВПП), для посадки — парашют или тоже ВПП.

В табл. 1 приведены тактико-технические характеристики проектов БПЛА, разработанных на территории России и наиболее похожих на наш по лётным и массо-габаритным характеристикам.

Таблица 1

ТТХ лёгких БПЛА

Наименование	ZALA 421–08	Т23 ЭЛЕРОН	ИРКУТ-2М	ИНСПЕКТР 201
Взлетная масса, кг	1,7 ... 2,1	2,8	3	1,3
Размах крыла, м	0,81	1,47	1,5	0,8
Длина, м	0,425	0,45	0,5	…
Скорость, км/ч	65–130	65–105	65–105	35–90
Высота полета, м	50–3600	до 3000	300–3000	50–1000
Радиус действия, км	до 15	…	20	5
Продолжительность полета, ч	1,5	1,25	до 1,5	0,5–0,6

Стоимость одного комплекта — от 2 млн рублей. В них применяется высококлассная аппаратура (например, фото и видеотехника, тепловизинные

приборы на подвижной платформе) и их спектр применения очень широк — от наблюдения за лесными пожарами до корректировки артеллерийского огня.

БПЛА, описанный в этой статье, пригоден для выполнения рассматриваемой задачи и его стоимость во много раз меньше.

Выбор структурных компонентов

Компоненты для размещения на БПЛА

Рассмотрим необходимый набор комплектующих для размещения на борту БПЛА:

1. для ведения непосредственно аэрофотосъёмки нужен фотоаппарат;
2. для последующей сортировки фотографий, а так же для отображения их на карте нужно знать их географические координаты, соответственно, нужен ГЛОНАСС/GPS датчик;
3. для передачи на сервер нужен Wi-Fi или GSM (в зависимости от конкретных условий постановки задачи) передатчик.

Весь необходимый функционал в полной мере присутствует во многих современных коммуникаторах. Цель фотографирования: визуальное и автоматическое наблюдение за перемещениями тяжёлой грузовой техники, поэтому высокое качество фотографий не нужно, возможности камеры, встроенной в мобильный телефон среднего качества, даже избыточны.

Стационарные компоненты

Необходимый набор стационарного оборудования:

4. для приёма и фотографий и связи с БПЛА нужен web-сервер;
5. для хранения фотографий нужен сервер;
6. для контроля и управления процессом нужен компьютер (ноутбук, планшет).

Для упрощения архитектуры первые две компоненты можно объединить, выбрав некоторый web-сервер.

В качестве web-сервера и клиентского компьютера можно использовать стационарный или переносной компьютер с той или иной операционный системой.

Общая структурная схема

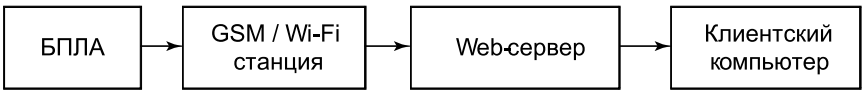


Рис. 1. Общая структурная схема

В первом приближении система работает следующим образом: БПЛА делает фотографии и по доступному протоколу отправляет на web-сервер. Клиентский компьютер получает фотографии с web-сервера.

Реализация

Выбор оборудования

В качестве операционной системы мобильного телефона была выбрана ОС Android 2.3.4. Разработка велась на языке Java.

Был выбран web-сервер со следующими параметрами:

- операционная система: Linux;
- web-сервер: Apache;
- язык для автоматического создания страниц: Php.

Выбор в пользу Linux был сделан по следующим причинам: во-первых, он бесплатен, во-вторых, настройка всех необходимых программ для запуска web-сервера осуществляется очень просто.

Способ отображения данных на клиентском компьютере: сайт с встроенной Google-картой. Google продоставляет удобные API-функции, позволяющие: встроить карту заданных размеров на сайт, поставить в нужных местах маркеры, выбрать нужный масштаб, начертить линии и многое другое. Этот инструментарий используется для отображения нужного учатска местности, для проведения линии траектории полёта и для выставления в реальном времени маркеров в местах, где были сделаны фотографии.

Разработка алгоритма

Алгоритм простой поочерёдной отправки координат мест съёмки и фотографий на web-сервер и поочерёдного отображения их на карте не подходит. Разница скоростей асинхронной передачи разных файлов через 3G-интернет и её общая ненадёжность при локальных ухудшениях качества покрытия ставят задачу разработки алгоритма, гарантирующего надёжность корректной работы комплекса.

Был разработан следующий алгоритм:

- При установке телефона на БПЛА телефон начинает делать запросы на web-сервер, и первый полученный ответ рассматривается как сигнал к старту работы комплекса.
- В основе работы программы лежит таймер, который постоянно контролирует положение самолёта в пространстве, и при входе в заранее отмеченную зону запускает процесс фотографирования через равные промежутки времени.

- Отправка фотографии на сервер представляет собой отдельный асинхронный процесс и происходит по следующему протоколу:
  - Посылается запрос на сервер, в котором передаются сведения о сделанной фотографии: порядковый номер, имя файла, время, GPS-координаты;
  - Ожидается ответ, содержащий порядковый номер, при отсутствии ответа через заданный временной интервал времени запрос повторяется;
  - После получения ответа стартует закачка файла;
  - Ожидается ответ, содержащий порядковый номер, при отсутствии ответа через заданный временной интервал времени закачка повторяется;
  - Когда закачка завершена, на сервере в базу данных вносится информация о закаченном файле: порядковый номер, имя файла, время, GPS-координаты;

Web-сайт работает так:

- через равные промежутки времени делаются запросы на сервер, на сервере отдельный скрипт находит в базе данных имена файлов фотографий, у которых нет отметок о том, что фотография уже отображается на сайте;
- скрипт возвращает на сайт имена файлов;
- делает отметку в базе данных, что имена файлов фотографий переданы на сайт.

На рис. 2 приведён пример отображения траектории движения БПЛА на клиентском компьютере с помощью карт Google.

Программа, в которой реализован описанный алгоритм, выполняется на мобильном телефоне, работающем под управлением операционной системы Android. В отличие от других мобильных операционных систем, Android предлагает в качестве среды разработки Eclipse, которая обладает открытым исходным кодом, бесплатна и совместима со всеми современными «настольными» операционными системами: Windows, Mac OS X, Linux. Так же отличием Android от Windows Phone и iOS является возможность дистрибуции программы без помощи официальных магазинов.

В Android есть весь необходимый функционал для управления устройствами, встроенными в смартфон: фотоаппаратом, ГЛОНАСС/GPS-датчиком, контроллером чтения и записи на SD-карту. На официальном сайте представлена необходимая документация.

Рис. 2. Пример результата отображения точек траектории на Google-карте в реальном времени

Заключение и планы на будущее

Таким образом, был разработан комплекс аэрофотосъёмки, способный кроме непосредственно фотографирования, визуализировать в реальном времени ход процесса.

В настоящее время в коплексе аэрофотосъёвмки отсутствуют следующие копмоненты:

- система регистрации пользователей, которая позволила бы использовать комплекс одновременно нескольким абонентам;
- система автоматического детектирования каких-либо изменений на местности по фотографиям, таких как вырубка леса и передвижение большегрузного транспорта.

Эти задачи включены в планы на будущее и в перспективе дополнят собой описанный в статье комплекс аэрофотосъёмки.

Литература

1. *Амелин К. С.* Лёгкий беспилотный летательный аппарат для автономной группы // Стохастическая оптимизация в информатике. Т. 6. 2010. С. 117–126.
2. *Амелин К. С., Граничин О. Н.* Мультиагентное сетевое управление группой лёгких БПЛА // Нейрокомпьютеры: разработка, применение. 2011. № 6. С. 64–72.
3. *Брюс Эккель*. Философия JAVA. Из-во Питер, 2009.
4. http://developer.android.com — официальная документация для разработчиков.
5. https://developers.google.com/maps/?hl=ru — API карт Google.
6. http://zala.aero — официальный сайт группы компаний «ZALA».
7. http://irkut.com — официальный сайт корпорации «Иркут».
8. http://www.enics.ru — официальный сайт компании «Эникс».
9. http://www.aerocon.ru — официальный сайт компании «Аэрокон».

СОЗДАНИЕ СИСТЕМЫ ОБРАБОТКИ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ АЛГОРИТМОВ ПОИСКА МИНИМАЛЬНОГО РАЗРЕЗА ГРАФА

Т. А. Кочанова

студент кафедры системного программирования;
kochanovatanya@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В работе рассматривается задача компьютерного зрения в контексте минимизации энергии для марковских случайных полей, ее актуальность и основные проблемы. Предлагается алгоритм, находящий достаточно хорошее приближение к точному решению, и требующий меньших вычислительных затрат, чем существующие методы.

Введение

Множество проблем компьютерного зрения могут быть сформулированы в терминах задачи расстановки меток, целью которой является найти приближения некоторой характеристики изображения.

Присваивание пикселям меток из некоторого заранее заданного набора фактически разбивает изображение на набор областей. Для примера возьмем пару фотографий одного и того же объекта, сделанных с разных ракурсов. Рассмотрим ситуацию, когда метки, присваиваемые пикселям изображения, будут соответствовать смещению (на сколько сдвинулся пиксель одного изображения относительно второго). На Рис. 1 изображена одна из исходных фотографий. Для наглядности каждой из меток, которые мы будем присваивать пикселям, сопоставим некоторый оттенок. Тогда полученная в результате функция расстановки меток естественным образом может быть представлена в виде изображения, раскрашенного с помощью цветов — меток. Результат такого разбиения изображен на рис. 2.

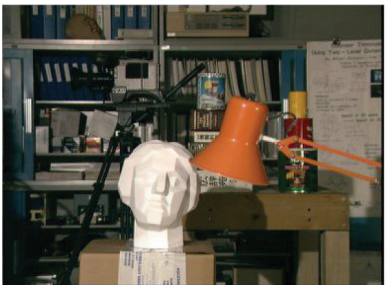


Рис. 1. Исходное изображение

Рис. 2. Полученное изображение

Проблемы

Сейчас основные сложности в решении подобных задач возникают из-за того, что они требуют слишком больших вычислительных затрат. В приложении к работе [1] показано, что задача минимизации энергии в одном из простейших случаев (для приближения, сохраняющего разрывы на границах объектов) является NP сложной. Следовательно, невозможно решить ее быстро, только если не P = NP.

NP-сложность поставленной задачи вынуждает нас искать не точное, а приближенное решение. Однако даже нахождение приближенного решения требует перебора значительного количества вариантов, что приводит к очень большим временным затратам и не позволяет обрабатывать изображения больших размеров. Поэтому важно разработать методы ускорения работы алгоритма, понять, за счет чего можно сократить количество вариантов для перебора, и постараться тем самым уменьшить вычислительные затраты, не теряя при этом точности, которую гарантируют классические подходы к решению поставленной задачи.

Существующие методы и способы их улучшения

Для решения задачи минимизации энергии широко используются методы, основанные на поиске минимального разреза графа [1–4]. С их помощью можно добиться очень хорошей точности при нахождении приближенного решения. Не стоит, однако, забывать про еще одно важное свойство, которое требуется для реального применения подобных алгоритмов — вычислительная эффективность.

Существующие методы работают, итеративно решая задачи по нахождению максимального потока для некоторого перестраиваемого от итерации к итерации графа. Соответственно, их эффективность непосредственно зависит от того, насколько быстро решается каждая из серии этих задач, что в свою очередь зависит от графа, для которого эта задача решается на каждой из итераций. Кроме того, безусловно, на скорость работы влияет также скорость сходимости приближений к итоговому решению, или, другими словами, количество необходимых итераций.

Следует отметить, что большинство существующих подходов не используют в своей работе знания о функционале энергии для уменьшения времени вычислений. Все методы вынуждены на каждой своей итерации проходить по всем существующим меткам, что крайне негативно сказывается на их вычислительной эффективности. Если учесть, что общее количество меток может быть довольно велико, в то время как реально значимыми из них чаще всего оказываются совсем немногие, кажется естественной идея присваивать меткам приоритеты, а не перебирать на каждом шаге их все.

Было проведено множество исследований с целью научиться выбирать наиболее подходящий набор возможных меток L и далее наилучшим образом с ним работать, что на каждом шаге алгоритма позволило бы находить оптимальное изменение текущего решения за полиномиальное время [5–7]. Однако оказывается, что если использовать данные для текущего приближения, то можно на каждой итерации расставлять меткам из L приоритеты и сортировать их в соответствии с ними.

Стоит отметить, что для большинства задач важными являются всего несколько меток, а остальные существенной роли не играют. Таким образом, возможные улучшения текущего приближения при рассмотрении наиболее приоритетных меток оказываются несравнимо существенней, чем улучшения, которых можно достичь, рассматривая низкоприоритетные метки. Это становится особенно актуальным, если говорить об использовании рассматриваемых нами алгоритмов в задачах сегментации изображений и стереозрения. Действительно, чаще всего достаточно четко выделить 2–3 основных объекта на изображении, не затрачивая на это слишком много времени, чем считать очень долго и получить в итоге разбиение на 20–30 областей, большая часть которых существенной роли не играет.

Итак, мы приходим к тому, что на каждой итерации, основываясь на обрабатываемых данных и текущем приближении, выгодным оказывается сначала найти метку, которая по нашим прогнозам даст наибольший выигрыш, а только после этого запускать процедуру поиска улучшений, но уже основываясь именно на данной метке.

Описание реализации

Сначала по входному изображению строится граф, который далее будет обрабатываться в алгоритме. Вершины графа — это пиксели изображения. Ребрами вершина соединена с пикселями соседями. То есть, не считая крайних случаев, из каждой вершины выходит по четыре ребра. Кроме того, инициализируются некоторые начальные приближения к решению, которые далее будут итеративно улучшаться. Фактически, начальное приближение строится очень просто — всем вершинам сначала присваивается одна и та же метка. Это практически не сказывается на времени работы алгоритма, поскольку за первую же итерацию, текущее приближение будет достаточно хорошо улучшено. После инициализации начинает работу процедура, представляющая собой цикл, который выполняется, пока не будет достигнуто достаточно хорошее приближение к точному решению.

Во время выполнения каждой из итераций происходит самая содержательная часть работы алгоритма, а именно, последовательно перебираются метки и для каждой из меток выполняется процедура, которая пытается улучшить текущее приближение к решению. Выбор очередной метки при переборе производится в зависимости от её текущего приоритета. Поиск улуч-

шения базируется на алгоритме нахождения минимального разреза графа (подробнее см. [1–3, 7]).

Кроме того, стоит обратить особое внимание на выбор метрик для предварительной обработки изображения. В реализованной системе использовалось несколько вариантов метрик. Оказалось, что даже незначительное изменение пороговых значений, или использование различных метрик для пересчета, приводит к существенным отличиям в результатах. Это приводит нас к идее создания модуля, который бы для каждого типа изображений пересчитывал входные данные при помощи различных метрик, и далее выбирал бы наиболее приемлемый результат.

Хочется также отметить, что хоть сейчас и существует множество алгоритмов, решающих рассматриваемую задачу, однако все они ещё очень далеки от совершенства и требуют значительных улучшений. А поскольку большое количество задач обработки изображений основываются на задаче расстановки меток, так важно научиться решать ее легко и быстро.

Заключение

В работе рассматривается задача поиска наилучшего способа расстановки меток для изображения, описывается алгоритм, созданный путем улучшения классических методов, а также приводятся наглядные примеры его работы.

Стоит отметить, что помимо использованных в работе, существует ещё множество различных подходов к решению задачи оптимизации и ускорения рассматриваемых алгоритмов. Это открывает нам широкие возможности, для улучшений существующей системы. В дальнейшем планируется применить рандомизированные методы на основе «сценарного подхода» [8], а также такие методы как вычисление псевдоградиента многомерной функции (потенциала) вдоль случайного направления [9].

Литература

1. *Boykov Y., Veksler O., Zabih R.* Fast Approximate Energy Minimization via Graph Cuts // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2001. Vol. 23. No. 11. P. 1222–1239.
2. *Komodakis N., Tziritas G., Paragios N.* Fast, Approximately Optimal Solutions for Single and Dynamic MRFs // IEEE Conference on Computer Vision and Pattern Recognition. Minneapolis, MN. 2007. P. 1–8.
3. *Komodakis N., Tziritas G.* A New Framework for Approximate Labeling via Graph Cuts // Proc. 10th IEEE International Conference on Computer Vision. Beijing. 2005. Vol. 2. P. 1018–1025.
4. *Batra D., Kohli P.* Making the Right Moves: Guiding Alpha-Expansion using Local Primal-Dual Gaps // IEEE Conf. On Computer Vision and Pattern Recognition. Providence, RI. 2011. P. 1865–1872.

5. *Komodakis N., Tziritas G., Paragios N.* Perfomance vs computational efficiency for optimizing single and dynamic mrfs: Setting the state of the art with primal-dual strategies // Computer Vision and Image Understading. 2008.
6. *Gould S., Amat F., Koller D.* Alphabet soup: A Framework for Approximate Energy Minimization // IEEE Conf. On Computer Vision and Pattern Recognition. 2009. P. 903–910.
7. *Veksler O.* Graph cut based optimization for mrfs with truncated convex priors// IEEE Conf. On Computer Vision and Pattern Recognition. 2007.
8. *Граничин О. Н., Шалымов Д. С., Аврос Р., Волкович З.* Рандомизи-рованный алгоритм нахождения количества кластеров // Автоматика и телемеханика. 2011. №4. С. 86–98.
9. *Граничин О. Н.* Рандомизированные алгоритмы стохастической ап-проксимации при произвольных помехах // АиТ. 2002. №2. С. 44–55.

МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛЕНИЯ ОПТИМАЛЬНЫХ ЗНАЧЕНИЙ ПАРАМЕТРОВ ВСТРОЕННОЙ СИСТЕМЫ

Е. В. Степанов

студент математико-механического факультета;
mikroz@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В работе рассматривается математическая модель линейной дискретной динамической системы, для которой предлагается метод построения управляющего воздействия на основе метода инвариантных эллипсоидов, а также метода построения доверительной зоны для неизвестных параметров системы.

Введение

История компьютерных встраиваемых систем берет начало еще в 50-х годах XX века. Одной из первых таких систем был Apollo Guidance Computer, разработанный в лаборатории MIT Чарльзом Драпером. Он был создан из логических элементов на основе транзисторов и имел жесткий диск для основной памяти. С падением стоимости микропроцессоров и микроконтроллеров появилась возможность замены дорогих аналоговых компонентов микропроцессорами. К середине 80-х годов большинство внешних системных компонентов было интегрировано в один чип вместе с процессором, что стало представлять современный «вид» микроконтроллеров и позволило естественным образом программно получать данные и управлять работой отдельных составляющих системы. Позднее стали использоваться так называемые однокристальные системы (System-on-a-Chip, SoC), что привело к возможности встраивать такие системы в мобильные устройства.

Встраиваемые системы получили хорошее распространение в мире и имеют обширную область применения, к которой относятся:

- средства автоматического регулирования и управления техническими процессами, например авионика, контроль доступа;
- банкоматы и платежные терминалы;
- телекоммуникационное оборудование.

Центральным процессорным устройством для встроенной системы могут служить многие из современных микропроцессоров и микроконтроллеров. Конкретный вид определяется из целей и задач, выполняемых встраиваемой системой. Совершенствуясь, встраиваемые системы становятся «интеллектуальными», т. е. способными адаптивно управлять каким бы то ни было процессом, принимая во внимание множество внутренних и внешних фак-

торов, например внешние помехи при получении каких-либо данных из среды. Это дает большее количество данных для управления целой системой, что существенно повышает качество ее работы. В дальнейшем вопрос остается лишь за методикой управления системой и учета внешних воздействий на нее. Для вычисления параметров управляющего воздействия таких систем используется математическая модель динамических систем, что делает возможным использование соответствующего математического аппарата.

Доверительные множества для параметров линейного объекта управления

Дадим некоторые общие определения, использованные в работе. Рассмотрим линейную дискретную динамическую систему

$$y_t = G_0(z^{-1})u_t + v_t, \tag{1}$$

где $t = 1, ..., N$ — дискретное время, u_t — управление (вход) и y_t — выход. Внешний ограниченный шум

$$v_t : \| v_t \| \le 1 \forall t \ge 0 \tag{2}$$

описывает совокупность внешних источников возмущения, независимых от u_t и вызывающих вариации по состоянию (y_t). Оператор сдвига z^{-1} определяется следующим образом: $z^{-1}u_t = u_{t-1}$. $G_0(z^{-1}) \in G(t, z^{-1})$ – передаточная функция из множества передаточных функций с параметром t: $G_0(z^{-1}) = G(t_0, z^{-1})$ для некоторого $t \in T$. Известна структура класса $G(t, z^{-1})$, но параметр t_0 неизвестен.

Постановка задачи построения доверительного множества состоит в том, чтобы, используя конечное число данных о входах и выходах, полученных в моменты времени $t = 1, 2, ..., N$, определить доверительную область $\overline{T}$, которой с заданной вероятностью принадлежит t_0. Также, область $\overline{T}$ должна быть построена без использования каких-либо априорных знаний об уровне, распределении или корреляции помех.

Метод построения доверительного множества предложен О. Н. Граничиным и подробно описан в работе [1].

Линейный динамический регулятор по состоянию

Определение 1. Назовем множество

$$E_y = \{y \in \mathbb{R}^n : y^T P^{-1} y \le 1\}, \; P > 1 \tag{3}$$

эллипсоидом с центром в начале координат. Будем называть такой эллипсоид инвариантным (по состоянию) для замкнутой линейной системы, соответ-

ствующей системе (1), если при попадании в какой-то момент времени траектории системы в этот эллипсоид, она более не выходит из него.

Несложно показать, что инвариантный эллипсоид является аттрактором для динамической системы. Мы будем рассматривать инвариантные эллипсоиды как характеристику влияния внешнего шума на траекторию системы. Критерием минимальности ограничивающего по состоянию эллипсоида будем считать критерий следа матрицы P (который и будем минимизировать).

Нетрудно заметить, что след матрицы P соответствует сумме квадратов полуосей эллипсоида. Таким образом, задача оценки степени влияния внешних ограниченных помех на состояние системы (1) сводится к нахождению минимального по критерию следа ограничивающего эллипсоида (3). В частности, если состояния — скаляры, то оцениваться будет максимальное по модулю значение состояния.

Важным моментом является построение нехрупкого регулятора K в виде статической линейной обратной связи по состоянию

$$u_t = Ky_t. \tag{4}$$

таким образом, чтобы возмущенный регулятор $K + d_K$, $\| d_K \| \le D_k$ стабилизировал замкнутую систему (1) и оптимально подавлял воздействие внешних возмущений.

В работе рассмотрен робастный вариант построения нехрупкого регулятора для подавления внешних возмущений, т. е. один из параметров динамической системы обладает некоторой ограниченной неопределенностью.

Решение поставленной задачи о нахождении параметров регулятора сводится к решению задачи полуопределенного программирования (Semidefinite Programming, SDP) с ограничениями в виде линейных матричных неравенств (Linear Matrix Inequalities, LMI).

Рассмотрим линейную дискретную систему:

$$y_{t+1} = (A + D_A)\, y_t + Bu_t + Dv_t, \tag{5}$$

с некоторым начальным состоянием $y_0 \in \mathbb{R}^n$, матрицы A, B, D — вещественные, размерности $n \times n$.

Теорема. Решение $\hat{R}, \hat{P}, \hat{Y}$ задачи минимизации $TrR \to \min$ при линейных матричных ограничениях

$$\begin{pmatrix} -\alpha P & (AP+BY)^T & 0 & P & P \\ AP+BY & -P+\gamma_A^2\varepsilon_1 E+\gamma_K^2\varepsilon_2 BB^T & D & 0 & 0 \\ 0 & D^T & -(1-\alpha)E & 0 & 0 \\ P & 0 & 0 & -\varepsilon_1 E & 0 \\ P & 0 & 0 & 0 & -\varepsilon_2 E \end{pmatrix} \le 0,$$

$$\begin{pmatrix} -R & -P & 0 \\ -P^T & -P & P \\ 0 & P & \varepsilon_3 E \end{pmatrix} \le 0,$$

где минимизация производится по матричным переменным P, R, Y, скалярным переменным ε_1, ε_2, ε_3 и скалярному параметру α, определяет матрицу R ограничивающего эллипсоида по состоянию системы (5) и статический регулятор по состоянию

$$K=YP^{-1}.$$

Заключение

В работе описаны метод построения доверительного множества для неизвестных параметров системы, а также метод для вычисления параметров статического регулятора по состоянию системы. Предлагается последовательное итеративное использование методов для уточнения неопределенности неизвестного параметра системы.

Литература

1. *Граничин О. Н.* Неасимптотическое доверительное множество для параметров линейного объекта управления при произвольном внешнем возмущении // Автоматика и Телемеханика. 2012.
2. *Хлебников М. В.* Подавление ограниченных внешних возмущений: линейный динамический регулятор по выходу // автореферат докт. диссер. Институт проблем управления им. В. А. Трапезникова РАН, Москва. 2010.

РАСПОЗНАВАНИЕ ЯЗЫКА ЖЕСТОВ НА ВИДЕО ПОТОКЕ

С. А. Землянская

студент кафедры системного программирования;
svetlana.zemlyanskaya@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Статья посвящена распознаванию языка жестов на видео потоке. Приводится описание методов машинного обучения, наиболее применимых в этой области, рассматривается конкретный вариант их использования при реализации практического приложения, производится сравнение их эффективности в частном случае.

Введение

В последнее время всё большее внимание уделяется автоматическому распознаванию жестов с помощью визуальных систем. Такой интерес связан, в первую очередь, удобством использования такого интерфейса и богатством новых возможностей, которые он привносит.

Постановка задачи в общем случае достаточно сложна, и на первом этапе в качестве распознаваемого набора жестов было решено использовать не весь набор жестов из языка глухонемых, а только лишь латинский алфавит. В работе описывается разработанное автором соответствующее программное приложение по распознаванию латинского алфавита. На входе приложения — видео поток, снятый с одной вебкамеры, а на выходе выдается полученный текст.

Классификаторы

Для классификации объекта выделяется набор признаков, который считается достаточным для идентификации класса объекта. Набор признаков представляется в виде n-мерного вектора. Классификатор может давать положительный результат, в случае принадлежности объекта к главному множеству, или отрицательный, в противном случае.

Для решения задачи классификации рассмотрим алгоритм обучения с учителем. На вход подаётся тренировочное множество из m векторов: $x^{(1)} = \mathrm{col}\,[x_1, x_2, ..., x_n]$, $x^{(2)}$, ..., $x^{(m)}$. Каждому вектору ставится в соответствие ожидаемый результат классификации: $y^{(1)}$, $y^{(2)}$, ..., $y^{(m)}$.

Логистическа регрессия

Метод логистической регрессии [6], основан на представлении классификатора как параметрически заданной функции:

$$h_{\Theta}(x) = \frac{1}{1 + e^{-\Theta^T \cdot x}}, \quad \Theta = [\Theta^1, \Theta^2, ..., \Theta^n].$$

С помощью тренировочного множества подбирается набор параметров наиболее оптимальный для классификации. Определяется функция ошибки и с помощью метода градиентного спуска находится её минимум:

$$J(\Theta) = \frac{1}{2 \cdot m} \cdot \sum_{i=1}^{n} \left(h_{\Theta}(x^{(i)} - y^{(i)}) \right)^2 \to \min, ,$$

где $y^i \in \{0, 1\}$ [6].

Support Vector Machines

Классификация по этому методу происходит с помощью разделения точек различных классов гиперплоскостью [7]. Таких гиперплоскостей может быть много, поэтому в качестве меры качества выбранной задается «зазор» между классами. Если существует гиперплосость, разделяющая классы с максимальным зазором, то она называется оптимальной разделяющей гиперплоскостью, а соответствующий ей линейный классификатор называется оптимально разделяющим классификатором.

Строим разделяющую гиперплоскость, которая имеет вид:

$$w \cdot x - b = 0,$$

где w — перпендикуляр к разделяющей гиперплоскости.

Отдельно введём понятие «ошибка классификации» для каждого элемента — e_i. Результат классификации, в таком случае, может принимать два значения: $y_i \in \{-1, 1\}$. Согласно такой классификации каждый вектор x должен удовлетворять такому условию:

$$y_i \cdot (w \cdot x_i - b) \geq 1 - e_i.$$

Для нахождения максимально возможного зазора между классами при таком линейном разделении необходимо минимизировать следующую функцию:

$$\frac{1}{2} \cdot \| w \|^2 + \sum_{i=1}^{n} e_i \to \min.$$

Нелиненая SVM

Идея классификации с помощью нелинейной SVM [7], практически полностью повторяет идеи линейной, с той лишь разницей, что каждое скалярное произведение заменяется нелинейной функцией ядра.

В работе использовалось ядро Гаусса:

$$k(x_1, x_2) = \exp\left(\frac{-\|x_1 - x_2\|^2}{2 \cdot \sigma^2}\right)..$$

Мультиклассификатор

Основой мультиклассификации был выбран алгоритм OneVsAll.

1. Необходимо произвести классификацию с m различными исходами.
2. Для каждого $i \in \{1 : m\}$ происходит классификация, где положительным исходом считается принадлежность объекта к *i*-ому классу и отрицательной в противном случае.
3. В результате получается *m*-мерный вектор вероятностей, принадлежности к каждому классу, на основе которого и происходит классификация.

Алгоритм

Входная информация — видео поток с одной вебкамеры. Изображения получаются по одному в секунду. Такая скорость была выбрана по причине того, что она позволяет существенно увеличить время на обработку изображения без потери важной информации.

1. *Предварительная обработка изображения*

На первоначальном этапе изображение уменьшается и с помощью алгоритма Канни [3], параметры для которого были выбраны эмпирическим путём. Этот алгоритм позволяет выделить границы.

2. *Построение предположений*

К бинарному изображению применяется метод Хаффа [4], нахождения дуг окружностей, исходя из результатов которого строится ряд предположений. Это место является наиболее узким в разработке, т.к. требует полного обхода изображения. На выходе получается список из областей, где согласно предположению, может находиться палец.

3. *Классификация пальцев*

Было сделано предположение, что в большинстве жестах, пальцы, либо находятся в непосредственной близости от каркаса ладони, т. е. многоуголь-

ника, описанного вокруг неё, либо близко друг от друга. На основе этого было выделено два признака классификации:

- расстояние до ближайшего потенциального пальца
- расстояние до каркаса ладони

4. *Классификация жестов*

На вход поступает список пальцев, уже прошедших первый этап классификации. Нормированные координаты пальцев и являются вектором признаков, по которому происходит классификация.

Результаты работы

Для оценки классификации использовались стандартные метрики (precision и recall):

Precision:

	Logistic regression	Gausian SVM
Logistic regression	0.35714	0.47619
Gausian SVM	0.43478	0.30435

Recall:

	Logistic regression	Gausian SVM
Logistic regression	0.27778	0.47619
Gausian SVM	0.47619	0.46667

Заключение

В результате проделанной работы, было написано приложение по распознаванию латинской азбуки жестов. На вход приложение получает видеопоток с одной вебкамеры, и, после соответствующей обработки, на выход возвращается полученный текст. В качестве дальнейшего продолжения работы наиболее перспективными представляются направления:

- *по оптимизации работы приложения*.

На первоначальном этапе составления предположений, скорость работы сильно замедляется в связи с необходимостью в полном обходе изображения. Это процесс в будущем стоит реализовать аппаратно.

- *по расширению базы распознаваемых жестов*

В работе была рассмотрена проблема распознавания азбуки жестов. В будущем, хотелось бы расширить круг и добавить возможность получения более сложных конструкций.

Литература

1. *Воскресенский А. Л., Ильин С. Н., Zelezny.* О распознавании жестов языка глухих. 2010.
2. *Абакумов В. Г, Ломакина Е. Ю.* Автоматическое распознавание жестов в интеллектуальных системах. 2010.
3. *Шапиро Л., Стокман Дж. С*. Компьютерное зрение. 2006.
4. *Форсайт Д., Понс Ж.* Компьютерное зрение. 2004.
5. *Tom M. Mitchell*. Machine Learning. 1997.
6. *Lemeshow, David W. Hosmer*. Applied logistic regression. 2000.
7. *Hsu Chih-Wei, Chang Chih-Chung,* and *Lin Chih-Jen.* A Prectical Guide to Support Vect Classification. 2003.
8. *A. Barr, Edvard A. Feigenbaum*. The Handbook of Artifical Intelligence. 1990.
9. *Gary Bradski, Adrian Kaehler.* Learning OpenCV. 2008.

НЕЛИНЕЙНАЯ ДИНАМИКА И ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Е. Н. Бендерская

к.т.н., доцент кафедры компьютерных сетей и программных технологий факультета технической кибернетики;
helen.bend@gmail.com

Санкт-Петербургский государственный политехнический университет

Аннотация: В докладе проведен анализ тенденций развития методов искусственного интеллекта и показано место биоинспирированных методов при решении задач обработки информации и управления. Раскрываются причины необходимости использования методов нелинейной динамики при разработке интеллектуальных систем.

Показано, что обращение к теории хаоса в области информатики и вычислительной техники является естественным продолжением общих тенденций их развития и является неизбежным для получения параллельных эффективных интеллектуальных вычислителей следующего поколения, работающих в условиях минимума априорной информации о решаемой задаче.

Введение

Сформировавшиеся первоначально и развивавшиеся независимо друг от друга, основные два подхода к разработке ИИ — символьный (ассоциируется с экспертными системами, базами знаний) и коннекционистский (ассоциируется с нейронными сетями (НС) и другими сетевыми моделями) в настоящее время уже не конкурируют, а дополняют друг друга, в том числе и через множество нейро-нечетких методов, в которых сочетается и обучение и задание явных знаний в виде некоторого набора правил [1, 4, 6].

Однако, несмотря на это взаимное обогащение в целом теория ИИ все еще достаточно далека от той амбициозной цели, которая была поставлена и казалась достаточно легко достижимой в скором времени — создание действительно интеллектуальных систем, способных развиваться, самостоятельно ставить задачи и решать их, находить новые, явно не заложенные в систему решения, то есть воспринимать информацию, думать, действовать как человек [2, 3, 4, 7].

Наука о мозге и ИИ

В связи с большими успехами в области генной инженерии, накоплением большого объема экспериментальных данных о внешних проявлениях рабо-

ты мозга, появившихся с развитием технологии неинвазивного наблюдения за функционированием мозга, а также с развитием вычислительной техники, позволяющей обрабатывать огромные объемы данных, удалось узнать достаточно много о различных сторонах работы мозга и с такой степенью детализации, что стало возможно описание на уровне математических моделей процессов записи и извлечения информации из отдельных клеток мозга с оценкой концентрации соответствующих химических соединений и уровня вызванных потенциалов.

Однако, на настоящий момент приходится говорить о разрозненности и фрагментарности имеющихся знаний. Задача интегрирования их в единую теорию ставится во многих научных работах, о ее необходимости также написано достаточно много.

Что же может выступить в качестве обобщающего разные урони знания о такой сложной системе как мозг? Один из возможных ответов — синергетика, наука о самоорганизации систем [8]. Причем привлечение синергетики для развития теории ИИ является достаточно естественным, если учесть, что мозг — динамическая система. Основными методами синергетики являются методы нелинейной динамики и теории хаоса.

Проблемы основных подходов ИИ

Основная проблема, которая встала перед разработчиками ИИ символьного направления — невозможность формального представления всех возможных ситуаций, с которыми может столкнуться система в процессе функционирования. Если ввести некоторые ограничения, то удается успешно решать задачи определенного класса, представленные именно введенными ограничениями, но универсальность оказывается недостижима. Система будет «умна» настолько, насколько при ее создании полно учтены все возможные варианты исходных данных. Во многих приложениях это и не требуется, но при создании ИИ как раз ставится задача получения новых знаний внутри самой системы. При этом часто речь идет не столько об умении логически доказать какую-либо теорему на основе уже показанной теоремы, сколько о возможности дополнить неполное, а иногда противоречивое описание исходных данных и вариантов принимаемых решений. В этом случае мы сталкиваемся с тем, что должны дать системе достаточно степеней свободы и в принципе, в сложной незнакомой ситуации мы заранее не сможем предсказать действия системы, можем лишь выдвинуть гипотезы и вычислить соответствующие вероятности.

С таким же ограничением по представлению всех возможных вариантов исходных данных и ситуаций столкнулись и разработчики коннекционистского направления. Хотя НС и оперируют с невидимыми образами, обучаясь, умеют обобщать и в принципе дополнять недостающие пробелы, тем не менее, в целом по большей части статические НС прямого распространения

не более чем универсальные аппроксиматоры [9]. Отдельное направление НС с самоорганизующимися картами Кохонена (СОК) также обладает множеством ограничений, несмотря на удобство визуального представления результатов обобщения и выявления неявных знаний в виде кластеров [9, 10].

Биоинспирированные методы

Несмотря на то, что сама идея создания ИИ является уже биоинспирированной, при ее реализации может быть разная степень приближения к биологическому прототипу.

В последнее время все больше внимания уделяется биоинстипированным нейронным сетям на биоподобных нейронах, хотя и классические НС тоже биоинспирированные, но на формальном нейроне. В силу того, что ставшие классическими структуры нейронных сетей на формальных нейронах исчерпали себя в том смысле, что качественного скачка в этих подходах уже не получить, их место среди прочих подходов к решению определенного класса задач уже определено и ясны пределы по достижимому качеству и условия получения решения тоже известны.

Скорее всего, попытки детального воспроизведения процессов, происходящих в отдельных нейронах, без достаточно четкого понимания того, какие принципы взаимодействия и развития заложены в биологических нейронах, и попытки построения искусственных НС на биологически подобных нейронах, не будут давать значительного продвижения в получении желаемого результата — ИИ. Воспроизведение во всех деталях химико-физических процессов, протекающих в нервных клетках, представляет интерес скорее для биологических приложений и менее — для ИИ. Конструирование общего снизу-вверх, причем с очень низкого уровня, и простое соединение частей не даст целого — системы с новыми свойствами. Для того, чтобы синергизм произошел нужно, чтобы изначально были заложены принципы самоорганизации.

Распределенные вычисления и неустойчивая динамика

Новые структуры на основе случайных НС, предложили независимо друг от друга Маас и Джагер [11]. Идея в целом одна и та же — наличие случайной НС и отдельной структуры, которая за этой НС наблюдает. Основное различие в предложенных подходах — НС Мааса реализована на биологически подобных нейронах (импульсных), а НС Джагера — на формальных сигмоидальных нейронах. Эти две модели дали начало новому направлению «резервуарных» НС.

«Резервуарными» НС названы потому, что в этих моделях выделяется некоторое множество нейронов (резервуар), и они соединяются между собой случайным образом (как правило, задается степень связности или иной

параметр, через который определяется полнота связей НС) и это множество нейронов и связей неизменно (не обучается) — обучение проходят связи с некоторым наблюдателем, который призван фиксировать значения на выходах выделенного множества нейронов. На некоторые из нейронов заданного множества подаются входные сигналы, которые изменяют динамику работы всего множества за счет множества межнейронных связей, слои внутри множества, как правило, не выделяются, то есть это случайная рекуррентная сеть.

Представляется, что этот подход заслуживает особого внимания, так как идея о том, что сложность структуры НС должна быть адекватна сложности решаемой ею задачи, в этом случае реализуется. Эволюцию подходов к созданию памяти можно представить следующим образом. Вначале множество хранимых (подлежащих распознаванию) образов кодировалось множеством весовых коэффициентов НС (**сети статические**), далее множеством образов — множеством точек притяжения (**точечных аттракторов**) в фазовом пространстве динамической системы (НС Хопфилда и НС Хакена), затем множеством замкнутых траекторий в фазовом пространстве (**аттракторы типа замкнутый цикл, тор**), и наконец, множеством траекторий, определяющих лишь местоположение в фазовом пространстве в виде бесконечного числа сменяющихся состояний (**хаотические аттракторы**). Последний подход может быть определен как использование систем с неустойчивой динамикой, подразумевая под этим отсутствие устойчивых состояний, к которым систем приходит со временем.

Основная сложность НС с «резервуарами» состоит в том, что случайно сгенерированная структура не гарантирует того, что она будет производить требуемую динамику. Здесь необходимо оговориться, что значит — требуемую динамику. Как показал анализ проведенных экспериментов по решению задач на «резервуарных» НС необходимо с одной стороны, разнообразие состояний системы, чтобы она не оказалась устойчивой, и с другой стороны, динамика должна быть восприимчива к действию входных сигналов и не обладать турбулентными режимами, которые не позволяют выделять реакцию системы на входные сигналы. Оперируя определениями Пригожина — система должна быть на границе между порядком (упорядоченный режим работы по Канеко) [12, 13] и хаосом (имеется ввиду турбулентный режим по Канеко), т.е. производить детерминированный хаос (в частично-упорядоченном режиме по Канеко). Причем как было показано в ряде работ, эта граница представляет собой некоторый интервал возможных значений параметров системы.

Перспективные направления — хаос в порождении виртуальных структур обработки информации

Получить достаточно простую с точки зрения задания связей структуру НС, обладающую сложным поведением можно посредством использо-

вания в рекуррентной нейронной сети нейронов с передаточной функцией, порождающей хаотические колебания выхода нейрона. С одной стороны дискретных детерминированных преобразований, порождающих хаос немало, и, с другой стороны, многие из них достаточно просты. Такой подход оказывается в русле с подходами, в которых усложняют нейрон, повторяя или биологический прототип, или включая множество логических операций в базис отдельного нейрона.

Наиболее распространенным среди дискретных преобразований, порождающих хаотические последовательности, является логистическое отображение. Это связано с тем, что при достаточно простой записи и простой реализации в нем присутствует параметр, позволяющий регулировать степень хаотичности порождаемых последовательностей.

Широкое исследование множества связанных логистических отображений проводится в области физики, теории динамических систем и нелинейной динамике. Различным вариантам организации таких систем и снятию характеристик динамических режимов как глобально, так и локально связанных отображений посвящено множество исследований. Их результаты могут быть частично задействованы при разработке НС на основе логистических отображений, но при этом потребуется исследование влияния внесения в систему внешних воздействий и способов интерпретации измененных режимов работы под этими воздействиями [12]. То есть встает вопрос о включении в систему входных сигналов.

Отчасти такие исследования уже проведены в области решения задач кластеризации [10]. А для построения универсальной системы ИИ необходимо рассматривать ее не изолировано, а в той среде, в которой предполагается ее использовать. Об этом же сделан вывод и в работе, посвященной исследованию роли хаоса в НС [16], и во многих работах связанных с разработкой интеллектуальных агентов [5]. Тогда естественны образом можно будет подойти и к ключевому понятию — контексту, необходимому при обобщении, накоплении и интерпретации информации, как контекст.

Заключение

Математические методы нелинейной динамики и хаоса можно рассматривать как следующий этап развития математических методов, в том числе и в ИИ, если прослеживать тенденцию перехода от детерминированных к стохастическим моделям, и далее к хаотическим. Хаотические модели могут быть и детерминированными, но за счет нелинейностей и большого числа элементов приводят к сложному и зачастую трудно прогнозируемому поведению. Однако, именно кооперативные эффекты хаотических систем приводят к процессам самоорганизации, которые как раз требуются для разработки новых систем ИИ.

Литература

1. *Люгер Дж.* Искусственный интеллект. Стратегии и методы решения сложных проблем. Вильямс, 2005.
2. *Cristianini N.* Are we still there? // Neural Networks. No. 23. 2010. Pp. 466–470.
3. *Goertzel B., Ruiting L., Itamar A., Hugo G., Shuo C.* A world survey of artificial brain projects, PartII: Biologically inspired cognitive architectures // Neurocomputing. No. 74. 2010. Pp. 30–49.
4. *Mira J. M.* Symbols versus connections: 50 years of artificial intelligence // Neurocomputing. No. 71. 2008. Pp. 671–680.
5. *Рассел С., Норвиг П.* Искусственный интеллект: Современный подход. Вильямс, 2007.
6. *Lin C., Lee G.* Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems. Prentice Hall, 1996.
7. *Baum S. D., Goertzel B., Goertzel T.* How long until human-level AI? Results from an expert assessment // Technological Forecasting & Social Change. No. 78. 2011. Pp.185–195.
8. *Haken H.* Synergetic Computers and Cognition: A Top-Down Approach to Neural Nets. SSS: Springer, 2010.
9. *Хайкин С.* Нейронные сети: Полный курс. Вильямс, 2006.
10. *Benderskaya E. N., Zhukova S. V.* Clustering by chaotic neural networks with mean field calculated via Delaunay triangulation // Lecture Notes in Artificial Intelligence. Vol. 5271, Springer, 2008. Pp. 400–416.
11. *Jaeger H., Maass W., Principe J.* Introduction to the Special Issue on Echo State Networks and Liquid State Machines. Neural Networks. No. 20 (3), 2007. Pp.287–289.
12. *Inoue M., Kaneko K.* Dynamics of Coupled Adaptive Elements: Bursting and Intermitteny Oscillations Generated by Frustration in the Network // Phys. Rev. E, 81. 2010. Pp.126–203.
13. *Пригожин И. Р., Стенгерс И.* Порядок из хаоса. Новый диалог человека с природой. М., 1986.
14. *Потапов А. Б., Али М. К.* Нелинейная динамика обработки информации в нейронных сетях // Известия Высших Учебных Заведений: Прикладная нелинейная динамика. Т. 9. Вып. 6. 2001. С. 3–44.

РЕАЛИЗАЦИЯ АЛГОРИТМА ЛОКАЛЬНОГО ГОЛОСОВАНИЯ ДЛЯ МУЛЬТИАГЕНТНОГО УПРАВЛЕНИЯ В УСЛОВИЯХ СТОХАСТИЧЕСКИХ НЕОПРЕДЕЛЁННОСТЕЙ

Е. Храбрых

студентка; ekaterina.khrabrykh@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В работе рассматривается задача достижения консенсуса в децентрализованной сети интеллектуальных агентов при неполной информации о текущих состояниях узлов и переменной структуре связей. Для решения используется алгоритм локального голосования, реализованный с помощью среды для разработки мультиагентных систем (MAC), что наглядно показывает его работоспособность.

Введение

В последнее время задачи распределённого взаимодействия в сетях динамических управляемых агентов привлекает внимание всё большего числа исследователей. Для решения таких задач зачастую используют мультиагентные технологии, под которыми понимают как технологии разработки и использования мультиагентных систем, так и мультиагентное управление.

Для группы взаимодействующих агентов, обменивающихся с задержкой неполной информацией в дискретные моменты времени, при изменяющейся топологии связей в [1–3] для достижения консенсуса предложен и обоснован алгоритм локального голосования с изменяющимся размером шага. При этом открытым остается вопрос о «чувствительности» работы алгоритмов по отношению к выбираемым параметрам размеров шагов. Теоретическое исследование этого вопроса достаточно сложно, но для начала актуально исследование этой зависимости с помощью методов иммитационного моделирования.

Для моделирования в этой работе была использована программная среда JADE (Java Agent Development Framework). JADE — широко используемая программная среда для создания мультиагентных систем и приложений, поддерживающая FIPA-стандарты для интеллектуальных агентов. Она включает в себя среду выполнения агентов (агенты регистрируются и работают под управлением среды), библиотеку классов, которые используются для разработки агентных систем, набор графических утилит для администрирования и наблюдения за жизнедеятельностью активных агентов.

Для иллюстации полученных результатов работы алгоритма, используются графики, постороенных с помощью библиотеки JFreeChart.

Алгоритм локального голосования в задаче достижения консенсуса на графе

Следуя [2], рассмотрим сетевую систему, состоящую из набора динамических подсистем (агентов), взаимодействующих в соответствии с ориентированным графом. Сопоставим каждому узлу i графа изменяющийся во времени вектор состояния $x_t^i \in \mathsf{R}$. Структура связей динамической сети описывается с помощью последовательности орграфов $\{G_t = (N, E_t)\}_{t\geq 0}$, где $E_t \subseteq E$ меняется во времени. Множеством соседей узла i назовём $N^i = \{j : (j, i \in \mathrm{E})\}$.

Пусть $u_t^i \in \mathsf{R}$ — управляющие воздействия, которые в момент времени t воздействуют на узел i. Будем считать, что изменения в дискретном времени $t = 0, 1, 2, \ldots$ состояний узлов описываются разностными уравнениями:

$$x_{t+1}^i = x_t^i + f^i(x_t^i, u_t^i), \quad i \in N, \tag{1}$$

в которых $f^i(\cdot)$ — некоторые функции.

Для формирования стратегии управления каждый узел $i \in N$ имеет информацию о своем собственном состоянии

$$y_t^{ii} = x_t^i + w_t^{ii}, \tag{2}$$

(может быть и зашумленную) и, если $N_t^i \neq \varnothing$, зашумленные наблюдения о состояниях соседей:

$$y_t^{ij} = x_{t-d_t^{ij}}^j + w_t^{ij}, \quad j \in N_t^i, \tag{3}$$

где w_t^{ij} — помехи, а $0 \leq d_t^{ij} \leq \overline{d}$ — целочисленная задержка, $\overline{d}$ — максимально возможная задержка. Так как система начинает работу при $t = 0$, неявным требованием к множеству соседей будет:

$$j \in N_t^i \Rightarrow t - d_t^{ij} \geq 0. \tag{4}$$

Определение 1. *Будем называть протоколом (алгоритмом) управления с топологией G_t обратную связь по наблюдениям состояний*

$$u_t^i = K_t^i(y_t^{ij_1}, \ldots, y_t^{ij_{m_i}}), \tag{5}$$

где множество $\{j_1, \ldots, j_{m_i}\} \in \{i\} \bigcup \overline{N}_t^i$, $\overline{N}_t^i \subseteq N_t^i$.

Широко используется протокол управления, известный под названием «протокол локального голосования»:

$$u_t^i = \alpha_t \sum_{j \in \overline{N}_t^i} b_t^{ij}(y_t^{ij} - y_t^{ii}), \tag{6}$$

в котором управляющий вход каждого узла зависит от взвешенной суммы разницы его состояния и информации о состояниях его соседей. Здесь $\alpha_t > 0$ — размеры шагов протокола управления (6), $b_t^{ij} > 0 \ \ \forall j \in \overline{N}_t^i$.

Задача о консенсусе заключается в согласовании состояний всех узлов сети между собой. В [3] сформулированы и доказаны достаточно общие условия достижения консенсуса в сети при применении алгоритма локального голосования (6).

Программная реализация сети агентов

Разработка агентов в JADE похожа на разработку обычных Java-приложений. Для реализации алгоритма использовалась традиционная интегрированная среда разработки для Java — Eclipse. Любой агент в системе JADE является классом-наследником класса jade.core.Agent. В наследнике достаточно переопределить метод setup (), в котором агент настраивается для последующей деятельности. Деятельность агента осуществляется с помощью его поведения, наследуемого от абстрактного класса Behaviour. Агенты обмениваются информацией с помощью сообщений ACLMEssage (реализуемыми в среде Jade).

Программа начинает работу как обычное приложение, для того чтобы на основе входящих данных запустить Jade с нужными настройками. Система создаёт агента, отвечающий за время работы алгоритма, количество узлов полученной топологии (экземпляры класса JadeAgent.java) и агента, собирающего информацию об узлах (PlotterAgent.java), на основе которой строит графики. Каждому агенту известны все возможные соседние узлы с вероятностью существования связи между ними, своя производительность и начальная загрузка, дисперсия помех, максимально возможная задержка, размер шага из алгоритма локального голосования (параметр α).

Работа узла (*JadeAgent.java*) определяется тремя циклическими поведениями, которые работают одновременно. У каждого поведения реализован метод action (), в котором описана логика работы с сообщениями. В первом поведении при получении сигнала от агента времени (агент отправляет сигнал каждую секунду всем существующим агентам узлам в системе), агент отправляет сообщение всем видимым в данный момент соседям с информацией о своей загрузке (с учётом задержек). Во втором поведении агент на основе полученной информации о загрузке узла, которым является соседним, высчитывает функцию управления по алгоритму локального голосования. Отправляет запрос об изменении своей загрузки узлу, который прислал информацию. По результату запроса принимает решение, менять ли загрузку узла. Этот запрос обрабатывает третье поведение.

PlotterAgent собирает информацию о загруженности каждого узла в каждый момент времени, на основе которой можно построить графики.

Взаимодействие агентов можно проследить с помощью графических утилит Jade (см. рис. 1).

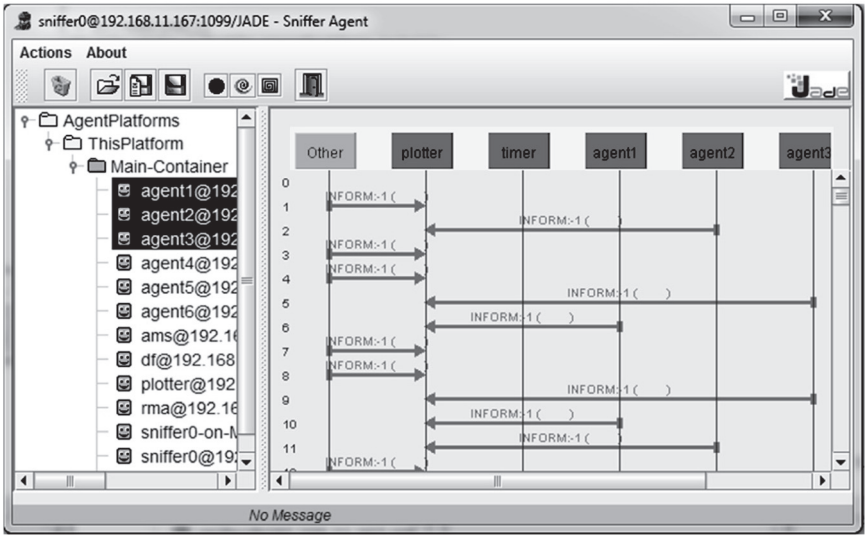


Рис. 1. Окно Sniffer Agent

Результаты моделирования

Во время работы алгоритма можно получить график сходимости загрузки каждого узла к сбалансированному значению с течением времени. Также для характеристики работы алгоритма введёна ошибка оценивания $Err = \sum_i \sqrt{\frac{(x_t^i - x^{\text{å}})^2}{n}}$ (средняя невязка). Ниже приведены графики при разных размерах шага из алгоритма локального голосования (параметр α) (см. рис. 2).

По полученным графикам видно, что при малых значениях параметра движение к консенсусу происходит очень медленно. С ростом параметра скорость сходимости увеливается, но до определённого момента, далее система становится не стабильной.

Заключение

Полученная модель работы алгоритма позволяет исследовать границы применимости алгоритма в условиях стохастических неопределённостей. Результаты моделирования показали, что для конкретной задачи есть свой размер шага из алгоритма локального голосования (параметр α), при котором система максимально быстро приходит к консенсусу. Дальнейшие исследования будут направлены на автоматическую оптимизацию параметра α.

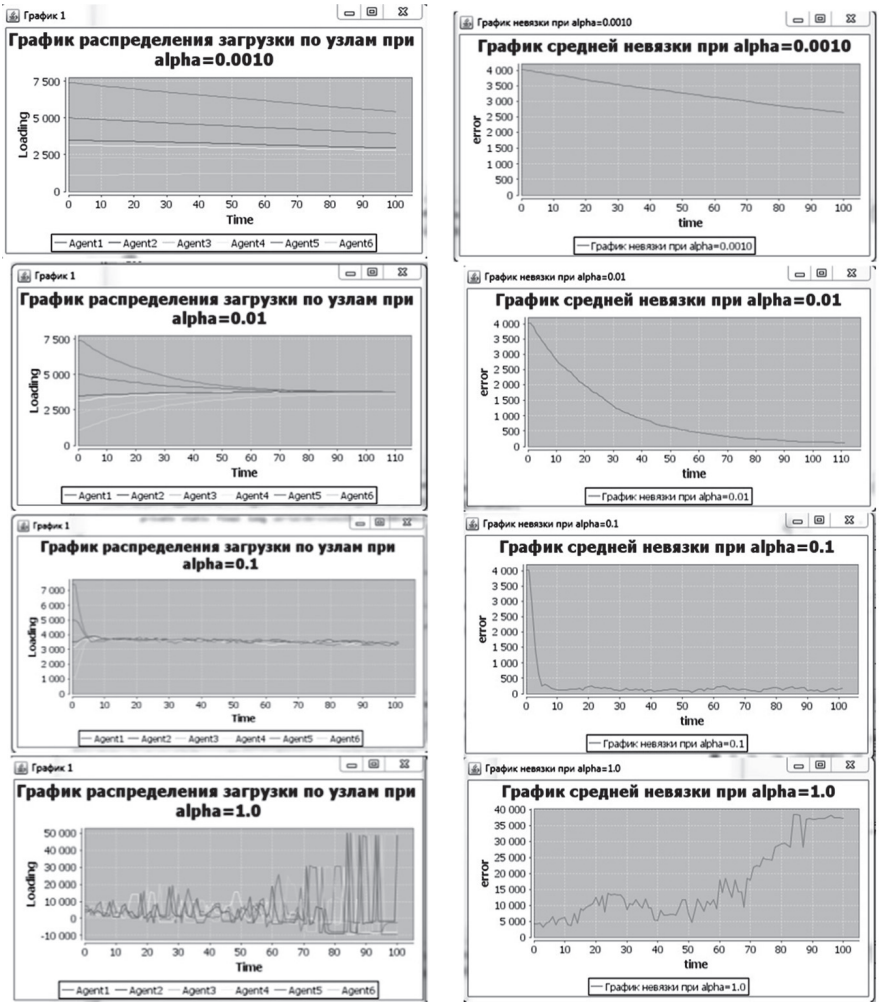


Рис. 2. Графики, полученные во время работы алгоритма

Литература

1. *Huang M.* Stochastic approximation for consensus with general time-varying weight matrices // Proc. of the 49th IEEE Conf. on Decision and Control (CDC-49). Dec. Atlanta. USA. 2010. P. 7449–7454.
2. *Амелина Н. О, Фрадков А. Л.* Метод усредненной модели в задаче достижения консенсуса // Стохастическая оптимизация в информатике. Т. 8 (1). 2012. С. 3–20.

3. *Амелина Н. О.* Мультиагентные технологии, адаптация, самоорганизация, достижение консенсуса // Стохастическая оптимизация в информатике. Т. 7. 2011. С. 149–185.
4. *Амелина Н. О.* Балансировка загрузки узлов децентрализованной вычислительной сети при неполной информации // Нейрокомпьютеры: разработка, применение. 2011. 6. С. 56–63.
5. *Вахитов А. Т., Граничин О. Н., Гуревич Л. С.* Алгоритм стохастической аппроксимации с пробным возмущением на входе в нестационарной задачи оптимизации // Автоматика и телемеханика. 2009. № 11. С. 70–79.

Параллельные алгоритмы и вэйвлетная обработка числовых потоков

Демьянович
Юрий Казимирович

д.ф.-м.н., профессор
заведующий кафедрой параллельных алгоритмов СПбГУ

О ЛАВИННОМ ЭФФЕКТЕ В АЛГОРИТМАХ СПЛАЙН-ВЕЙВЛЕТНОГО ШИФРОВАНИЯ

К. Н. Быков

аспирант 2 курса кафедры параллельных алгоритмов;
Konstantin.Bykov@emc.com

Санкт-Петербургский государственный университет

Аннотация: Исследуются свойства криптографического преобразования, построенного на базе сплайн-вейвлетного разложения третьего порядка. Доказывается, что из описания алгоритма циклический сдвиг можно исключить без потери стойкости. Изучена структура раундового преобразования.

Введение

Идея о применении алгоритмов сплайн-вейвлетного разложения впервые была озвучена в работе [1]. Впоследствии, на базе приведенного аппарата, разработано несколько симметричных алгоритмов шифрования [2]. В данной статье мы ограничимся рассмотрением одного из них — шифрованием с использованием вейвлетов третьего порядка (алгоритмы шифрования с использованием вейвлетов второго и перового порядков аналогичны рассмотренному ниже и имеют принципиально схожую схему преобразования).

До настоящего времени объем исследований о стойкости и свойствах сплайн-вейлветного шифрования (далее СВШ) ограничивается материалами [2] и [3]. Настоящая статья имеет целью продолжение данного исследования и расширение круга рассмотрения криптографических характеристик раундового преобразования в СВШ.

Основные определения

Описание алгоритма шифрования

Обозначим за X ($x_0 < x_1 < \ldots < x_M$) неравномерную сетку и порядок выбрасывания узлов их этой сетки γ. Ключом шифрования будем считать пару $K = K(X, \gamma)$. Количество производимых раундов шифрования обозначим R. Все вычисления проводятся в конечных полях по модулю N (N — простое число, передаваемое вместе с шифротекстом). Открытым текстом назовем последовательность l блоков одинаковой длины M: $C = \{c_i\}_{i \in J}$, $J \subset \mathbf{Z}$. Здесь

и далее под СВШ будем понимать R-кратное применение следующего алгоритма к блокам $\{c_i\}_{i \in J}$.

Алгоритм.

1. Из сетки X^r, полученной на предыдущем шаге (X для первого раунда) выбрасывается узел x_{γ_r}.
2. Полученная сетка обозначается X^{r-1}, а ее узлы, соответственно, x_j^{-r}:

$$x_j^{-r-1} = x_j^{-r} \text{ при } j < \gamma_r,$$

$$x_j^{-r-1} = x_{j+1}^{-r} \text{ при } j > \gamma_r,$$

$$X^{-r-1} = \left\{x_j^{-r-1}\right\}_{j=1,\ldots,\,L-r-1}.$$

Выбрасываемый узел обозначается $x_{\gamma_r} = \xi$.

3. Производится расчет по формулам декомпозиции:

$$c_i^{-r-1} = c_i^{-r} \text{ при } 0 \le i \le \gamma_r - 4,$$

$$c_i^{-r-1} = c_{i+1}^{r} \text{ при } \gamma_r - 1 \le i \le M - r - 1,$$

$$c_{\gamma_r-3}^{-r-1} = \left(\frac{\xi - x_{\gamma_r}^{-r-1}}{\xi - x_{\gamma_r-3}^{-r-1}} \cdot c_{\gamma_r-4}^{-r} + \frac{x_{\gamma_r}^{-r-1} - x_{\gamma_r-3}^{-r-1}}{\xi - x_{\gamma_r-3}^{-r-1}} \cdot c_{\gamma_r-3}^{-r}\right)(mod N),$$

$$c_{\gamma_1-2}^{-r-1} = \Big((\xi - x_{\gamma_r}^{-r-1})(\xi - x_{\gamma_r+1}^{-r-1}) \cdot c_{\gamma_r-4}^{-r} + (\xi - x_{\gamma_r+1}^{-r-1})(x_{\gamma_r}^{-r-1} - x_{\gamma_r-3}^{-r-1}) \cdot c_{\gamma_r-3}^{-r} +$$

$$+ (x_{\gamma_r+1}^{-r-1} - x_{\gamma_r-2}^{-r-1})(\xi - x_{\gamma_r-3}^{-r-1}) \cdot c_{\gamma_r-2}^{-r}\Big) \cdot [\xi - x_{\gamma_r-2}^{-r-1}]^{-1} [\xi - x_{\gamma_r-3}^{-r-1}]^{-1} (mod\ N),$$

$$b^{-1} = \left(c_{\gamma_r-1}^{-r} - \frac{x_{\gamma_r+2}^{-r-1} - \xi}{x_{\gamma_r+2}^{-r-1} - x_{\gamma_r-1}^{-r-1}} \cdot c_{\gamma_r-2}^{-r-1} - \frac{\xi - x_{\gamma_r-1}^{-r-1}}{x_{\gamma_r+2}^{-r-1} - x_{\gamma_r-1}^{-r-1}} \cdot c_{\gamma_r-1}^{-r-1}\right)(mod N).$$

4. Если раунд — не последний, производится циклический сдвиг элементов полученной последовательности $\{c_i^{-r-1}\}_{i=0,\ldots,\,M-r-1}$

$$c_0^{-r-1} \to c_1^{-r-1} \to c_2^{-r-1} \to \ldots \to c_{M-r-1}^{-r-1} \to c_0^{-r-1}.$$

5. Если раунд — последний, сдвиг не производится, а полученная пара последовательностей $\{c_i^{-R}\}_{i=0,1,\ldots,M-R-1} ub^{-R}$ называется шифротекстом.

Замечание. *Предполагается, что сетка X и промежуточный шифротекст* $\{c_i^{-r-1}\}_{i=0,\ldots,\,M-r-1}$ *продолжены циклическим способом.*

Замечание. *Опустим описание процесса дешифрования, т.к. на дальнейшие рассуждения он не влияет.*

Понятие лавинного эффекта и критерия

Под *лавинный эффектом* понимается способность алгоритма шифрования приводить к значительным изменениям в выходной последовательности при незначительных изменениях во входной — данное свойство край-

не важно для противостояния корреляционным атакам на блочные шифры. Для формализации понятия лавинного эффекта введем понятие расстояния, аналогичное расстоянию Хемминга.

Определение. *Расстоянием* между двумя текстами $C = \{c_i\}_{i=0,\ldots,M}$ и $D = \{d_i\}_{i=0,\ldots,M}$ одинаковой длины назовем количество несовпадающих элементов в этих текстах, стоящих на одинаковой позиции.

$$\rho(C, D) = \left|\Omega\right|, \quad \Omega = \{i \mid c_i^1 d_i,\ i = \overline{0,\ L}\}.$$

При оценке лавинного эффекта под входной последовательностью в общем случае понимается пара ключ + шифруемый текст, мы же будем рассматривать случай, когда ключ шифрования одинаковый, а два текста различаются одним байтом.

Определение. Пусть тексты $C = \{c_i\}_{i=0,\ldots,M-1}$ и $D = \{d_i\}_{i=0,\ldots,M-1}$ отличаются единственным байтом в позиции $m\,(m = \overline{1,\ M-1})$. *Лавинным эффектом по тексту* для четверки (E, C, K, D) назовем следующее расстояние:

$$\Delta\big(E,\ C,\ K,\ D\big) = \rho\big(E\big(C,\ K\big),\ E\big(D,\ K\big)\big),$$

где E — алгоритм шифрования, K — ключ шифрования.

В веденных обозначениях лавинный эффект будем считать случайной величиной. Тогда, известное определение лавинного критерия [4] можно обобщить следующим образом.

Определение. Будем считать, что шифр удовлетворяет *лавинному критерию по тексту*, если математическое ожидание значения лавинного эффекта равняется половине длины шифруемого сообщения.

$$M\left(\Delta(E,\ C,\ K,\ D)\right) = L/2.$$

Свойства раундового преобразования СВШ

Влияние циклического сдвига

Выясним, каким образом циклический сдвиг влияет на процесс шифрования. Обозначим введенный выше алгоритм СВШ как *A1*, а также рассмотрим «упрощенный» алгоритм *A2*, в котором циклический вдиг производится $R-1$ раз а конце шифрования.

Пусть в алгоритмах *A1* и *A2* элемент открытого текста c_0 выбрасывается в раунде с номером r. До этого раунда сдвиг последовательности $\{c_i^{-r-1}\}_{i=0,\ldots,M-r-1}$ в *A1* будет произведен $r-1$ раз. Это значит, что для *A1* элемент c_0 окажется на позиции $r-1$, т. е., в принятых обозначениях, $\gamma_r = r-1$.

Если для алгоритмов *A1* и *A2* открытый текст и сетка *X* совпадают, то для раунда *r* узел с номером $r-1$ в *A1* совпадает с узлом 0 в *A2*. Если теперь обозначить для *A1* и *A2* номера выбрасываемых узлов на раунде *r* как γ_r^{A1} и γ_r^{A2} соответственно, то при выполнении условия

$$\gamma_r^{A1} = \left(\gamma_r^{A2} - k + 1\right) mod\ (M - r + 1), \tag{1}$$

все элементы $\{c_i^{-r-1}\}_{i=0,\ldots,M-r-1}$ для алгоритма *A1* будут смещены циклически по отношению к элементам в *A2*. Отсюда, по заданию алгоритма *A2*, следует, что при выполнении условия (1) шифротексты алгоритмов совпадают. Следовательно, алгоритм *A1* — при соответствующем выборе ключа — всегда можно заменить алгоритмом *A2*. Данное свойство позволяет не учитывать циклический сдвиг при дальнейшем рассмотрении раундового преобразования.

Влияние режима шифрования

Из определения алгоритма СВШ следует, что преобразования блоков в открытом тексте производятся независимо (режим Electronic Codebook ECB [4]). Очевидно, что в данном режиме величина лавинного эффекта по тексту не может превышать размера блока шифрования. Данное утверждение справедливо для всех блочных шифров, работающих в режиме ECB и, в частности, для СВШ.

Замечание. *Данное утверждение в общем случае неверно, если используется другой режим шифрования (CBC, OFB или CFB), или отличный байт входной последовательности принадлежит ключу.*

В описании алгоритма СВШ длина блока не является фиксированной и может быть например, соразмерно длине шифруемого сообщения. Следует заметить, что процесс увеличения длины блока не приводит к увеличению длины ключа только в том случае, если число раундов шифрования остается постоянным. Можно показать, что значение лавинного эффекта возрастает в зависимости от количества раундов и выбора ключа, но не зависит от выбранной длины блока. Это означает, что для достижения максимального значения лавинного эффекта количество раундов следует брать *максимальным*. Следовательно, длину блока следует ограничить сверху некоторым значением, при котором длина ключа является «разумной» (например, до 1024 б).

Раундовое распространение

В принятых обозначениях введем несколько дополнительных понятий.

Определение. Пусть $E^r(C^{-r-1}, K) = C^{-r}$, $E^r(D^{-r-1}, K) = D^{-r}$ $1 \le r \le R$ — промежуточный результат криптографического преобразования, где

$C^{-r} = (C_0^{-r}, \ldots, C_N^{-r})$, $D^{-r} = (D_0^{-r}, \ldots, D_N^{-r})$. *Раундовым распространением* будем называть множество позиций элементов, ставших отличными на раунде r:

$$\Theta_r = \{i \mid C_0^{-r-1} \neq D_0^{-r-1} \wedge C_0^{-r} = D_0^{-r}\}.$$

Замечание. *На первом данное распространение состоит из одного элемента — номера отличного байта m.*

Определение. *Областью распространения* для раунда r будем называть объединение всех множеств раундовых распространений, полученных до данного раунда:

$$\Theta'_r = \bigcup_{i=1}^{r} \Theta_i.$$

Замечание. *Значение лавинного эффекта соответствует количеству элементов в области распространения на последнем раунде.*

Определение. *Раундовым расстоянием* между промежуточными шифротекстами назовем количество элементов в раундовом распространении.

$$\rho_r(E, K, C, D) = |\Theta_r|.$$

Исследуем свойства раундового расстояния для СВШ.

По заданию алгоритма СВШ раундовое распространение содержит не более трех элементов шифротекста (при том, что выброшенный элемент может быть одним из отличающихся). Следовательно, раундовое расстояние для СВШ задается следующим образом:

$$\rho_r\left(E, K, C^{-r}, D^{-r}\right) = \rho\left(c_{\gamma_r-3}^{-r}, d_{\gamma_r-3}^{-r}\right) + \rho\left(c_{\gamma_r-2}^{-r}, d_{\gamma_r-2}^{-r}\right) + \rho\left(b_C^{-r}, d_D^{-r}\right) - \overline{\delta}_{0, c_{\gamma_r}, d_{\gamma_r}}, \quad (2)$$

где $\overline{\delta}$ — символ, обратный символу Кронекера.

Введем функции $\rho_r^{-3}, \rho_r^{-2}, \rho_r^{b}$:

$$\rho_r^{-3}(\gamma, C^{-r}, D^{-r}) = \begin{cases} 1, \text{ при } C^{-r}, D^{-r} \mid c_{\gamma_r-4}^{-r} \neq d_{\gamma_r-4}^{-r} \wedge\ c_{\gamma_r-3}^{-r} = d_{\gamma_r-3}^{-r} \\ 0, \text{ при всех остальных } C^{-r}, D^{-r} \end{cases},$$

$$\rho_r^{-2}(\gamma, C^{-r}, D^{-r}) = \begin{cases} 1, \text{ при } C^{-r}, D^{-r} \mid c_{\gamma_r-2}^{-r} = d_{\gamma_r-2}^{-r} \wedge (c_{\gamma_r-4}^{-r} \neq d_{\gamma_r-4}^{-r} \vee c_{\gamma_r-3}^{-r} \neq d_{\gamma_r-3}^{-r}) \\ 0, \text{ при всех остальных } C^{-r}, D^{-r} \end{cases},$$

$$\rho_r^{b}(\gamma, C^{-r}, D^{-r}) = \begin{cases} 1, \text{ при } C^{-r}, D^{-r} \mid c_{\gamma_r-2}^{-r} \neq d_{\gamma_r-2}^{-r} \vee c_{\gamma_r-1}^{-r} \neq d_{\gamma_r-1}^{-r} \\ 0, \text{ при всех остальных } C^{-r}, D^{-r} \end{cases}.$$

Нетрудно показать, что значения слагаемых в (2) равняются значениям введенных функций за исключением разве что случаев, когда они, в силу специального выбора сетки X, оказываются строго меньше. Это означает, что раундовое распространение зависит только от области распростра-

нения на предыдущем раунде и выбрасываемой точки на текущем, т. е. $\Theta_r = \Theta_r(\Theta_{r-1}, \gamma_r)$. В силу рекурсивности данной зависимости, а также явного задания расстояния (2), получаем, что лавинный эффект для СВШ можно считать — за исключением упомянутого случая, когда он строго меньше — функцией (m, γ).

Структура распространения изменений

Введенная выше область распространения изменений для СВШ состоит из объединения двух непересекающихся множеств, соответствующих распространению изменений в частях шифротекста C^{-r} и b^{-r}: $\Theta'_r = \Theta'_{C^{-r}} \cup \Theta'_{b^{-r}}$.

Сформулируем важную особенность раундового распространения в алгоритме СВШ в виде леммы, доказательство которой опустим за очевидностью.

Лемма. На любом раунде область распространения $\Theta'_{C^{-r}}$ состоит из последовательности возрастающих индексов.

Данная особенность означает локальность распространения изменений и позволяет оценить значение раундовые расстояния (2). Обозначим $m_{\min} = \min\left(\Theta'_{C^{-r}}\right)$, $m_{\max} = \max\left(\Theta'_{C^{-r}}\right)$ и пусть на раунде r в текстах C^{-r} и D^{-r} совпадают хотя бы 4 элемента, т.е. величина области распространения $\Theta'_{C^{-r}}$ не превышает значения $M - r - 4$. Учитывая результаты предыдущего пункта и производя последовательный перебор значений γ_r относительно диапазона индексов, входящих в $\Theta'_{C^{-r}}$, получаем следующую таблицу:

Таблица 1

Диапазон раундовых расстояний

γ_r	$\rho\left(c^{-r}_{\gamma_r-3}, d^{-r}_{\gamma_r-3}\right) + \rho\left(c^{-r}_{\gamma_r-2}, d^{-r}_{\gamma_r-2}\right) - \overline{\delta}_{0, c_{\gamma_r}, d_{\gamma_r}}$	$\rho\left(b^{-r}_C, d^{-r}_D\right)$	ρ_r
…	...	…	…
$m_{\min} - 1$	0	0	0
$m_{\min}$	–1	0	–1
$m_{\min} + 1$	–1	1	0
$m_{\min} + 2$	–1	1	0
…	…	…	…
$m_{\max}$	–1	1	0
$m_{\max} + 1$	0	1	1
$m_{\max} + 2$	0	1	1
$m_{\max} + 3$	1	0	1
$m_{\max} + 4$	2	0	2
$m_{\max} + 5$	0	0	0
…	…	…	…

В случае, если $m_{\min} = m_{\max}$ (как, например, при $r = 1$), выбирается строка $m_{\min}$. Отличие случаев $M - r - 3 \le |\Theta'_{C^{-r}}| \le M - r - 1$ состоит в том, что значения второй и третьей колонок не могут принимать значения (–1, 0), (0, 0) и (2, 0).

Если на некотором раунде область распространения вырождается $(|\Theta'_{C^{-r}}| = 0)$ или наоборот, достигает своего предела $(|\Theta'_{C^{-r}}| = M - r),$ для всех последующих раундов соотвествующая таблица будет состоять из нулей или троек (— 1, 1, 0) соответственно — выполнимость любого из данных условий означает *остановку* распространения изменений и все последующие раунды не влияют на значение лавинного эффекта.

Значение лавинного эффекта

Основываясь на результатах предыдущего раздела, выясним, удовлетворяет ли алгоритм СВШ введенному лавинному критерию при длине шифруемого сообщения, равной длине блока (от 16 до 1024 байт) и максимально возможном количестве раундов. Для этого попробуем оценить значение вероятности принятия величиной $|\Theta'_R|$ значения ξ при помощи следующей геометрической интерпретации распространения изменений.

На плоскости введем прямоугольную систему координат со значением $\Theta'_{C^{-r}}$ по оси абсцисс и $\Theta'_{b^{-r}}$ по оси ординат. Каждому случайному выбору выбрасываемой точки соответствует случайное раундовое расстояние, определяемое введенной выше таблицей. Сопоставим каждому варианту строки из этой таблицы вектор следующим образом $a = (0, 1)$, $b = (–1, –1)$, $c = (1, 0)$, $d = (0, 1)$, $e = (2, 0)$, $f = (0, 0)$. Тогда совокупности R раундов будет соответствовать цепочка векторов, выбранных случайным образом из множества $\{a, b, c, d, e, f\}$ R раз с учетом следующих особенностей:

- На шаге $i_0 : 1 \le i_0 \le R$ вектор b можно выбирать $|\Theta'_{C^{-i_0}}| - 1$ «способом». Если $|\Theta'_{C^{-i_0}}| \le 2$ вектор b выбирать нельзя.
- На шаге i_0 вектор f можно выбрать $M - |\Theta'_{C^{-i_0}}| - 4$ «способами».
- На шаге i_0 вектор c можно выбрать двумя «способами».
- Если на шаге i_0 выполнено $|\Theta'_{C^{-i_0}}| = 0,$ на всех последующих шагах можно выбирать только вектор f всеми возможными M — i, $i_0 \le i \le R$ «способами» (остановка распространения).
- Если на шаге i_0 выполнено $M - i_0 - 3 \le |\Theta'_{C^{-i_0}}| = M - i_0 - 1,$ на всех последующих шагах нельзя выбирать векторы a, e и f.
- Если на шаге i_0 выполнено $|\Theta'_{C^{-i_0}}| = M - i_0,$ на всех последующих шагах можно выбирать только вектор f всеми возможными M — i, $i_0 \le i \le R$ «способами» (остановка распространения).

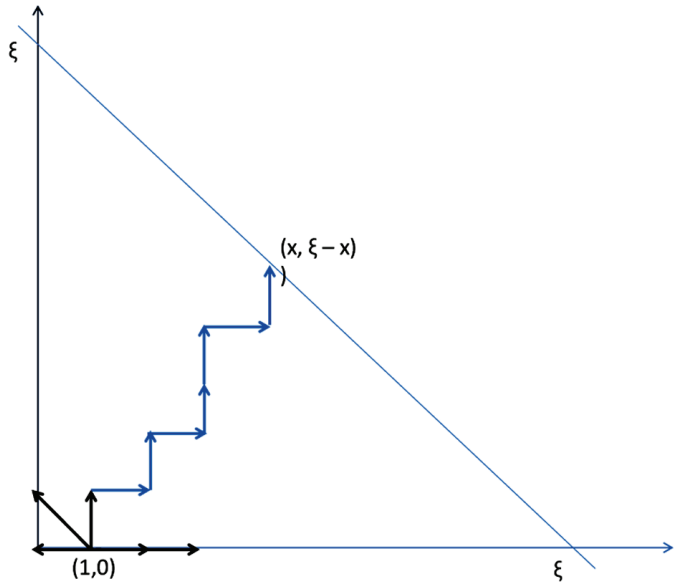


Рис. 1. Геометрическая интерпретация изменений

Таким образом, лавинный эффект принимает значение ξ в том случае, если построенная описанным способом цепочка векторов заканчивается на прямой $\Theta'_{C^{-r}} + \Theta'_{b^{-r}} = \xi$, а, следовательно, вероятность построения такой цепочки равняется искомой вероятности $P(|\Theta'_r| = \xi)$.

Для решения поставленной задачи осталось вычислить количество различных путей $S^{\xi}(x)$ из точки (1, 0) в точку $(x, \xi - x)$ $\forall x = \overline{0, \xi}$. Это довольно легко можно сделать методом рекурсивного спуска по номеру раунда. Опустим описание процесса вычисления $S^{\xi}(x)$ и воспользуемся формулой для вычисления искомого мат. ожидания (учитывая, что совокупное число возможных цепочек равняется $M!/4$):

$$M(|\Theta'_R|) = \sum_{i=0}^{M} \sum_{j=0}^{i} \frac{S^i(j)}{M!/4} \cdot i.$$

Производя необходимые вычисления, получаем результаты, представленные в табл. 2.

Из табл. 2 видно, что ни при одном из рассматриваемых значений M условие лавинного критерия не выполняется. Ближе остальных к нужному значению лавинный эффект наблюдается при размере блока 16, увеличение размера блока совместно с количеством производимых раундов увеличивают лавинный эффект незначительно.

Таблица 2

Значения лавинного эффекта

M	$M(\lvert\Theta'_R\rvert)$
16	7
32	10
64	12
128	14
256	16
512	18
1024	21

Заключение

Как можно видеть из результатов последнего пункта, увеличение размера блока, по всей видимости, не дает желаемого результата — наоборот, для обеспечения лучшей «диффузии», следует брать как можно меньший размер блока, получая взамен проблемы, связанные с режимом ECB. Полученное значение лавинного эффекта для СВШ объясняется тем, что на каждом раунде получаемый шифротекст зависит только от трех байт результата предыдущего раунда. Очевидно, что в таких условиях следует выполнять столько раундов, сколько позволяет длина блока, однако неизвестно, насколько подобная стратегия скажется на общей производительности шифра.

Как было сказано выше, низкое значение лавинного эффекта свидетельствует о высокой корреляции между открытым и зашифрованным текстом, вполне вероятно, что какая-либо из атак на основе подобранных или связанных текстов будет иметь успех. Вне рамок рассмотрения оставлено влияние отличных от ECB режимов шифрования на лавинный эффект.

Полученные результаты подтверждаются статистическими тестами. Для иллюстрации результатов следует построить репрезентативную выборку с указанием частотных характеристик.

Литература

1. *Демьянович Ю. К., Левина А. Б.* Вэйвлетные разложения и шифрование // Сб. Методы вычислений. Вып. 22. СПб.: Изд-во С.-Петербургского ун-та, 2008. С. 56–64.
2. *Левина А. Б.* Сплайн-вэйвлеты и их некоторые применения. Автореферат диссертации на соискание степени кандидата физико-математических наук. СПб.: Изд-во С.-Петербургского ун-та, 2009.
3. *Кустов В. А.* Криптоанализ алгоритмов шифрования, основанных на сплайн-вейвлетных разложениях, Аннотированный сборник научно-исследовательских выпускных квалификационных работ выпускников СПб НИУ ИТМО / Под ред. Л. М. Студеникина. СПб: СПб НИУ ИТМО, 2012. 83 с.
4. *Шнайер Б.* Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. М.: Триумф, 2002. С. 221–222. 816 с.

СТОЯЧИЕ ВОЛНЫ В СПЛАЙН-ВЭЙВЛЕТНОМ РАЗЛОЖЕНИИ [1]

Ю. К. Демьянович

зав. кафедрой параллельных алгоритмов;
Yuri.Demjanovich@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Для двухгнездового сплайн-вэйвлетного разложения рассмотрен эффект интерференции и образования стоячих волн в вэйвлетном потоке, позволяющий при передаче вэйвлетных компонент по каналам связи уменьшить объем передачи на одну треть. Установлено, что для исходного потока, порожденного гладкой функцией, вэйвлетный поток имеет порядок второго дифференцила упомянутой функции.

Исследования в области вэйвлетных разложений числовых информационных потоков требуют разработки гибкого аппарата сплайн-вэйвлетных аппроксимаций с учетом свойств гладкости и скорости изменения упомянутых потоков. Для этого требуется использовать сплайновые аппроксимации различных порядков, неравномерные сетки, разложения со свойствами локализации в отдельных частях рассматриваемой области и применять гнездовые разложения для экономии вычислительных ресурсов с учетом возможностей параллельных вычислительных систем (см. работу [1]).

Цель данной работы состоит в том, чтобы в случае двухгнездового сплайн-вэйвлетного разложения рассмотреть эффект интерференции в вэйвлетном потоке и исследовать эффект, который состоит в подобии компонент вэйвлетного разложения: оказывается, что для любого поступающего потока одна из компонент может быть вычислена из другой компоненты умножением последней на некоторую константу. В частности, если речь идет о передаче вэйвлетных компонент по каналам связи, то выявленный эффект позволяет уменьшить объем передачи на одну треть. Показано, что для потоков, порождаемых дважды непрерывно дифференцируемой функцией, вэйвлетный поток имеет порядок второго дифференцила упомянутой функции.

Предварительные сведения

Для любого натурального числа m введем обозначения

$$J_m \stackrel{\text{def}}{=} \{0, 1, ..., m\}, \quad J'_m \stackrel{\text{def}}{=} \{0, 1, ..., m\}.$$

[1] Работа частично поддержана грантами РФФИ 10-01-00297 и 10-01-00245.

Пусть N — натуральное число. Рассмотрим сетку

$$X: \quad a = x_0 < x_1 < \ldots < x_{N-2} < x_{N-1} < x_N = b. \tag{1}$$

Рассмотрим полную систему векторов $\{\mathbf{a}_i\}_{i \in J'_{N-1}}$ пространства $\mathbb{R}^2$, т. е. такую, что $\det(\mathbf{a}_{j-1}, \mathbf{a}_j) \neq 0$ при $j \in J'_{N-1}$.

По двухкомпонентной вектор-функции $\varphi(t)$ с линейно независимыми компонентами на любом интервале (a', b') определим функции $\omega_j(t)$, $j \in J'_{N-1}$, с помощью аппроксимационных соотношений

$$\sum_{j \in J'_{N-1}} \mathbf{a}_j \omega_j(t) = \varphi(t), \quad \omega_j(t) \equiv 0 \quad \text{при} \quad t \notin (x_j, x_{j+2}), \quad j \in J'_{N-1};$$

таким образом, функции $\omega_j(t)$, $j \in J'_{N-1}$, определены следующим образом:

$$\omega_{-1}(t) = \begin{cases} \dfrac{\det(\varphi(t), \mathbf{a}_0)}{\det(\mathbf{a}_{-1}, \mathbf{a}_0)} & \text{при } t \in (x_0, x_1), \\ 0 & \text{при } t \notin [x_0, x_1], \end{cases}$$

для $j \in J_{N-2}$

$$\omega_j(t) = \begin{cases} \dfrac{\det(\mathbf{a}_{j-1}, \varphi(t))}{\det(\mathbf{a}_{j-1}, \mathbf{a}_j)} & \text{при } t \in (x_j, x_{j+1}), \\ \dfrac{\det(\varphi(t), \mathbf{a}_{j+1})}{\det(\mathbf{a}_j, \mathbf{a}_{j+1})} & \text{при } t \in (x_{j+1}, x_{j+2}), \\ 0 & \text{при } t \notin [x_j, x_{j+2}], \end{cases}$$

$$\omega_{N-1}(t) = \begin{cases} \dfrac{\det(\mathbf{a}_{N-2}, \varphi(t))}{\det(\mathbf{a}_{N-2}, \mathbf{a}_{N-1})} & \text{при } t \in (x_{N-1}, x_N), \\ 0 & \text{при } t \notin [x_{N-1}, x_N], \end{cases}$$

Дальше рассматривается пространство $\mathbb{S}_N \overset{\text{def}}{=} L\{\omega_j\}_{j \in J'_{N-1}}$, где $L\{\ldots\}$ — линейная оболочка функций, указанных в фигурных скобках. Заметим, что ввиду предположений относительно вектор-функции $\varphi(t)$ функции $\omega_j(t)$, $j \in J'_{N-1}$, линейно независимы и, следовательно, являются базисом пространства $\mathbb{S}_N$; поэтому $\dim \mathbb{S}_N = N+1$.

Из сетки (1) удалим узлы x_{k-1} и x_{k+1}, так что укрупненная сетка имеет вид $\tilde{X}: a = \tilde{x}_0 < \tilde{x}_1 < \tilde{x}_2 < \ldots < \tilde{x}_{\tilde{N}-1} < \tilde{x}_{\tilde{N}} = b$, (здесь $\tilde{N} = N-2$, а $\tilde{x}_j = x_j$ при $0 \le j \le k-1$, $\tilde{x}_k = x_{k+1}$, $\tilde{x}_j = x_{j+2}$ при $k+1 \le j \le \tilde{N}$).

Кроме того, рассмотрим полную цепочку векторов $\tilde{A} \overset{\text{def}}{=} \{\tilde{\mathbf{a}}_{-1}, \tilde{\mathbf{a}}_0, \ldots, \tilde{\mathbf{a}}_{\tilde{N}-1}\}$, предполагая, что выполнено условие

(*A*) Справедливы соотношения

$$\tilde{\mathbf{a}}_j = \mathbf{a}_j \ \text{ при } \ 0 \le j \le k-2, \quad \tilde{\mathbf{a}}_{k-1} = \mathbf{a}_k,$$

$$\tilde{\mathbf{a}}_j = \mathbf{a}_{j+2} \ \text{ при } \ k \le j \le \tilde{N}-1.$$

Рассмотрим два гнезда (см. [1]) $\Gamma_1 \stackrel{\text{def}}{=} \{x_{k-1}\}$ и $\Gamma_2 \stackrel{\text{def}}{=} \{x_{k+1}\}$. Обозначим $\Gamma \stackrel{\text{def}}{=} \{\Gamma_1, \Gamma_2\}$.

Построим систему функций $\{\tilde{\omega}_j\}_{j \in J'_{N-3}}$, отыскивая ее из соотношений $\sum_{j \in J'_{N-3}} \tilde{\mathbf{a}}_j \tilde{\omega}_j(t) = \varphi(t) \quad \forall t \in \tilde{G}, \quad \tilde{\omega}_j(t) \equiv 0 \quad \forall t \in \tilde{G} \setminus \tilde{S}_j, \ j \in J'_{N-3}$. и обозначим $\mathbb{S}_{N\{\Gamma\}} \stackrel{\text{def}}{=} \mathcal{L}\{\tilde{\omega}_j\}_{j \in J'_{N-3}}$. Очевидно, что $\dim \mathbb{S}_{N\{\Gamma\}} = N-1$.

Теорема 1. *При условии (A) справедливы следующие утверждения:*

1) если $s+1 \le k-2$, то $\omega_s(t) \equiv \tilde{\omega}_s(t)$ при $s \le k-3$,
2) если $k \le s-1$, то $\omega_s(t) \equiv \tilde{\omega}_{s+2}(t)$ при $s \ge k+1$.

Рассмотрим матрицу $P_{N\{\Gamma\}}$ с элементами $p_{i,j}$, элементы которой вычисляются по формулам

$$p_{i,j} = \delta_{i,j} \ \text{ при } \ -1 \le j \le k-2, \qquad p_{i,j} = \delta_{i,j-2} \ \text{ при } \ k+2 \le j \le N-1,$$

$$p_{k-2,k-1} = \frac{\det(\mathbf{a}_{k-1}, \mathbf{a}_k)}{\det(\mathbf{a}_{k-2}, \mathbf{a}_k)}, \quad p_{k-1,k-1} = \frac{\det(\mathbf{a}_{k-2}, \mathbf{a}_{k-1})}{\det(\mathbf{a}_{k-2}, \mathbf{a}_k)},$$

$$p_{i,k-1} = 0 \ \text{ при } \ i \in J'_{N-3} \setminus \{k-2, k-1\},$$

$$p_{k-1,k+1} = \frac{\det(\mathbf{a}_{k+1}, \mathbf{a}_{k+2})}{\det(\mathbf{a}_k, \mathbf{a}_{k+2})}, \quad p_{k,k+1} = \frac{\det(\mathbf{a}_k, \mathbf{a}_{k+1})}{\det(\mathbf{a}_k, \mathbf{a}_{k+2})},$$

$$p_{i,k+1} = 0 \ \text{ при } \ i \in J'_{N-3} \setminus \{k-1, k\}.$$

Кроме того, положим $\tilde{\omega} \stackrel{\text{def}}{=} (\tilde{\omega}_{-1}, \tilde{\omega}_0, \ldots, \tilde{\omega}_{N-3})^T$, $\omega \stackrel{\text{def}}{=} (\omega_{-1}, \omega_0, \ldots, \omega_{N-1})^T$.

Теорема 2. *Пусть выполнено условие (A). Тогда справедливо соотношение* $\tilde{\omega} = P_{N\{\Gamma\}}\omega$.

Матрица продолжения

Рассмотрим систему функционалов $\{\tilde{g}_i\}_{i \in J'_{\tilde{N}-1}}$, биортогональную к системе $\{\tilde{\omega}_j\}_{j \in J'_{\tilde{N}-1}}$, $\langle g_i, \tilde{\omega}_j \rangle = \delta_{i,j}$, со свойством

$$\operatorname{supp} \tilde{g}_i \subset [\tilde{x}_i, \tilde{x}_i + \varepsilon), \quad \varepsilon > 0, \quad i \in J_{\tilde{N}-1}, \quad \operatorname{supp} \tilde{g}_{-1} \subset (\tilde{x}_0, \tilde{x}_0 + \varepsilon).$$

Рассмотрим матрицу $Q_{N\{\Gamma\}}$ с элементами $q_{i,j} \stackrel{\text{def}}{=} \langle \tilde{g}_i, \omega_j \rangle$; здесь $i \in J'_{\tilde{N}-1}$, $j \in J'_{N-1}$. Матрица $Q_{N\{\Gamma\}}$ называется *матрицей продолжения*. Вычисле-

ние элементов этой матрицы проводится с использованием соотношения $\langle \tilde{g}_i, \varphi \rangle = \tilde{\mathbf{a}}_i$. В результате получаем

$$q_{i,j} = \delta_{i,j} \text{ при } i \le k-2, \quad q_{i,j} = \delta_{i,j+2} \text{ при } k+1 \le i, j \in J'_{N-1},$$

$$q_{k-1,k-2} = \frac{\det(\mathbf{a}_k, \mathbf{a}_{k-1})}{\det(\mathbf{a}_{k-2}, \mathbf{a}_{k-1})}, \quad q_{k-1,k-1} = \frac{\det(\mathbf{a}_{k-2}, \mathbf{a}_k)}{\det(a_{k-2}, \mathbf{a}_{k-1})},$$

$$q_{k-1,j} = 0 \text{ при } j \in \in J'_{N-1} \setminus \{k-2, k-1\},$$

$$q_{k,k} = \frac{\det(\mathbf{a}_{k+2}, \mathbf{a}_{k+1})}{\det(\mathbf{a}_k, \mathbf{a}_{k+1})}, \quad q_{k,k+1} = \frac{\det(\mathbf{a}_k, \mathbf{a}_{k+2})}{\det(\mathbf{a}_k, \mathbf{a}_{k+1})},$$

$$q_{k,j} = 0 \text{ при } j \in \in J'_{N-1} \setminus \{k, k+1\}.$$

Теорема 3. *Справедливо соотношение* $Q_{N\{\Gamma\}} P^T_{N\{\Gamma\}} = I$, *где* I *— единичная квадратная матрица порядка* $N-1$.

Интерференция в вэйвлетном потоке

Теорема 4. *Коммутатор* $V_{N\{\Gamma\}}$ *матриц* $Q_{N\{\Gamma\}}$ *и* $P^T_{N\{\Gamma\}}$ *представляет собой квадратную матрицу с элементами* $v_{i,j}$, $i, j \in J'_{N-1}$; *все ее элементы равны нулю, кроме перечисленных ниже семи элементов:*

$$v_{k+1,k-2} = -\frac{\det(\mathbf{a}_{k+1}, \mathbf{a}_{k+2})}{\det(\mathbf{a}_k, \mathbf{a}_{k+2})} \cdot \frac{\det(\mathbf{a}_k, \mathbf{a}_{k-1})}{\det(\mathbf{a}_{k-2}, \mathbf{a}_{k-1})}, \quad v_{k,k} = 1, \quad v_{k+2,k+2} = 1,$$

$$v_{k+1,k-1} = -\frac{\det(\mathbf{a}_{k+1}, \mathbf{a}_{k+2})}{\det(\mathbf{a}_k, \mathbf{a}_{k+2})} \cdot \frac{\det(\mathbf{a}_{k-2}, \mathbf{a}_k)}{\det(\mathbf{a}_{k-2}, \mathbf{a}_{k-1})}, \quad v_{k+1,k} = -\frac{\det(\mathbf{a}_{k+2}, \mathbf{a}_{k+1})}{\det(\mathbf{a}_k, \mathbf{a}_{k+2})},$$

$$v_{k,k-2} = -\frac{\det(\mathbf{a}_k, \mathbf{a}_{k-1})}{\det(\mathbf{a}_{k-2}, \mathbf{a}_{k-1})}, \quad v_{k,k-1} = -\frac{\det(\mathbf{a}_{k-2}, \mathbf{a}_k)}{\det(\mathbf{a}_{k-2}, \mathbf{a}_{k-1})},$$

$$v_{k+2,k} = -\frac{\det(\mathbf{a}_{k+2}, \mathbf{a}_{k+1})}{\det(\mathbf{a}_k, \mathbf{a}_{k+1})}, \quad v_{k+2,k+1} = -\frac{\det(\mathbf{a}_k, \mathbf{a}_{k+2})}{\det(\mathbf{a}_k, \mathbf{a}_{k+1})}.$$

Следствие. *Вэйвлетная составляющие* $b = (I - P^T_{N\{\Gamma\}} Q_{N\{\Gamma\}})c$ *имеет вид*

$$b_j = 0 \text{ при } j \in J'_{N-1} \setminus \{k-1, k, k+1\},$$

$$b_k = v_{k,k-2} c_{k-2} + v_{k,k-1} c_{k-1} + c_k, \tag{2}$$

$$b_{k+1} = v_{k+1,k-2} c_{k-2} + v_{k+1,k-1} c_{k-1} + v_{k+1,k} c_k, \tag{3}$$

$$b_{k+2} = v_{k+2,k} c_k + v_{k+2,k+1} c_{k+1} + c_{k+2}. \tag{4}$$

Справедливо соотношение

$$b_{k+1} - p_{k-1,k+1} b_k = 0. \tag{5}$$

Соотношение (5) справедливо для любого исходного потока; оно означает, что коэффициент $p_{k-1,k+1}$ позволяет всегда восстановить значение b_{k+1} по значению b_k (так что значение b_{k+1} передавать не нужно).

Свойство вида (5) будем называть *результатом интерференции в вэйвлетном потоке* (или просто *интерференцией вэйвлетов*).

В случае, когда имеется две вэйвлетные компоненты, отличающиеся друг от друга множителем, не зависящим от исходного потока $\{c_j\}$, будем говорить, что *интерференция в вэйвлетном потоке порождает стоячую волну второго порядка*.

Об аппроксимационных свойствах вэйвлетов

Предположим, что $\varphi \in C[a,b]$. Для непрерывности рассматриваемых сплайнов необходимо и достаточно, чтобы $\mathbf{a}_j = \varphi_{j+1}$; здесь $\varphi_j \stackrel{\text{def}}{=} \varphi(x_j)$.

Пусть вектор-функция $\varphi(t)$ имеет вид $\varphi(t) = (1, f(t))^T$, где $f \in C[a,b]$.

Обозначая $f_i \stackrel{\text{def}}{=} f(x_i)$, имеем $\det(\mathbf{a}_i, \mathbf{a}_j) = f_{j+1} - f_{i+1}$. Таким образом, в случае непрерывных вэйвлетов из (2) получаем

$$b_k = f_{k+1} - f_k f_k - f_{k-1} \cdot c_{k-2} - f_{k+1} - f_{k-1} f_k - f_{k-1} \cdot c_{k-1} + c_k, \tag{6}$$

а из (3) с использованием (5) выводим

$$b_{k+1} = f_{k+3} - f_{k+2} f_{k+3} - f_{k+1} \cdot b_k; \tag{7}$$

наконец, из (4) находим

$$b_{k+2} = f_{k+3} - f_{k+2} f_{k+2} - f_{k+1} \cdot c_k - f_{k+3} - f_{k+1} f_{k+2} - f_{k+1} \cdot c_{k+1} + c_{k+2}, \tag{8}$$

Если $\varphi(t) = (1,t)^T$ имеем $f(t) = t$, $f_i = x_i$, и формулы (6) — (8) принимают вид

$$b_k = x_{k+1} - x_k x_k - x_{k-1} \cdot c_{k-2} - x_{k+1} - x_{k-1} x_k - x_{k-1} \cdot c_{k-1} + c_k, \tag{9}$$

$$b_{k+1} = x_{k+3} - x_{k+2} x_{k+3} - x_{k+1} \cdot b_k \tag{10}$$

$$b_{k+2} = x_{k+3} - x_{k+2} x_{k+2} - x_{k+1} \cdot c_k - x_{k+3} - x_{k+1} x_{k+2} - x_{k+1} \cdot c_{k+1} + c_{k+2}, \tag{11}$$

Рассматривая здесь равномерную сетку $x_j = jh$, $h > 0$, находим

$$b_k = c_{k-2} - 2c_{k-1} + c_k, \tag{12}$$

$$b_{k+1} = 12c_{k-2} - c_{k-1} + 12c_k, \tag{13}$$

$$b_{k+2} = c_k - 2c_{k+1} + c_{k+2}. \tag{14}$$

Замечание. Здесь видно, что для использования вэйвлетных составляющих достаточно принять во внимание компоненты b_k и b_{k+2}.

Предполагая, что источником исходного потока $\{c_j\}_{j\in\mathbb{Z}}$ является функция $u \in C^2$, а именно $c_j = u(jh)$, из (11) — (14) имеем

$$b_k = u''(\xi)h^2, \quad b_{k+1} = 12u''(\xi)h^2, \quad b_{k+2} = u''(\zeta)h^2,$$

где ξ, ζ — некоторые точки интервала (x_{k-2}, x_{k+2}).

Заключение

В работе рассмотрен эффект интерференции и образования стоячих волн в вэйвлетном потоке, позволяющий при передаче вэйвлетных компонент по каналам связи уменьшить объем передачи на одну треть. Установлено, что для исходного потока, порожденного гладкой функцией, вэйвлетный поток имеет порядок второго дифференцила упомянутой функции.

Литература

1. *Демьянович Ю. К., Мирошниченко И. Д.* Гнездовые сплайн-вэйвлетные разложения//Проблемы математического анализа. 2012. Вып.64. С.51–61.

ГНЕЗДОВЫЕ УКРУПНЕНИЯ И ВЛОЖЕННОСТЬ СПЛАЙНОВЫХ ПРОСТРАНСТВ[1]

И. Д. Мирошниченко

ст. преп. кафедры параллельных алгоритмов;
irinamir @mail.ru

Санкт-Петербургский государственный университет

Аннотация: Рассматриваются условия вложенности (вообще говоря, разрывных и неполиномиальных) сплайновых пространств, получающихся при удалении группы узлов (гнезда), устанавливаются калибровочные соотношения и даются рекомендации в случае удаления нескольких гнезд. Результаты могут быть применены для сплайн-вэйвлетного разложения рассмотренных пространств.

1. Введение

В предыдущей работе авторов (см. [1]) рассматривались негладкие сплайн-вэйвлетные разложения первого порядка, получаемые последовательным удалением узлов. Это позволяло упростить (по сравнению с гладкими разложениями) формулы для вычисления упомянутых разложений с сохранением свойств адаптивности (приспособляемости) к обрабатываемому потоку. Там же была установлена независимость вэйвлетного разложения от порядка удаления узлов. Однако, при численной реализации в машинной арифметике с плавающей точкой последовательное удаление большого количества узлов приводит к быстрому нарастанию ошибок округления, и потому такой подход оправдан лишь при вычислениях в реальном масштабе времени, когда запаздывание не допустимо.

В данной работе рассматривается ситуация, когда возможно запаздывание при обработке поступающего числового потока; в этом случае актуально одновременное удаление групп последовательных узлов (называемых здесь гнездами). В предлагаемой статье рассматриваются условия вложенности (вообще говоря, разрывных и неполиномиальных) сплайновых пространств, получающихся при удалении группы узлов (гнезд), и даются калибровочные соотношения для базисных функций рассматриваемых пространств. Изложение ведется для одного гнезда, но в заключении указано, как распространить полученные результаты на множество гнезд; последнее важно для параллельной обработки числовых потоков.

2. Предварительные обозначения

Пусть m и N — натуральные числа. Рассмотрим сетку

$$X_N : a = x_0 < x_1 < \ldots < x_{N-2} < x_{N-1} < x_N = b,$$

[1] Работа частично поддержана грантами РФФИ 10-01-00297 и 10-01-00245.

и обозначим

$$J_m \stackrel{\text{def}}{=} \{0, 1, \ldots, m\},\ J'_m \stackrel{\text{def}}{=} \{-1, 0, 1, \ldots, m\},$$

$$S_j \stackrel{\text{def}}{=} (x_j, x_{j+1}) \cup (x_{j+1}, x_{j+2}),\ j \in J'_{N-1},$$

$$G \stackrel{\text{def}}{=} \bigcup_{i=0}^{N-1} (x_i, x_{i+1}),\ a \stackrel{\text{def}}{=} x_0,\ b \stackrel{\text{def}}{=} x_N.$$

Система векторов $\{\mathbf{a}_i\}_{i \in J'_m}$ пространства $\mathbb{R}^2$ называется полной цепочкой векторов, если $\det(\mathbf{a}_{j-1}, \mathbf{a}_j) \neq 0$ при $j \in J_m$.

3. Построение пространства сплайнов

Пусть $A_N \stackrel{\text{def}}{=} \{\mathbf{a}_i\}_{i \in J'_{N-1}}$ — полная цепочка двумерных векторов. По двухкомпонентной вектор-функции $\varphi(t)$, $t \in G$, с линейно независимыми компонентами на любом интервале $(a', b') \subset G$, определим функции $\omega_j(t)$, $t \in G_N$, $j \in J'_{N-1}$, с помощью аппроксимационных соотношений

$$\sum_{j \in J'_{N-1}} \mathbf{a}_j \omega_j(t) = \varphi(t) \quad \forall t \in G,$$

$$\omega_j(t) \equiv 0 \quad \forall t \in G \setminus S_j,\ j \in J'_{N-1}.$$

Рассмотрим пространство $\mathbb{S}_N \stackrel{\text{def}}{=} Cl_p\, \mathcal{L}\{\omega_j\}_{j \in J'_{N-1}}$, где $\mathcal{L}\{...\}$ — линейная оболочка функций, указанных в фигурных скобках, а Cl_p — замыкание в топологии поточечной сходимости. Ввиду предположений относительно $\varphi(t)$ полученные функции $\omega_j(t)$, $j \in J'_{N-1}$, линейно независимы на множестве G и, следовательно, являются базисом пространства $\mathbb{S}_N$, и $\dim \mathbb{S}_N = N + 1$.

4. Удаление гнезда

Пусть s, k, N — целые числа, удовлетворяющие условию $0 \le s + 1 < k \le N - 2$. Рассмотрим сетку

$$X_{N\{\Gamma\}}: \quad x_0^* < x_1^* < \ldots < x_{N-s-3}^* < x_{N-s-2}^*,$$

где

$$\Gamma = \Gamma(k, s) \stackrel{\text{def}}{=} \{x_{k-s}, x_{k-s+1}, \ldots, x_k, x_{k+1}\},$$

$$x_j^* \stackrel{\text{def}}{=} x_j \ \text{ при } \ j \le k - s - 1,$$

$$x_j^* \stackrel{\text{def}}{=} x_{j+s+2} \ \text{ при } \ j \ge k - s,\ j \in J_{N-s-2}.$$

Множество Γ удаляемых узлов называется *гнездом*.

Положим

$$S_j^* \stackrel{\text{def}}{=} (x_j^*, x_{j+1}^*) \cup (x_{j+1}^*, x_{j+2}^*), \;\; j \in J'_{N-s-4},$$

$$G^* \stackrel{\text{def}}{=} \bigcup_{j \in J_{N-s-3}} (x_j^*, x_{j+1}^*).$$

Пусть задана полная цепочка двумерных векторов $\{\mathbf{a}_j^*\}_{j \in J'_{N-s-3}}$. С использованием этой цепочки построим систему функций $\{\omega_j^*\}_{j \in J'_{N-s-3}}$, отыскивая ее из соотношений

$$\sum_{j \in J'_{N-s-3}} \mathbf{a}_j^* \, \omega_j^*(t) = \varphi(t) \quad \forall t \in G^*,$$

$$\omega_j^*(t) \equiv 0 \quad \forall t \in G^* \setminus S_j^*, \;\; j \in J'_{N-s-3}.$$

и введем пространство $\mathbb{S}_{N\{\Gamma\}} \stackrel{\text{def}}{=} Cl_p \, \mathcal{L}\{\omega_j^*\}_{j \in J'_{N-s-3}}$. Очевидно, что $\dim \mathbb{S}_{N\{\Gamma\}} = N - s - 1$.

5. Калибровочные соотношения

Пусть выполнено условие

(B) при $\forall t \in G, \;\; j \in J'_{N-s-3}$ справедливы соотношения

$$\mathbf{a}_j^* \omega_j^*(t) = \mathbf{a}_j \omega_j(t) \;\text{ при }\; j \le k - s - 3,$$

$$\mathbf{a}_j^* \omega_j^*(t) = \mathbf{a}_{j+s+2} \omega_{j+s+2}(t) \;\text{ при }\; j \ge k - s.$$

Теорема 1. *При условии (B) выполняются калибровочные соотношения*

$$\omega_{k-s-2}^*(t) \equiv \sum_{j=k-s-2}^{k+1} \mathfrak{p}_{k-s-2,\,j} \; \omega_j(t),$$

$$\omega_{k-s-1}^*(t) \equiv \sum_{j=k-s-2}^{k+1} \mathfrak{p}_{k-s-1,\,j} \; \omega_j(t),$$

где

$$\mathfrak{p}_{k-s-2,\,j} \stackrel{\text{def}}{=} \det(\mathbf{a}_j, \mathbf{a}_{k-s-1}^*) \det(\mathbf{a}_{k-s-2}^*, \mathbf{a}_{k-s-1}^*),$$

$$\mathfrak{p}_{k-s-1,\,j} \stackrel{\text{def}}{=} \det(\mathbf{a}_{k-s-2}^*, \mathbf{a}_j) \det(\mathbf{a}_{k-s-2}^*, \mathbf{a}_{k-s-1}^*), \;\; j = k-s-2, \ldots, k+1.$$

Следствие. *Верно включение*

$$\mathbb{S}_{N\{\Gamma\}} \subset \mathbb{S}_N. \tag{1}$$

6. Заключение

Пусть натуральные числа k_i, $i = 0, 1, 2, ..., r$, удовлетворяют соотношениям

$$k_0 = 0 < k_1 < \ldots < k_r < N - 2,\ s_l + 2 < k_l - k_{l-1},\ l = 1, 2, \ldots, r,$$

где $s_l \geq -1$ — некоторые целые числа.

Рассмотрим гнезда (т. е. множества узлов, удаляемых из основной сетки X)

$$\Gamma_q \stackrel{\text{def}}{=} \{x_{k_q - s_q}, x_{k_q - s_q + 1}, \ldots, x_{k_q + 1}\},\quad q = 1, 2, \ldots, r.$$

В сформулированных ранее условиях справедливо включение (1), где следует заменить гнездо Γ на совокупность гнезд Γ_1, Γ_2, ..., Γ_r.

В заключение заметим, что полученные результаты могут быть применены для вэйвлетного разложения рассмотренных здесь сплайновых пространств.

Литература

1. *Ю. К. Демьянович, И. Д. Мирошниченко.* Негладкие сплайн-вэйвлетные разложения на отрезке // Проблемы математического анализа. 2012. Вып. 63. С. 23–40.

КОМПЬЮТЕРНАЯ ОБРАБОТКА ДВУМЕРНЫХ ЧИСЛОВЫХ ПОТОКОВ С ПОМОЩЬЮ ВЭЙВЛЕТОВ

Ю. К. Демьянович

зав. кафедрой параллельных алгоритмов;
Yuri.Demjanovich@gmail.com

А. С. Виласак

студент 543 группы математико-механического факультета,
кафедра параллельных алгоритмов;
shaurma²@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Рассматривается общая структура вэйвлетного разложения числового потока, построенного с использованием вложенных сплайновых пространств и специального проектирования. Дана характеристика особенностей компьютерной обработки сплайн-вэйвлетного разложения двумерного потока числовой информации.

Введение

В современном мире технологического прогресса и инновационных технологий огромное значение придается информации, ее качеству, а также скорости ее передачи. И если в некоторых случаях высокое качество может быть не основным критерием, то скорость передачи всегда существенна. В частности, при передаче изображений в первую очередь важно знать, что передается, и лишь затем в случае необходимости уточнить переданное изображение. Для обработки числовых информационных потоков используется вэйвлетное разложение на основной и уточняющий (вэйвлетный) потоки (см. [1–2]).

В данной работе рассматриваются вэйвлетные разложения, выводятся формулы декомпозиции/реконструкции числовых потоков и дается характеристика особенностей компьютерной обработки сплайн-вэйвлетного разложения двумерного потока числовой информации.

Общая структура вэйвлетного разложения

Рассмотрим пространство непрерывных функций $\underline{C}(\Omega)$, заданных в двумерной области Ω, а в нем сплайновые пространства $\tilde{S}$ и S такие, что

$$\tilde{S} \subset S \subset C(\Omega). \qquad (1)$$

Пусть сплайновые базисы этих пространств $\{\omega_\alpha\}$ и $\{\omega_\beta\}$ соответственно удовлетворяют калибровочному соотношению

$$\tilde{\omega}_\alpha = \sum_\beta p_{\alpha,\beta}\omega_\beta \tag{2}$$

с числовыми коэффициентами $p_{\alpha,\beta}$. Обозначим $\{\tilde{g}_\alpha\}$ распространение на $C(\Omega)$ системы линейных функционалов, биортогональной к $\{\tilde{\omega}_\alpha\}$.

Рассмотрим оператор $\tilde{P}$ проектирования пространства $C(\Omega)$ на подпространство $\tilde{S}$, задаваемое формулой

$$\tilde{P}u = \sum_\alpha \langle \tilde{g}_\alpha, u\rangle\, \omega_\alpha \quad \forall u \in C(\Omega),$$

и введем оператор $Q_0 \overset{\text{def}}{=} I - \tilde{P}$; здесь I — тождественный в $C(\Omega)$ оператор.

Пространством вейвлетов называется пространство $W_0 \overset{\text{def}}{=} Q_0 S$. В результате получаем прямое разложение $S = \tilde{S} \dotplus W$, которое называется *сплайн-вейвлетным разложением* пространства S.

Пусть $u \in S$. Приравнивая правые части двух представлений этого элемента,

$$u = \sum_\gamma c_\gamma \omega_\gamma, \quad u = \sum_\alpha a_\alpha \tilde{\omega}_\alpha + \sum_\alpha b_\beta \omega_\beta, \quad a_\alpha \overset{\text{def}}{=} \langle \tilde{g}_\alpha, u\rangle,$$

и используя калибровочные соотношения (1), получим *формулы реконструкции*

$$c_\gamma = \sum_{\alpha'} a_{\alpha'} p_{\alpha',\gamma} + b_\gamma, \tag{3}$$

а также *формулы декомпозиции*

$$a_\alpha = \sum_\gamma q_{\alpha,\gamma} c_\gamma, \quad b_\gamma = c_\gamma - \sum_\beta c_\beta \sum_{\alpha'} q_{\alpha',\beta} p_{\alpha',\gamma}; \tag{4}$$

здесь

$$q_{\alpha',\beta} \overset{\text{def}}{=} \langle \tilde{g}_{\alpha'}, \omega_\beta\rangle.$$

О реализации вэйвлетного разложения для двумерного числового потока

Рассмотрим топологически правильные триангуляции $\tilde{\mathcal{T}}$ и $\mathcal{T}$, содержащиеся в области Ω и такие, что множество вершин первой из них содержится во множестве вершин второй. На этих триангуляциях построим пространства сплайновых аппроксимаций с использованием базисных функций Р. Куранта, и примем их за пространства $\tilde{S}$ и S соответственно. Очевидно, что для этих пространств справедливы соотношения (1) и (2); в частном случае двукратного измельчения триангуляции $\tilde{\mathcal{T}}$ коэффициенты $p_{\alpha,\beta}$ в калибровочном соотношении (2) найдены в работе [2]. Определим функционалы g_α формулой

$$\langle \tilde{g}_\alpha, u \rangle = u(\tilde{M}_\alpha),$$

где $\tilde{M}_\alpha$ — вершины триангуляции $\tilde{\mathcal{T}}$. Легко видеть, что так определенная система функционалов биортогональна функциям Р. Куранта $\tilde{\omega}_\beta$, построенным на барицентрической звезде с центром $\tilde{M}_\beta$.

Компьютерная реализации сплайн-вэйвлетного разложения базируется на формулах (3) и (4), структура которых в рассматриваемом случае оказывается весьма простой: число слагаемых в них не более удвоенного числа треугольников упомянутых барицентрических звезд. Коэффициенты этих формул не зависят от диаметров рассматриваемых треугольников и определяются лишь отношением соответствующих сторон триангуляций $\tilde{\mathcal{T}}$ и $\mathcal{T}$; таким образом, устойчивость вычислений не зависит от мелкости триангуляций и определяется лишь упомянутым соотношением. Отметим также независимость вычислительного процесса по формулам (3) и (4) при различных γ и α, что позволяет эффективно распараллелить вычислительный процесс, передавая вычисления по каждой из упомянутых формул одному параллельному вычислительному модулю.

Заключение

В данной работе рассматривается общая структура вэйвлетного разложения числового потока, построенного с использованием вложенных сплайновых пространств и специального проектирования. Дана характеристика особенностей компьютерной реализации указанного разложения двумерного потока числовой информации. Установлено, что устойчивость вычислений не зависит от мелкости триангуляций и определяется лишь упомянутым соотношением соответствующих сторон треугольников крупной и мелкой триангуляций. Вычисления по предложенным формулам декомпозиции и реконструкции не зависят друг от друга при различных значениях фигурирующих в них индексов; это позволяет эффективно распараллелить вычислительный процесс, передавая вычисления по упомянутым формулам параллельным вычислительным модулям.

Литература

1. *Демьянович Ю. К., Ходаковский В. А.* Введение в вейвлеты. 2007.
2. *Арсентьева Е. П., Демьянович Ю. К.* Адаптивные сплайн-вейвлетные разложения двумерных потоков числовой информации // Сб. Проблемы математического анализа. 2011. Вып. 56. С. 3–17.

АЛГОРИТМ АДАПТИВНОГО УКРУПНЕНИЯ ТРИАНГУЛЯЦИИ

С. А. Калашян

студент 543 группы кафедры параллельных алгоритмов;
sedrak@kalashyan.com

Санкт-Петербургский государственный университет

Аннотация: Триангуляция области широко применяется для аппроксимации функций двух переменных. Для эффективного вэйвлетного разложения исходной функции необходимо ее локальное укрупнение, однако не всякую триангуляцию можно локально укрупнить. В данной работе рассматривается триангуляция плоскости, которая допускает локальное укрупнение с сохранением правильности, что позволяет описать алгоритм адаптивного укрупнения триангуляции.

Введение

Наиболее эффективные аппроксимации двумерных потоков числовой информации базируются на триангуляции, правильной в топологическом смысле; однако, построение адаптивных вэйвлетов наталкивается на трудности: не всякая триангуляция допускает локальное укрупнение с сохранением правильности триангуляции; например, триангуляция, часто используемая в методе конечных элементов, не допускает упомянутого укрупнения. В моей работе рассматривается триангуляция плоскости, которая допускает локальное укрупнение с сохранением правильности, что позволит описать алгоритм адаптивного укрупнения триангуляции.

Данная проблема рассматривалась в частности в работе Арсентьевой Е. П.: Адаптивные сплайн-вэйвлетные разложения двумерных потоков числовой информации.

Описание алгоритма

В дальнейшем рассматривается плоскость $\{t \mid t = (x, y) \in R^2\}$, правильно подразделенная на треугольники. Правильность подразделения означает, что вершина треугольника не может лежать внутри стороны другого треугольника. Соответственно получаем правильную триангуляцию.

Для описания триангуляции достаточно указать таблицу инциденций, каждая строка которой описывает треугольник перечнем инцидентных ему вершин. Задача состоит в том, чтобы описать локальное укрупнение одного из вариантов исходной триангуляции, которое может быть полезно для сжатия потоков информации, естественным образом связываемых с (конечной

или бесконечной) областью на плоскости. Поэтому рассматриваемая триангуляция может состоять из конечного или бесконечного числа треугольников.

Введем обозначения

$$\mathbb{Z}^2 \stackrel{\text{def}}{=} \{(i,\ j) \mid i \in \mathbb{Z} \forall\ j \in \mathbb{Z}\},$$

$$\mathbb{Z}_0^2 \stackrel{\text{def}}{=} \{(2i,\ 2j) \mid i \in \mathbb{Z} \forall\ j \in \mathbb{Z}\},$$

$$\mathbb{Z}_1^2 \stackrel{\text{def}}{=} \{(2i+1, 2j+1) \mid i \in \mathbb{Z} \forall\ j \in \mathbb{Z}\},$$

$$\tilde{\mathbb{Z}}^2 \stackrel{\text{def}}{=} \mathbb{Z}_0^2 \cup \mathbb{Z}_1^2.$$

Пусть фиксированы числа $h' > 0$ и $h'' > 0$. Обозначим через $M_{i,j}$ точки с координатами (ih', jh''), $(i,\ j) \in \mathbb{Z}^2$, и рассмотрим прямоугольники вида

$$\Pi_{i,\ j} \stackrel{\text{def}}{=} \{(x,\ y) \mid ih' \le x \le (i+2)h',\ \ jh'' \le x \le (j+2)h''\},\ (i,\ j) \in \mathbb{Z}_0^2.$$

Пример триангуляции можно описать трех-столбцовой таблицей с бесконечным числом строк, получающейся объединением таблиц

$$\left\| \begin{matrix} M_{i,\ j} & M_{i+1,\ j} & M_{i,\ j+1} \\ M_{i+1,\ j+1} & M_{i+1,\ j} & M_{i,\ j+1} \end{matrix} \right\|$$

при $(i,\ z) \in \mathbb{Z}^2$. Здесь строка $M_{i,j}\ M_{i+1,j}\ M_{i,j+1}$ означает, что рассматривается треугольник с вершинами $M_{i,j}$, $M_{i+1,j}$, $M_{i,j+1}$, а строка $M_{i+1,j+1}\ M_{i+1,j}\ M_{i,j+1}$ означает, что рассматривается треугольник у которого вершинами служат точки $M_{i+1,j+1}$, $M_{i+1,j}$, $M_{i,j+1}$. Заметим, что порядок объединения таблиц не существен; не существен также порядок строк и порядок следования вершин в строках рассматриваемых таблиц. Получаемая в результате триангуляция не допускает локального укрупнения с сохранением правильности.

Пусть $X^* \in \mathbb{Z}_0^2$ — некоторое (конечное или бесконечное) множество пар четных целочисленных индексов. Рассмотрим замкнутую область

$$\Omega \stackrel{\text{def}}{=} \bigcup_{(i,\ j) \in X^*} \Pi_{i,\ j}$$

(в частности, Ω совпадает со всей плоскостью $\mathbb{R}_2$, если $X^* = \mathbb{Z}_0^2$).

Через X обозначим множество индексов $(i,\ j) \in \mathbb{Z}_0^2$ и $\mathbb{Z}_1^2$ таких, что точки $M_{i,j} = (ih', jh'')$ лежат в Ω. Эти точки $M_{i,j}$ будем называть узлами исходной сетки, они являются вершинами определяемой ниже исходной триангуляции.

Узел $M_{2i_0,\ 2j_0}$ называется внутренним узлом для Ω, если он является внутренней точкой в Ω (таким образом, в Ω содержатся прямоугольники $\Pi_{i,\ j}$ при $i \in \{2i_0,\ 2i_0 - 2\}$, $j \in \{2j_0,\ 2j_0 - 2\}$; здесь же заметим, что вводимое понятие относится только к узлам с четными индексами). Множество пар $(i,\ j) \in X^*$, для которых $M_{i,j}$ — внутренний узел, обозначим через X_0. Очевидно, что $X_0 \subset X^* \subset X \subset \tilde{\mathbb{Z}}^2$.

Заметим, что рассматриваемую далее триангуляцию можно привести для всей плоскости; область Ω существенна при рассмотрении калибровочных соотношений и формул декомпозиции и реконструкции.

В отличие от только что упомянутой триангуляции рассмотрим триангуляцию, которая получается объединением таблиц

$$\begin{Vmatrix} M_{i,j} & M_{2+i,j} & M_{1+i,1+j} \\ M_{2+i,2+j} & M_{2+i,j} & M_{1+i,1+j} \\ M_{2+i,2+j} & M_{i,2+j} & M_{1+i,1+j} \\ M_{i,j} & M_{i,2+j} & M_{1+i,1+j} \end{Vmatrix} \qquad \forall (i,\ j) \in \mathbb{Z}_0^2.$$

Укрупнение триангуляции будем производить объединением двух соседних (т. е. имеющих общую сторону) треугольников. Полученные в результате треугольники будем называть *укрупненными* треугольниками.

Не нарушая общности, в дальнейшем предполагаем, что $M_{0,0}$ внутренний узел в Ω, т. е. $(0,\ 0) \in X_0$. Рассмотрим такое укрупнение, при котором вершину $M_{0,0}$ будут окружать лишь укрупненные треугольники. Для этого заменим перечисленные ниже соседние треугольники на треугольник, получающийся их объединением. Эквивалентное преобразование таблицы инциденций состоит в том, что из нее исключаются строки, соответствующие объединяемым треугольникам и добавляются строки, соответствующие результатам такого объединения — укрупненным треугольникам. Как было отмечено выше, расположение строк в таблице инциденций не существенно, и потому строки могут быть добавлены между любыми строками упомянутой таблицы. Таким образом, достаточно перечислить выбрасываемые строки таблицы и указать вставляемые в нее строки. Однако, для наглядности преобразования таблицы инциденций будем задавать указанием двух строк заменяемых треугольников (в левой от стрелки части формулы) и указанием строки укрупненного треугольника (в правой части формулы).

Итак, укрупнение зададим следующим преобразованием таблицы инциденций

$$\begin{Vmatrix} M_{0,0} & M_{-2,0} & M_{-1,1} \\ M_{-2,2} & M_{-2,0} & M_{-1,1} \end{Vmatrix} \longrightarrow \| M_{0,0} \quad M_{-2,0} \quad M_{-2,2} \|, \tag{1.2}$$

$$\begin{Vmatrix} M_{-2,2} & M_{0,2} & M_{-1,1} \\ M_{0,0} & M_{0,2} & M_{-1,1} \end{Vmatrix} \longrightarrow \| M_{0,0} \quad M_{0,2} \quad M_{-2,2} \|, \tag{1.3}$$

$$\begin{Vmatrix} M_{0,0} & M_{0,2} & M_{1,1} \\ M_{2,2} & M_{0,2} & M_{1,1} \end{Vmatrix} \longrightarrow \| M_{0,0} \quad M_{0,2} \quad M_{2,2} \|, \tag{1.4}$$

$$\begin{Vmatrix} M_{2,2} & M_{2,0} & M_{1,1} \\ M_{0,0} & M_{2,0} & M_{1,1} \end{Vmatrix} \longrightarrow \| M_{0,0} \quad M_{2,0} \quad M_{2,2} \|, \tag{1.5}$$

$$\begin{Vmatrix} M_{0,0} & M_{2,0} & M_{1,-1} \\ M_{2,-2} & M_{2,0} & M_{1,-1} \end{Vmatrix} \longrightarrow \| M_{0,0} \quad M_{2,0} \quad M_{2,-2} \|, \tag{1.6}$$

$$\begin{Vmatrix} M_{2,-2} & M_{0,-2} & M_{1,-1} \\ M_{0,0} & M_{0,-2} & M_{1,-1} \end{Vmatrix} \longrightarrow \| M_{0,0} \quad M_{0,-2} \quad M_{2,-2} \|, \tag{1.7}$$

$$\begin{Vmatrix} M_{0,0} & M_{0,-2} & M_{-1,-1} \\ M_{-2,-2} & M_{0,-2} & M_{-1,-1} \end{Vmatrix} \longrightarrow \| M_{0,0} \quad M_{-2,0} \quad M_{-2,-2} \|, \tag{1.8}$$

$$\begin{Vmatrix} M_{-2,-2} & M_{-2,0} & M_{-1,-1} \\ M_{0,0} & M_{-2,0} & M_{-1,-1} \end{Vmatrix} \longrightarrow \| M_{0,0} \quad M_{-2,0} \quad M_{-2,-2} \|. \tag{1.9}$$

Заключение

Результатом исследования является алгоритм построения правильной триангуляции, допускающей локальное укрупнение. Актуальна проблема отыскания симплициального подразделения с аналогичными свойствами в трехмерном случае; эту проблему предполагается изучить в дальнейшем.

Литература

1. *Демьянович Ю. К., Ходаковский В. А.* Введение в вэйвлеты. 2007.
2. *Демьянович Ю. К.* Вэйвлеты к минимальные сплайны. 2003.
3. *Арсентьева Е. П., Демьянович Ю. К.* Адаптивные сплайн-вэйвлетные разложения двумерных потоков числовой информации // Сб. Проблемы математического анализа. 2011. Вып. 56. С. 3–17.

ЛОКАЛЬНОЕ УКРУПНЕНИЕ ТРИАНГУЛЯЦИИ И ДВУМЕРНЫЕ СПЛАЙН-ВЭЙВЛЕТЫ[1]

Ю. К. Демьянович

зав. кафедрой параллельных алгоритмов;
Yuri.Demjanovich@gmail.com

Л. М. Романовский

аспирант кафедры параллельных алгоритмов;
lromanovskey@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Рассмотрена триангуляция, которая допускает локальное укрупнение с сохранением правильности. Укрупненная триангуляция порождает курантовское пространство сплайнов, вложенное в подобное пространство, построенное для исходной триангуляции. Построено сплайн-вэйвлетное разложение объемлющего пространства, которое позволяет учесть локальные особенности двумерного потока данных.

1. Введение

Вэйвлетные аппроксимации применяются, главным образом, к одномерным потокам данных. Однако, при работе с многомерными потоками для построения вэйвлетной аппроксимации приходится из многомерного потока получать одномерный. Это приводит к потере информации о топологии исходного потока, и к неоптимальности его сплайн-вэйвлетной обработки. Преодолению этой трудности посвящена работа [1], в которой предлагается использовать вэйвлетные разложения курантова типа, основаные на триангуляции рассматриваемой области. Однако, для эффективных вэйвлетных разложений исходного потока нужны такие триангуляции, которые допускают локальное укрупнение; заметим, что широко используемые (например, в методе конечных элементов) триангуляции не допускают локального укрупнения.

В данной работе рассматривается триангуляция, допускающая локальное укрупнение с сохранением топологической правильности; с ее помощью строятся вложенные пространства курантовских аппроксимаций и дается их вэйвлетное разложение.

2. Предварительные сведения

Правильной триангуляцией называется триангуляция, в которой ни одна вершина не лежит внутри стороны другого треугольника. Рассматриваемая

[1] Работа частично поддержана грантами РФФИ 10-01-00297 и 10-01-00245.

в данной работе триангуляция позволяет производить локальное укрупнение с сохранением правильности, причем полученная в результате укрупнения триангуляция сохраняет прежнюю топологическую структуру и ее можно снова локально укрупнить.

3. Локальное укрупнение триангуляции

Введем обозначения

$$\mathbb{Z}^2 \stackrel{\text{def}}{=} \{(i,\, j) \,|\, i \in \mathbb{Z} \quad \forall j \in \mathbb{Z}\}, \quad \mathbb{Z}_{\mathbf{0}}^2 \stackrel{\text{def}}{=} \{(2i,\, 2j) \,|\, i \in \mathbb{Z} \quad \forall j \in \mathbb{Z}\},$$

$$\mathbb{Z}_{\mathbf{1}}^2 \stackrel{\text{def}}{=} \{(2i+1,\, 2j+1) \,|\, i \in \mathbb{Z} \quad \forall j \in \mathbb{Z}\}, \qquad \widehat{\mathbb{Z}}^2 \stackrel{\text{def}}{=} \mathbb{Z}_{\mathbf{0}}^2 \cup \mathbb{Z}_{\mathbf{1}}^2.$$

Пусть фиксированы числа $h' > 0,\, h'' > 0$. Обозначим $\mathbf{r}_{i,\,j}$ точки с координатами $(ih',\, jh'')$, $(i,\, j) \in \mathbb{Z}^2$, и рассмотрим прямоугольники вида $\Pi_{i,\,j} \stackrel{\text{def}}{=} \{(x,\, y) \,|\, ih' \le x \le (i+2)h',\ jh'' \le x \le (j+2)h''\}$, где $(i,\, j) \in \mathbb{Z}_{\mathbf{0}}^2$.

Обещанный во введении пример триангуляции можно описать трехстолбцовой таблицей (с бесконечным числом строк), получающейся объединением таблиц

$$\left\| \begin{matrix} \mathbf{r}_{i,\,j} & \mathbf{r}_{i+1,\,j} & \mathbf{r}_{i,\,j+1} \\ \mathbf{r}_{i+1,\,j+1} & \mathbf{r}_{i+1,\,j} & \mathbf{r}_{i,\,j+1} \end{matrix} \right\|$$

при $(i,\, j) \in \mathbb{Z}^2$; здесь строка $\mathbf{r}_{i,\,j}\ \mathbf{r}_{i+1,\,j}\ \mathbf{r}_{i,\,j+1}$ означает, что рассматривается треугольник с вершинами $\mathbf{r}_{i,\,j}, \mathbf{r}_{i+1,\,j}, \mathbf{r}_{i,\,j+1}$, а строка $\mathbf{r}_{i+1,\,j+1}\ \mathbf{r}_{i+1,\,j}\ \mathbf{r}_{i,\,j+1}$ означает, что рассматривается треугольник у которого вершинами служат точки $\mathbf{r}_{i+1,\,j+1}, \mathbf{r}_{i+1,\,j}, \mathbf{r}_{i,\,j+1}$. Получаемая в результате триангуляция не допускает локального укрупнения с сохранением правильности.

Пусть $\mathbb{X}^*$ — некоторое (конечное или бесконечное) множество пар четных целочисленных индексов: $\mathbb{X}^* \subset \mathbb{Z}_{\mathbf{0}}^2$; рассмотрим замкнутую область

$$\Omega \stackrel{\text{def}}{=} \bigcup_{(i,\,j) \in \mathbb{X}^*} \Pi_{i,\,j}.$$

(в частности, Ω совпадает со всей плоскостью $\mathbb{R}^2$, если $\mathbb{X}^* = \mathbb{Z}_{\mathbf{0}}^2$).

Через $\mathbb{X}$ обозначим множество индексов $(i,\, j) \in \mathbb{Z}_{\mathbf{0}}^2 \cup \mathbb{Z}_{\mathbf{1}}^2$ таких, что точки $\mathbf{r}_{i,\,j} = (ih',\, jh'')$ лежат в Ω; только что упомянутые точки $\mathbf{r}_{i,\,j}$ будем называть узлами исходной сетки, они являются вершинами определяемой ниже исходной триангуляции.

Узел $\mathbf{r}_{2i_0,\,2j_0}$ называется внутренним узлом для Ω, если он является внутренней точкой в Ω (таким образом, в Ω содержатся прямоугольники $\Pi_{i,\,j}$ при $i \in \{2i_0,\, 2i_0 - 2\}$, $j \in \{2j_0,\, 2j_0 - 2\}$; здесь же заметим, что вводимое понятие относится только к узлам с четными индексами). Множество пар $(i,\, j) \in \mathbb{X}^*$, для которых $\mathbf{r}_{i,\,j}$ — внутренний узел, обозначим $\mathbb{X}_0$. Очевидно, что

$$\mathbb{X}_0 \subset \mathbb{X}^* \subset \mathbb{X} \subset \hat{\mathbb{Z}}^2.$$

Рассмотрим триангуляцию, которая получается объединением таблиц

$$\left\| \begin{matrix} \mathbf{r}_{i,j} & \mathbf{r}_{2+i,j} & \mathbf{r}_{1+i,1+j} \\ \mathbf{r}_{2+i,2+j} & \mathbf{r}_{2+i,j} & \mathbf{r}_{1+i,1+j} \\ \mathbf{r}_{2+i,2+j} & \mathbf{r}_{i,2+j} & \mathbf{r}_{1+i,1+j} \\ \mathbf{r}_{i,j} & \mathbf{r}_{i,2+j} & \mathbf{r}_{1+i,1+j} \end{matrix} \right\| \qquad \forall (i,\, j) \in \mathbb{Z}_{\mathbf{0}}^2. \tag{1}$$

Укрупнение триангуляции будем производить объединением двух соседних (т. е. имеющих общую сторону) треугольников. Полученные в результате треугольники будем называть *укрупненными* треугольниками. Не нарушая общности, укрупнение зададим следующим преобразованием таблицы инциденций.

$$\left\| \begin{matrix} \mathbf{r}_{0,0} & \mathbf{r}_{-2,0} & \mathbf{r}_{-1,1} \\ \mathbf{r}_{-2,2} & \mathbf{r}_{-2,0} & \mathbf{r}_{-1,1} \end{matrix} \right\| \rightarrow \left\| \mathbf{r}_{0,0} \quad \mathbf{r}_{-2,0} \quad \mathbf{r}_{-2,2} \right\|, \tag{2}$$

$$\left\| \begin{matrix} \mathbf{r}_{-2,2} & \mathbf{r}_{0,2} & \mathbf{r}_{-1,1} \\ \mathbf{r}_{0,0} & \mathbf{r}_{0,2} & \mathbf{r}_{-1,1} \end{matrix} \right\| \rightarrow \left\| \mathbf{r}_{0,0} \quad \mathbf{r}_{0,2} \quad \mathbf{r}_{-2,2} \right\|, \tag{3}$$

$$\left\| \begin{matrix} \mathbf{r}_{0,0} & \mathbf{r}_{0,2} & \mathbf{r}_{1,1} \\ \mathbf{r}_{2,2} & \mathbf{r}_{0,2} & \mathbf{r}_{1,1} \end{matrix} \right\| \rightarrow \left\| \mathbf{r}_{0,0} \quad \mathbf{r}_{0,2} \quad \mathbf{r}_{2,2} \right\|, \tag{4}$$

$$\left\| \begin{matrix} \mathbf{r}_{2,2} & \mathbf{r}_{2,0} & \mathbf{r}_{1,1} \\ \mathbf{r}_{0,0} & \mathbf{r}_{2,0} & \mathbf{r}_{1,1} \end{matrix} \right\| \rightarrow \left\| \mathbf{r}_{0,0} \quad \mathbf{r}_{2,0} \quad \mathbf{r}_{2,2} \right\|, \tag{5}$$

$$\left\| \begin{matrix} \mathbf{r}_{0,0} & \mathbf{r}_{2,0} & \mathbf{r}_{1,-1} \\ \mathbf{r}_{2,-2} & \mathbf{r}_{2,0} & \mathbf{r}_{1,-1} \end{matrix} \right\| \rightarrow \left\| \mathbf{r}_{0,0} \quad \mathbf{r}_{2,0} \quad \mathbf{r}_{2,-2} \right\|, \tag{6}$$

$$\left\| \begin{matrix} \mathbf{r}_{2,-2} & \mathbf{r}_{0,-2} & \mathbf{r}_{1,-1} \\ \mathbf{r}_{0,0} & \mathbf{r}_{0,-2} & \mathbf{r}_{1,-1} \end{matrix} \right\| \rightarrow \left\| \mathbf{r}_{0,0} \quad \mathbf{r}_{0,-2} \quad \mathbf{r}_{2,-2} \right\|, \tag{7}$$

$$\left\| \begin{matrix} \mathbf{r}_{0,0} & \mathbf{r}_{0,-2} & \mathbf{r}_{-1,-1} \\ \mathbf{r}_{-2,-2} & \mathbf{r}_{0,-2} & \mathbf{r}_{-1,-1} \end{matrix} \right\| \rightarrow \left\| \mathbf{r}_{0,0} \quad \mathbf{r}_{-2,0} \quad \mathbf{r}_{-2,-2} \right\|, \tag{8}$$

$$\left\| \begin{matrix} \mathbf{r}_{-2,-2} & \mathbf{r}_{-2,0} & \mathbf{r}_{-1,-1} \\ \mathbf{r}_{0,0} & \mathbf{r}_{-2,0} & \mathbf{r}_{-1,-1} \end{matrix} \right\| \rightarrow \left\| \mathbf{r}_{0,0} \quad \mathbf{r}_{-2,0} \quad \mathbf{r}_{-2,-2} \right\|. \tag{9}$$

Легко видеть, что в результате получается правильная триангуляция.

Исходную триангуляцию (1) обозначим $\mathcal{T}$, описанную только что укрупненную (согласно формулам (2)–(9)) триангуляцию обозначим $\tilde{\mathcal{T}}$, а переход от исходной триангуляции к укрупненной обозначим $[\mathcal{T} \mapsto \tilde{\mathcal{T}}]$.

4. Калибровочные соотношения

Как известно, функцией Куранта, ассоциированной с выделенной вершиной правильной триангуляции, называется непрерывная функция, равная единице в упомянутой вершине, линейная на каждом треугольнике барицентрической звезды этой вершины и равная нулю вне указанной барицентрической звезды. Система функций Куранта — линейно независимая система.

В дальнейшем вектор $(i,\ j)$ будем обозначать через α; положим

$$\mathbf{r}_\alpha \stackrel{\text{def}}{=} \mathbf{r}_{i,j}, \qquad \mathbf{0} \stackrel{\text{def}}{=} (0,0), \qquad \mathbf{e} \stackrel{\text{def}}{=} (1,1), \qquad \mathbf{e}^* \stackrel{\text{def}}{=} (-1,1)$$

и введем множества

$$\mathbb{I}_1 = \{\mathbf{0}, \mathbf{e}, \mathbf{e}^*, -\mathbf{e}, -\mathbf{e}^*\}, \quad \mathbb{I}_1' = \mathbb{I}_1 \setminus \{\mathbf{0}\}, \quad 2\mathbb{I}_1 = \{\mathbf{0}, 2\mathbf{e}, 2\mathbf{e}^*, -2\mathbf{e}, -2\mathbf{e}^*\}.$$

Функцию Куранта, соответствующую выделенной вершине $\mathbf{r}_\alpha$ исходной триангуляции, обозначим $\omega_\alpha(t)$, $\alpha \in \mathbb{X}$, $t \in \mathbb{R}^2$. На исходной триангуляции имеется два типа функций Куранта, соответствующих двум типам барицентрических звезд: у функций Куранта с нечетными индексами носитель состоит из четырех треугольников, а у функций Куранта с четными индексами носитель состоит из восьми треугольников.

Для укрупненной триангуляции функцию Куранта, соответствующую выделенной вершине $\mathbf{r}_\alpha$, будем обозначать $\tilde{\omega}_\alpha$; заметим, что не все вершины исходной триангуляции участвуют в укрупненной триангуляции, а именно, индексы α пробегают не все множество $\mathbb{X}$, а лишь его часть: $\alpha \in \mathbb{X} \setminus \mathbb{I}_1'$; обозначим $\mathbb{Y} \stackrel{\text{def}}{=} \mathbb{X} \setminus \mathbb{I}_1'$.

Теорема 1. *Справедливы следующие калибровочные соотношения*

$$\tilde{\omega}_\alpha(t) \equiv \sum_{\gamma \in \mathbb{X}} \mathfrak{p}_{\alpha,\gamma}\ \omega_\gamma(t) \qquad \forall \alpha \in \mathbb{Y},$$

где

$$\mathfrak{p}_{\alpha,\gamma} \stackrel{\text{def}}{=} \delta_{\alpha,\gamma} \ \text{ при } \ \alpha \in \mathbb{X} \setminus \mathbb{I}_1 \setminus 2\mathbb{I}_1, \quad \gamma \in \mathbb{X},$$

$$p_{2\alpha,2\alpha} \stackrel{\text{def}}{=} 1 \ \text{ при } \ \alpha \in \mathbb{I}_1, \quad p_{0,\alpha} \stackrel{\text{def}}{=} 1/2,\ p_{2\alpha,\alpha} = 1/2 \ \text{ при } \ \alpha \in \mathbb{I}'_1;$$

здесь $\delta_{\alpha,\alpha'}$ — символ Кронекера.

Рассмотрим линейные пространства

$$\mathbb{S} \stackrel{\text{def}}{=} Cl_p\, \mathcal{L}(\{\omega_\gamma \,|\, \forall \gamma \in \mathbb{X}\}), \qquad \tilde{\mathbb{S}} \stackrel{\text{def}}{=} Cl_p\, \mathcal{L}(\{\tilde{\omega}_\alpha \,|\, \forall \alpha \in \mathbb{Y}\}),$$

где $\mathcal{L}(...)$ означает линейную оболочку множества элементов, указанного в круглых скобках, а Cl_p означает замыкание в топологии поточечной сходимости.

Ввиду теоремы 1 имеем

$$\tilde{\mathbb{S}} \subset \mathbb{S}. \tag{10}$$

5. Вэйвлетное разложение

Обозначим $\tilde{g}_\alpha$ линейный функционал в $C(\Omega)$, определяемый тождеством $\langle \tilde{g}_\alpha, u\rangle \stackrel{\text{def}}{=} u(r_\alpha)$ и рассмотрим оператор P проектирования пространства $C(\Omega)$ на подпространство $\tilde{\mathbb{S}}$, задаваемый формулой

$$Pu \stackrel{\text{def}}{=} \sum_{\alpha\in\mathbb{Y}} \langle \tilde{g}_\alpha, u\rangle \tilde{\omega}_\alpha \quad \forall u \in C(\Omega); \tag{11}$$

пусть $Q = I - P$, где I — тождественный в $C(\Omega)$ оператор.

Пространством вэйвлетов (всплесков) называется пространство $\mathbb{W} \stackrel{\text{def}}{=} Q\mathbb{S}$.

Благодаря соотношениям (10) и (11) получаем прямое разложение

$$\mathbb{S} = \tilde{\mathbb{S}} \dotplus \mathbb{W} \tag{12}$$

— *сплайн-вэйвлетное разложение* пространства $\mathbb{S}$.

Для исходного двумерного потока $\{c_\alpha\}$ построим сплайн $u \stackrel{\text{def}}{=} \sum_\alpha c_\alpha \omega_\alpha$ и представим его проекцию $\tilde{u} \stackrel{\text{def}}{=} Pu$ в виде линейной комбинации базисных сплайнов пространства $\tilde{\mathbb{S}}$: $\tilde{u} \stackrel{\text{def}}{=} \sum_{\beta\in\mathbb{Y}} a_\beta \tilde{\omega}_\beta$; вэйвлетную часть $w \in \mathbb{W}$ представим разложением по базису пространства $\mathbb{S}$, $w = \sum_\gamma b_\gamma \omega_\gamma$.

Справедлива следующие утверждения.

Теорема 2. *При локальном укрупнении триангуляции* $[\mathcal{T} \mapsto \tilde{\mathcal{T}}]$ *в вэйвлетном разложении (12) формулы декомпозиции имеют вид*

$$b_\alpha = 0, \quad a_\alpha = c_\alpha, \quad \alpha \in \mathbb{Y},$$

$$b_{-\mathbf{e}^*} = c_{-\mathbf{e}^*} - 12c_{-2\mathbf{e}^*} - 12c_{\mathbf{0}}, \qquad b_{-\mathbf{e}} = c_{-\mathbf{e}} - 12c_{-2\mathbf{e}} - 12c_{\mathbf{0}},$$

$$b_{\mathbf{e}} = c_{\mathbf{e}} - 12c_{2\mathbf{e}} - 12c_{\mathbf{0}}, \qquad b_{\mathbf{e}^*} = c_{\mathbf{e}^*} - 12c_{2\mathbf{e}^*} - 12c_{\mathbf{0}}.$$

Теорема 3. *Для локального укрупнения* $[\mathcal{T} \mapsto \tilde{\mathcal{T}}]$ *вэйвлетному разложению (12) соответствуют формулы реконструкции*

$$c_\alpha = a_\alpha \quad \forall \alpha \in \mathbb{Y},$$

$$c_{-e^*} = b_{-e^*} + 12a_{-2e^*} + 12a_0, \qquad c_{-e} = b_{-e} + 12a_{-2e} + 12a_0,$$

$$c_{\mathbf{e}} = b_{\mathbf{e}} + 12a_{2\mathbf{e}} + 12a_{\mathbf{0}}, \quad c_{\mathbf{e}^*} = b_{\mathbf{e}^*} + 12a_{2\mathbf{e}^*} + 12a_{\mathbf{0}}.$$

6. Заключение

Рассмотрен способ построения триангуляции, которая допускает локальное укрупнение с сохранением правильности. Укрупненная триангуляция

порождает курантовское пространство, которое вложено в подобное пространство, построенное для исходной триангуляции. В результате удается построить вэйвлетное разложение объемлющего пространства, которое позволяет учесть локальные особенности двумерного потока данных. Аналогичные построения для трехмерных потоков весьма актуальны, но существенных результатов в этом направлении пока нет.

Литература

1. *Демьянович Ю. К., Зимин А. В.* Аппроксимации курантова типа и их вэйвлетные разложения // Проблемы математического анализа. 2008. Вып. 37. С. 3–22.

О СРАВНИТЕЛЬНОМ АНАЛИЗЕ АЛГОРИТМОВ ШИФРОВАНИЯ

П. П. Чистяков

студент IV курса, кафедра параллельных алгоритмов математико-механического факультета;
Deutschland.cpp@bk.ru

Санкт-Петербургский государственный университет

Аннотация: Проведено сравнение алгоритмов ГОСТ 28147-89 и RC5 и установлено, что более гибким в отношении приспособляемости к различным вычислительным средствам является алгоритм RC5, а более устойчивым по отношению к криптоатакам взломщика шифров следует считать алгоритм ГОСТ 28147-89. Намечены перспективы дальнейших исследований криптостойкости алгоритмов, основанных на комбинации вариантов блочного шифрования с использованием вэйвлетных разложений и раундов, отождествляемых с генерацией вэйвлет-пакетов.

Введение

Алгоритмы ГОСТ 28147-89 и RC5 часто применяются для шифрования. Первый из указанных алгоритмов используется для шифрования и подписи в электронной почте, для обеспечения защищённой передачи данных между узлами сети Internet, для установления безопасного соединения между клиентом и сервером. Второй алгоритм применяется для шифрования запросов вызывающей стороны и ответа прибора при подключении к сети Internet, для защиты электронной почты от фальшивых адресов в некоторых коммерческих программах, для предотвращения несанкционированного доступа к системе электронных денег и т.п. Вопросы разработки, режимы работы и структура алгоритмов неоднократно исследовались; результаты этих исследований отражены, в частности, в работах [1–3].

Цель данной работы состоит в сравнении алгоритмов ГОСТ 28147-89 и RC5. В результате установлено, что более гибким в отношении приспособляемости к различным вычислительным средствам является алгоритм RC5, а более устойчивым по отношению к криптоатакам взломщика шифров следует считать алгоритм ГОСТ 28147–89. Намечены перспективы дальнейших исследований криптостойкости алгоритмов, основанных на комбинации вариантов блочного шифрования с использованием вэйвлетных разложений и раундов, отождествляемых с генерацией вэйвлет-пакетов.

Предварительные сведения

Базовая модель криптографии состоит в следующем. Сообщения передаются от отправителя к получателю по открытому каналу, к которому имеет

доступ «противник» (т.е. взломщик шифра). Он либо только прослушивает канал (пассивный перехват сообщений), либо модифицирует передаваемые сообщения и посылает свои (активный перехват сообщений). Перед отправкой сообщения в открытый канал отправителю необходимо подвергнуть исходное сообщение специальному преобразованию (зашифровыванию), которое позволяет скрыть информацию, содержащуюся в сообщении; после приёма на другом конце нужно выполнить обратное преобразование (расшифровывание). Исходное сообщение называется открытым текстом; открытый текст подается на вход устройства шифрования (шифратора). На выходе этого устройства получается шифртекст, который передается получателю. Шифртекст попадает на вход дешифратора, который преобразует его в исходный (открытый) текст. Несомненным условием упомянутых преобразований является сохранение информации, содержащейся в исходном тексте, на каждом из перечисленных этапов.

Существует три категории криптоалгоритмов: бесключевые, одноключевые и двухключевые. Одноключевые криптоалгоритмы характеризуются наличием общего секретного ключа для отправителя и получателя. Двухключевые отличаются тем, что на разных этапах вычислений применяются два вида ключей: для зашифровывания информации отправитель использует уникальный открытый ключ, предоставленный получателем; для расшифровывания получатель применяет соответствующий (парный) секретный ключ. В данной работе рассматриваются одноключевые алгоритмы.

Краткий обзор свойств алгоритмов

Переходя к характеристике алгоритма ГОСТ 28147-89, заметим, что в этом алгоритме шифрование информации осуществляется для каждого 64-битного блока; каждый из них разбивается на два 32-битных субблока. Для генерации подключей исходный 256-битный ключ разбивается на восемь 32-битных блоков.

Основой алгоритма служит сеть Фейстеля, причем в каждом из 16 или 32 раундов алгоритма (в зависимости от режима работы) выполняются преобразования с блоками с использованием подключей и таблиц замен. Имеется четыре режима работы алгоритма (описание этих режимов см. в книге [1]). Остановимся на двух из них, а именно, на режимах простой замены и гаммирования. В режиме простой замены при наличии часто повторяющихся блоков исходного текста соответствующий шифртекст содержит то же самое число одинаковых блоков; это резко снижает криптостойкость при работе в упомянутом режиме. В случае шифрования в режиме гаммирования результат шифрования очередного блока информации зависит от результата шифрования предыдущего блока, и, следовательно, от результата шифрования всех зашифрованных ранее блоков; такой режим существенно увеличивает криптостойкость алгоритма.

К достоинствам алгоритма ГОСТ 28147-89 можно отнести: 1) эффективность реализации и высокое быстродействие на современных компьютерах; 2) использование одинакового шифрующего преобразования во всех режимах, несмотря на их различное предназначение; 3) наличие защиты от навязывания ложных данных (выработка имитовставки в соответствующем режиме); 4) большую длину ключа (256 битов; длина ключа увеличивается до 320 битов при использовании секретной синхропосылки или имитоприставки); 5) 32 раунда преобразований, гарантирующие влияние одного бита открытого текста на все биты шифртекста (это обеспечивает значительную стойкость алгоритма). Заметим, что в настоящий момент существует немного работ по криптоанализу данного алгоритма.

К недостаткам относится использование слабых таблиц замен (например, «тождественной»), которые иногда могут быть предоставлены преднамеренно (возможно, потенциальным взломщиком шифра). Без информации о них невозможно определить криптостойкость алгоритма, к тому же при использовании различных таблиц замен разными производителями получающиеся версии алгоритма могут быть несовместимы между собой.

Алгоритм RC5 является блочным шифром. Здесь используются операции сложения по модулю 2^w, «исключающее или» (XOR), а также операция циклического сдвига на переменное количество битов (вносящая существенную новизну). На современных процессорах эти операции выполняются достаточно быстро.

Параметрами алгоритма RC5 являются: 1) размер w слова в битах, что составляет половину длины блока шифрования; может принимать одно из значений 16, 32 или 64 (рекомендуется брать значение, равное длине машинного слова); 2) количество R раундов алгоритма (любое целое число в диапазоне от 0 до 255): чем больше это значение, тем выше уровень безопасности шифра (заметим, что при R=0 информация не зашифровывается); 3) секретный ключ и его размер b в байтах. При шифровании/расшифровывании применяется процедура расширения ключа. Основой алгоритма RC5 служит сеть Фейстеля.

Перечислим положительные стороны алгоритма RC5:

1. Присутствие переменных параметров ведет к большой гибкости алгоритма: если использовать длинный ключ и большое количество раундов, то имеем большую надёжность алгоритма, а при наличии короткого ключа и малого количества раундов можно достичь значительного ускорения вычислений. Кроме того, можно эффективно учесть длину машинного слова, используемую аппаратурой, что приведет к упрощению программы и ускорению работы.
2. Применяемые операции легко реализуются как аппаратно, так и программно на различных процессорах. Алгоритм не требователен к объему памяти; поэтому его можно реализовать в портативных устройствах.

Недостаток алгоритма RC5 состоит в том, что параметры могут быть перехвачены взломщиком шифра.

Выводы

Таким образом, сравнение рассмотренных выше алгоритмов позволяет сделать вывод, что более гибким в отношении приспособляемости к различным вычислительным средствам несомненно является алгоритм RC5, а более устойчивым по отношению к криптоатакам взломщика шифров следует считать алгоритм ГОСТ 28147–89 (однако это не относится к режиму простой замены).

Заключение

Большой интерес представляет создание алгоритмов симметричного шифрования, сочетающих в себе использование нескольких видов структур, например, как сети Фейстеля, так и структуры «квадрат», и исследование их криптостойкости. Как известно, слабо изученные криптосистемы подчас становятся весьма эффективными средствами шифрования; примером такого рода является алгоритм Rijndael, являющийся стандартом шифрования в США.

Совмещение различных структур возможно путём применения матричных преобразований к блоку данных, представленному в виде двумерного байтового массива фиксированного размера, после выполнения каждого из раундов сети Фейстеля, где в качестве блоков данных используются блоки или субблоки, возникающие при разбиении исходных данных.

В дальнейшем предполагается исследовать криптостойкость алгоритмов, основанных на комбинации рассмотренных здесь вариантов блочного шифрования с использованием вэйвлетных разложений; последние характеризуются нелинейной зависимостью от ключа, который порождает сетки вэйвлетных разложений, а также последовательностями раундов — итерациями вэйвлетных разложений, которыми сопровождается создание вэйвлет-пакетов (см. [4]). Ввиду локальности рассматриваемых преобразований можно надеяться на эффективное применение параллельных вычислительных систем при реализации комбинированных алгоритмов.

Литература

1. *Панасенко С. П.* Алгоритмы шифрования. Специальный справочник. СПб.: БХВ-Петербург, 2009. 576 с.: ил.
2. *Винокуров А.* Алгоритм шифрования ГОСТ 28147–89, его использование и реализация для компьютеров платформы Intel x86 (http://www.twirpx.com/file/220569)
3. *Rivest R. L.* The RC5 Encryption Algorithm. 1994.
4. *Демьянович Ю. К., Левина А. Б.* О вэйвлетных разложениях линейных пространств над произвольным полем и о некоторых приложениях // Математическое моделирование. 2008. Т. 20. № 11. С.104–108.

Теория и практика защиты и кодирования информации

Крук
Евгений Аврамович

д.т.н., профессор
декан факультета информационных систем и защиты информации
заведующий кафедрой
безопасности информационных систем СПбГУАП

Трифонов
Петр Владимирович

к.т.н., доцент СПбГПУ

ОЦЕНКА МИНИМАЛЬНОГО РАССТОЯНИЯ КВАЗИЦИКЛИЧЕСКИХ КОДОВ

А. Ю. Абрамов

асп. кафедры комплексной защиты информации;
aabramov@promo.ru

Санкт-Петербургский государственный университет аэрокосмического приборостроения

Аннотация: В статье изучается класс квазициклических кодов исправляющих ошибки. Известно, что коды, принадлежащие этому классу достигают границы Варшамова — Гилберта.

В работе излагается подходы к улучшению оценки минимального расстояния квазициклических (КЦ) кодов со скоростью 1/2, которая была предложена Е. А. Круком в работе [11], — она позволяет оценивать вес кодовых слов исходного КЦ-кода через вес слов циклического кода, полученного при сложении половин. Данная оценка может быть уточнена, за счет проверки некоторых свойств над множеств слов минимального веса.

Введение

Квазициклическим называется линейный блоковый (n, k) — код, где $n = mn_0$, а $k = mk_0$, если в нем каждый циклический сдвиг кодового слова на n_0 позиций также является кодовым словом.

Циркулянтная $m \times m$ матрица определяется как:

$$C = \begin{bmatrix} c_0 & c_1 & \dots & c_{m-1} \\ c_{m-1} & c_0 & \dots & c_{m-2} \\ \vdots & \vdots & & \vdots \\ c_1 & c_2 & \dots & c_0 \end{bmatrix},$$

где c_i — элемент конечного поля GF (q).

Тогда порождающая матрица квазициклического (mn_0, mk_0) — кода над GF (q) при помощи перестановок строк и столбцов может быть приведена к виду:

$$G = \begin{bmatrix} C_{11} & C_{12} & \dots & C_{1n_0} \\ C_{21} & C_{22} & \dots & C_{2n_0} \\ \vdots & \vdots & & \vdots \\ C_{k_0 1} & C_{k_0 2} & \dots & C_{k_0 n_0} \end{bmatrix},$$

где C_{ij} — $m \times m$ циркулянтная матрица.

В [1] было показано, что алгебра $m \times m$ циркулянтных матриц изоморфна алгебре многочленов в кольце $F[x]/(x_m - 1)$, следовательно любая циркулянтная матрица может быть задана, также, и при помощи циркулянтного полинома $c(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{m-1} x^{m-1}$, где коэффициенты взяты из её первой строки.

Квазициклические коды были введены Таунсендом и Уэлдоном в работе [4], а затем изучались в работах [5], [6]. В работе [10] были применены методы линейного программирования для поиска «хороших» кодов, и достаточно большое число таковых было найдено, однако такой подход неприменим для поиска на больших длинах.

В статье изложены подходы для оценки минимального расстояния некоторого подкласса квазициклических кодов, циркулянтный многочлен которого удовлетворяет специальным свойствам.

Оценка минимального расстояния квазициклических кодов

Изложение способа получения оценки

Рассмотрим двоичный квазициклический код с $n_0 = 2, k_0 = 1$ со скоростью ½ над полем $\mathrm{GF}(2^m)$, порождающую матрицу которого можно привести к следующему виду:

$$G = \left(\begin{array}{ccc|cccc} 1 & & & c_0 & c_1 & \ldots & c_{m-1} \\ & 1 & & c_{m-1} & c_0 & \ldots & c_{m-2} \\ & & \ddots & \vdots & \vdots & & \vdots \\ & & 1 & c_1 & c_2 & \ldots & c_0 \end{array}\right).$$

Тогда вес его слова может быть оценен как

$$wt(iG) = wt(i + iC) \geq wt(i(C + I)) = wt(iG_+),$$

где i — информационное слово длины m, а

$$G_+ = \begin{pmatrix} c_0 + 1 & c_1 & \ldots & c_{m-1} \\ c_{m-1} & c_0 + 1 & \ldots & c_{m-2} \\ \vdots & \vdots & & \vdots \\ c_1 & c_2 & \ldots & c_0 + 1 \end{pmatrix},$$

т. е. $c_+(x) = c(x) + 1$. В том случае, если матрица G_+ вырождена (а $c_+(x) \mid (x^m - 1)$), она задает некоторый циклический (m, k¢) — код с минимальным расстоянием d.

Тогда выполняется одно из следующих тождеств:

$$\left[\begin{array}{l} wt(iG_+) \geq d \\ wt(iG_+) = 0 \end{array}\right.,$$

и если $wt(iG_+) = 0$, следовательно i является кодовым словом кода, дуального к коду, задаваемому порождающей матрицей G_+. Пусть дуальный код имеет минимальное расстояние $d_\perp$. Тогда $wt(i) \geq d_\perp \Rightarrow wt(i+iC) \geq wt(i) + wt(iC) \geq 2d_\perp$, а если i — слово минимального веса в дуальном коде, то $wt(i+iC) \leq 2d_\perp$.

Следовательно минимальное расстояние квазициклического кода такого вида может быть оценено как:

$$\min(d, 2d_\perp) \leq d_{qc} \leq 2d_\perp \quad (2).$$

Подходы к увеличению оценки

Очевидно, что верхнюю границу оценки (2) уточнить невозможно, т. к. слова, удовлетворяющие верхней границе, содержатся в оцениваемом квазициклическом коде. Однако, было найдено несколько подходов, позволяющие уточнить оценку снизу для минимального расстояния.

Найдем вес слова, получающегося сложением половин исходного слова квазициклического кода:

$$wt(i(x) + i(x)c(x)) = wt(i(x)) + wt(i(x)c(x)) - 2wt(i(x) * i(x)c(x)),$$

где * — операция, обозначающая скалярное произведение. Можно заметить, что $wt(i(x)) + wt(i(x)c(x))$ есть вес слова исходного квазициклического кода, а $wt(i(x) + i(x)c(x)) \geq \min(d, d_\perp)$ по (2).

Следовательно, оценка (2) будет точной в том случае, когда хотя бы для одного слова минимального веса в коде, задаваемом $c_+(x)$, будет выполняться свойство

$$wt(i(x) * i(x)c(x)) = 0 \quad (3).$$

В этом случае вес соответствующего слова квазициклического кода, очевидно, будет равен d. Если условие (3) не выполняется ни для одного слова из множества слов минимального веса, то оценка (2) может быть улучшена и приведена к виду $\min(d+1, 2d_\perp) \leq d_{qc} \leq 2d_\perp$.

Свойство (3) может быть модифицировано для слов циклического кода, вес которых равен (d+1), и если для этого множества модифицированное свойство (3) не выполняется, то оценку (2) можно уточнить еще больше. Очевидно, что такой подход можно применять до тех пор, пока не будет найдено истинное минимальное расстояние исходного квазициклического кода.

Перестановки

При получении оценки (2) было использование сложение половин кодового слова квазициклического кода:

$$wt(iG) = wt(i + iC) \geq wt(i(C + I)) = wt(iG_+),$$

однако можно не просто складывать половины, но и применять к каждой из них перестановки из группы автоморфизмов кода. В частности, очевидно, что, например, циклическая перестановка, примененная к матрице I или перестановка $i \to 2i$, примененная к C, не изменяют исходного кода.

Тогда оценку (2) можно модифицировать и привести к следующему виду:

$$wt(iG) = wt(i + iC) \geq \min_{\pi,\mu}[wt(i(\pi C + \mu I))].$$

Использование множества перестановок позволяет сократить перебор, необходимый для проверки условия (3) и как следствие — уменьшить сложность улучшения оценки (2).

Покрытия

В том случае, если выполняется условие (3), можно сказать, что $wt(i^*(ic + i)) = i$. Если затем сложить половины кодового слова квазициклического кода, как это было сделано раньше, получится, что должно выполняться условие $wt(i(x)*i(x)c_+(x)) = i(x)$ (4). Тогда, оценку (2) можно уточнить в том случае, если всё множество кодовых слов минимального веса кода, задаваемого порождающим многочленом $c_+(x)$, будет покрывать соответствующие им информационные многочлены $i(x)$, т.е. будет выполняться условие (4). На данный момент не предложено какого-либо эффективного алгоритма для нахождения множества кодовых слов, которые покрывают соответствующие им информационные многочлены, однако, если известно множество слов минимального веса, то для них можно проверить выполнение условия (4), и либо найти истинное минимальное расстояние исходного квазициклического кода, либо улучшить оценку (2).

Сложность применения метода покрытий для улучшения оценки (2) напрямую зависит от сложности метода нахождения множества слов минимального веса в некотором циклическом коде. В самом простом случае, если используется перебор слов, ее можно оценить как $2^{n-\deg(c_+(x))}$.

При этом для нахождения множества слов минимального веса существуют более эффективные алгоритмы, в частности — при помощи построения базиса Гребнера.

Заключение

В работе предложены методы по улучшению оценки минимального расстояния квазициклических кодов, предложенной Е. А. Круком. При помощи модифицированной оценки удалось получить коды, лучшие из известных, однако, следует отметить, что используя данный метод, получить коды, лежащие на границе Варшамова-Гилберта, не удастся.

Литература

1. *MacWilliams, F. J.* and *Sloane, N. J. A.*, The Theory of Error-Correcting Codes, North-Holland Publishing Co., 1977.
2. *Weldon, E. J., Jr.*, «Long Quasi-Cyclic Codes are Good», (abstract), IEEE Trans. Inf. Theory, vol. IT-13, pp. 130, 1970.
3. *Kasami, T.*, «A Gilbert-Varshamov Bound for Quasi-Cyclic Codes of Rate 1/2», IEEE Trans. Inf. Theory, vol. IT-20, p. 679, 1974.
4. *Townsend, R. L.*, and *Weldon, E. J., Jr.*, «Self-Orthogonal Quasi-Cyclic Codes», IEEE Trans. Inf. Theory, vol. IT-13, pp. 183–195, 1967.
5. *Karlin, M.*, «New Binary Coding Results by Circulants», IEEE Trans. Inf. Theory, vol. IT-15, pp. 81–92, 1969.
6. *Karlin, M.*, «Decoding of Circulant Codes», IEEE Trans. Inf. Theory, vol. IT-16, pp. 797–802, 1970.
7. *Hoffner, C. W.* and *Reddy, S. M.*, «Circulant Bases for Cyclic Codes», IEEE Trans. Inf. Theory, vol. IT-16, pp. 511–512, 1970.
8. *Bhargava, V. K., Seguin, G. E.* and *Stein, J. M.*, «Some (mk, k) Cyclic Codes in Quasi-Cyclic Form», IEEE Trans. Inf. Theory, vol. IT-25, pp. 112–118, 1979.
9. *Solomon, G.* and *van Tilborg, H. C. A.*, «A Connection Between Block Codes and Convolutional Codes», J. Soc. Ind. Appl. Math., vol. 37, pp. 358–369, 1979.
10. *Gulliver, T.*, «Construction of Quasi-Cyclic Codes», Univercity of Victoria, 1989.
11. Крук *Е.* А. «Передача сообщения обобщенными квазициклическими кодами по дискретным каналам связи», ЛИАП, 1978.

СПИСОЧНОЕ ДЕКОДИРОВАНИЕ ТУРБО КОДОВ

А. И. Акмалходжаев
akmal.ilh@gmail.com

А. В. Козлов
akozlov@vu.spb.ru

Санкт-Петербургский государственный университет аэрокосмического приборостроения

Аннотация: Данная работа посвящена задаче списочного декодирования турбо кодов. Рассматривается новый метод списочного декодирования, в рамках которого предложен списочный декодер сверточного кода с мягким выходом. В то время как известные методы дают выигрыш лишь в области насыщения вероятности ошибки, предложенный метод списочного декодирования дает выигрыш в области спада вероятности ошибки, т.е. наиболее интересной с практической точки зрения.

Введение

Начиная с момента появления турбо кодов в 1996 году [2], область применения этих кодов неуклонно растет. Основными элементами турбо кода являются два рекурсивных систематических сверточных кода, связанные между собой интерливером. Выходом турбо кода являются информационная часть и проверочные биты каждого сверточного кода (рис. 1.).

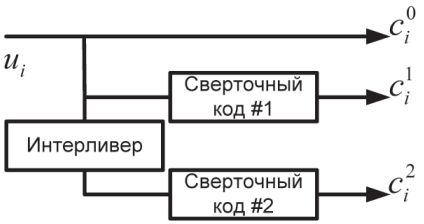


Рис. 1. Схема кодера турбо кода

Ядром декодера турбо кода являются два декодера сверточных кодов с мягким выходом [1] [4]. Помимо информационных бит, мягкий декодер сверточного кода рассчитывает надежности каждого бита, из которых можно выделить внешнюю информацию. В процессе декодирования турбо кода два сверточных декодера несколько раз обмениваются внешней информацией. Результаты имитационного моделирования показывают, что на практике достаточного 8–16 итераций для того, чтобы декодер сошелся, после чего

дальнейшее итерирование турбо декодера не дает уменьшения вероятности ошибки. Одним из способов улучшить производительность турбо декодера является использование списочного декодирования.

Существующие методы списочного декодирования турбо кодов дают лишь выигрыш в области насыщения вероятности ошибки [6], но практически не дают выигрыша в области спада, наиболее интересной с практической точки зрения. В данной работе рассматривается новый метод списочного декодирования турбо кодов, в рамках которого предлагается списочный декодер сверточного кода с мягким выходом.

Работа построена следующим образом. Во втором разделе статьи представлена общая схема декодирования турбо кода. В третьем разделе предлагается новый метод списочного декодирования сверточного кода с мягким выходом и описан метод списочного декодирования турбо кода. В четвертом разделе представлены результаты моделирования предложенного метода в канале с белым Гауссовским шумом (АБГШ), на примере турбо кода стандарта 3GPP LTE [9].

Декодирование турбо кодов

На вход турбо декодера, схема которого представлена на рис. 2, подаются надежности символов, которые можно разделить на три части: систематическую часть $L_s^0(x_t)$. проверки первого сверточного кода $L_p^1(x_t)$ и проверки второго сверточного кода $L_p^2(x_t)$. Используя входные надежности, сверточные декодеры рассчитывают внешнюю информацию ($L_{ext,i}^1(x_t)$ и $L_{ext,i}^2(x_t)$), которая на входе следующего декодера становится априорной информацией ($L_{apr,i}^1(x_t)$ и $L_{apr,i}^2(x_t)$) и которой они обмениваются на протяжении нескольких итераций.

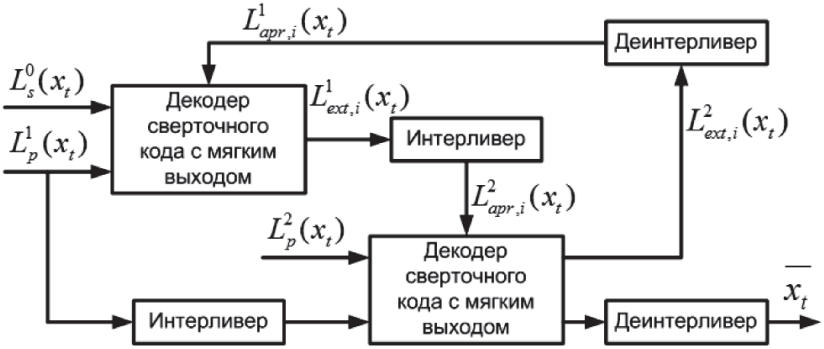


Рис. 2. Схема декодера турбо кода

Декодер по максимуму апостериорной вероятности

Обычно в качестве декодера сверточного кода с мягким выходом используют декодер по максимуму апостериорной вероятности (MAP декодер), ко-

торый является оптимальным декодером сверточного кода с точки зрения минимизации вероятность ошибки на символ [1]. Обычно MAP алгоритм реализуется в логарифмической области и называется Log-MAP алгоритмом. Также известны его субоптимальные варианты: Max-Log-MAP и Sclaed-Max-Log-MAP [3].

Для расчета выходных надежностей алгоритм Log-MAP выполняет прямой и обратный проход по решетке сверточного кода [2]. Надежности информационных бит на выходе Log-MAP декодера для систематического сверточного кода могут быть представлены в следующем виде:

$$L(x_t) = \log\left(\frac{P(x_t = 1 \mid \overline{r})}{P(x_t = 0 \mid \overline{r})}\right) = L_c(x_t) + L^1_{apr,i}(x_t) + L^1_{ext,i}(x_t),$$

из которых легко получить $L^1_{ext,i}(x_t)$.

Списочный декодер турбо кода

Существующие подходы списочного декодирования турбо кодов используют списочный алгоритм Витерби, который находит M наиболее вероятных путей в решетке сверточного кода [5, 6]. Таким образом, список битовых последовательной генерируется после последней итерации турбо кода, либо на каждой итерации декодера.

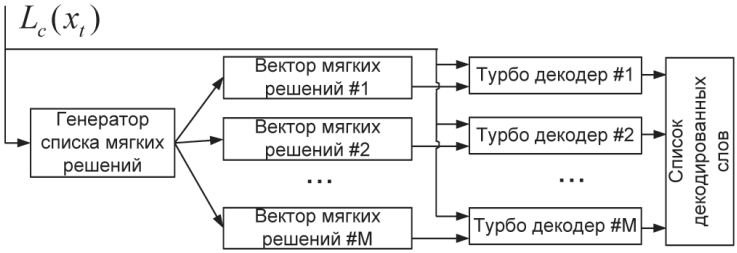


Рис. 3. Схема списочного декодера турбо кода

В предлагаемом методе вначале процесса турбо декодирования генерируется список внешних вероятностей (мягких решений), которые, в свою очередь, подаются на вход независимых турбо декодеров (рис. 3). Цель данного метода в том, чтобы разные декодеры сошлись к разным кодовым словам, среди которых далее выбирается искомое слово. Список мягких решений может генерироваться с использованием первого, второго или обоих сверточных кодов. Однако основной задачей предложенного подхода списочного декодирования является нахождение самого списка мягких решений.

Списочный декодер сверточного кода с мягким выходом

Для решения данной задачи, предлагается алгоритм, который совмещает в себе списочный декодер Витерби и Log-MAP декодер. Мягкие решения, полученные на выходе декодера, являются надежностями бит, которые были получены с помощью алгоритма Log-MAP на измененной решетке. В общем виде алгоритм выглядит следующим образом:

1. Найти M самых вероятных путей в решетке сверточного кода помощью списочного Витерби алгоритма [9, 10].
2. Первый мягкий выход идентичен выходу Log-MAP декодера.
3. Обозначим как Γ_m, где $m = 1, ..., M$, множество всех ребер в решетке, которые принадлежат пути m. Для нахождения k-ого мягкого выхода:
   3.1. Выколем из решетки все ребра, которые принадлежат множествам Γ_m, для $m = 1, ..., k-1$, и вставим обратно ребра, которые принадлежат множествам Γ_m, для $m = k, ..., M$. Т. е. выколем из решетки все ребра из множества $\left(\bigcup_{m=1}^{k-1}\Gamma_m - \bigcup_{m=1}^{k-1}(\Gamma_m \cap \Gamma_k)\right)$.
   3.2. Выполним Log-MAP алгоритм на решетке с выколотыми ребрами.

Удаляя из решетки ребра лучших путей, которые будут рассмотрены в соответствующих элементах списка, мы пытаемся рассматривать менее вероятные решения. Таким образом, мы рассчитываем на то, что последующие процессы турбо декодирования будут сходиться к другим кодовым словам, среди которых возможно будет переданное.

Результаты моделирования

Анализ предложенного метода производился с помощью моделирования на примере турбо кода LTE [7]. Схема модели представлена на рис. 4.

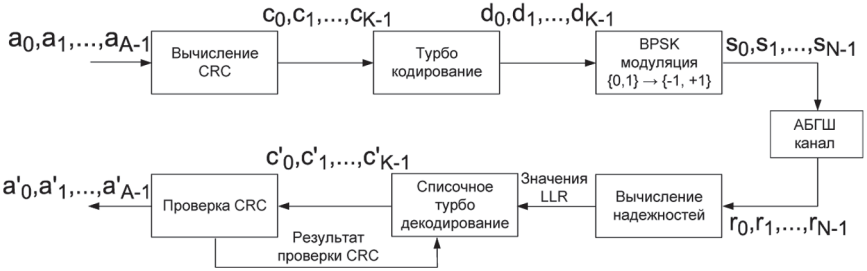


Рис. 4. Схема тестовой модели

Как видно на схеме, информационное слово содержит в себе CRC код, который используется для выбора правильного слова из списка. В качестве CRC используется полином CRC_B из стандарта 3GPP LTE [9].

Список мягких решений генерируется с помощью обоих решеток сверточных кодов. Т. е. длина итогового списка равна 2*M*. Так как турбо декодеры запускаются последовательно, то моделирование продолжается до тех пор, пока не будет удовлетворена проверка CRC.

Внимание при сравнении уделялось вероятности ошибки на информационное слово.

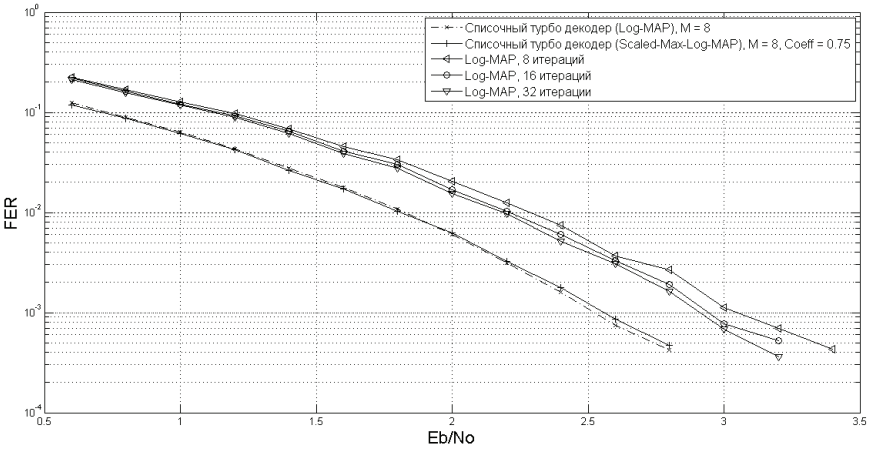


Рис. 5. Результаты моделирования ($K = 40$)

Из результатов моделирования видно (рис. 5), что увеличение числа итераций обычного турбо декодера практически не дает улучшения. Предложенный метод по сравнению с Log-MAP алгоритмом дает выигрыш порядка 0.5 дБ для списка длины 16. Также приведены результаты моделирования для списочного декодера, который использует Scaled-Max-Log-MAP.

Заключение

В данной работе был рассмотрен новый метод списочного декодирования турбо кодов, в рамках которого был предложен новый алгоритм списочного декодирования сверточного кода. Предложенный метод позволяет добиться выигрыша по сравнению с обычным Log-MAP декодером вплоть до 0.5 дБ на списке длины 16. Развитие предложенного метода может быть связано с уменьшением числа итераций в независимых турбо декодерах с использованием критериев остановки [10]. Также вместо Log-MAP алгоритма могут быть использованы его подоптимальные варианты. Из графика также видно, что на короткой длине, для которой проводилось моделирование, использование подоптимального варианта дает практически такой же результат, что и оптимальный.

Литература

1. *L. Bahl*. Optimal decoding of linear codes for minimizing symbol error rate / L. Bahl, J. Cocke, F. Jelinek, J. Raviv: IEEE transactions on Information Theory, 284–287, 1974, Март.
2. *C. Berrou*. Near Shannon limit error-correcting coding: turbo codes / C. Berrou, A. Glavieux, P. Thitimajshima: Processing IEEE International Conference on Communications, Geneva, Switzerland, 1064–1070, 1993.
3. *P. Robertson*. A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain / P. Robertson, E. Villebrun, P. Hoeher: Proc. Int. Conf. Communications, 1009–1013, Июнь 1995.
4. *J. Hagenauer*. A Viterbi Algorithm with Soft-Decision Outputs and its Applications / J. Hagenauer, P. Hoeher: Proc. IEEE GLOBECOM, 1680–1686, Ноябрь 1989.
5. *C. Nill*. List and Soft Symbol Output Viterbi Algorithms: Extensions and Comparisons / C. Nill, C.-E. W. Sundberg: IEEE Trans. Commun, 277–287, Январь 1993.
6. *K. R. Narayanan*. List Decoding of Turbo Codes / K. R. Narayanan, G. L. Stuber: IEEE Trans. Commun, 754–762, Июнь 1998.
7. *V. Guruswami*. List Decoding of Error-Correcting Codes / V. Guruswami: Submitted for the degree of Doctor of Philosophy at the Massachusetts institute of technology, Сентябрь 2001.
8. *P. Elias*. List decoding for noisy channels / P. Elias: Technical Report 335, Research Laboratory of Electronics, MIT, 1957.
9. 3GPP LTE TS 36.212: «Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding». Техническая спецификация.
10. *A. Matache*. Stopping Rules for Turbo Decoders / A. Matache, S. Dolinar, F. Pollara: TMO Progress Report 42–142, август 2000.

МЕТОДЫ ИСПОЛЬЗОВАНИЯ АКУСТИЧЕСКИХ СВОЙСТВ СЕРДЦА В СИСТЕМАХ АУТЕНТИФИКАЦИИ

Е. А. Андреева

студентка кафедры комплексной защиты информации;
eandreeva89@gmail.com

Санкт-Петербургский государственный университет аэрокосмического приборостроения

Введение

Идентификация и аутентификация пользователя в системе являются одними из важных задач информационной безопасности. С помощью данных процедур осуществляется управление доступом в информационное пространство. Но с развитием информационных технологий становится важно не только проверять подлинность пользователя при входе в систему, но и контролировать его аутентичность во время работы в системе, для того чтобы исключить возможность осуществления несанкционированного доступа от лица авторизованного пользователя.

Для решения этой задачи необходимо непрерывно проводить аутентификацию пользователя в системе. В статье предложена модель системы аутентификации, которая обеспечивает постоянный контроль авторизованного пользователя. В качестве самой процедуры аутентификации предложено использовать биометрическую технологию, основанную на акустических свойствах сердца. Такой метод отличается от других способов аутентификации, и применительно к системе, рассмотренной в данной статье, имеет ряд преимуществ.

Модель системы аутентификации

Модель функционирования системы аутентификации представлена на рисунке 1. Система обладает следующими свойствами:

- непрерывное накопление биометрических данных;
- непрерывная аутентификация;
- непрерывное обновление биометрических данных;
- непрерывное ведение статистики.

В данной системе процедура аутентификации должна происходить независимо от действий пользователя, но при этом оставаться надежной и устойчивой к атакам. Эти факторы играют решающую роль при выборе метода аутентификации для данной системы. Но прежде чем перейти к рассмотрению выбранного метода, приведем сравнительную характеристику современных способов аутентификации.

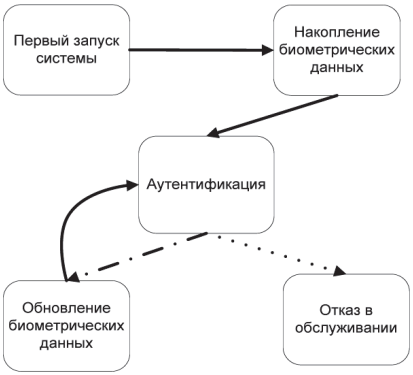


Рис. **1.** Модель функционирования системы аутентификации

Современные способы аутентификации

Способы аутентификации можно разделить на три категории:

- Пароль — аутентификация с помощью информации, которую знает пользователь;
- Устройство аутентификации — аутентификации с помощью устройства, которым обладает пользователь;
- Биометрия — аутентификация с помощью особой физической или психологической черты, которой обладает пользователь;

Биометрия является наиболее простым способом аутентификации с точки зрения пользователя. Нет необходимости запоминать пароль или носить с собой устройство аутентификации. Но с другой стороны биометрия наиболее дорогостоящий и сложный в реализации метод аутентификации. Выбор метода аутентификации зависит от свойств и характеристик конкретной системы.

В таблице 1 представлена сравнительная характеристика методов аутентификации, которые можно применить в рассмотренной системе.

Таблица 1

Сравнительная характеристика современных способов аутентификации

Название	Преимущества	Недостатки
Пароль	Простота использования	Возможность потери и кражи Обязательная процедура ввода данных
Устройство аутентификации (ключ)	Простота использования	Возможность потери и кражи Обязательная процедура ввода данных

Название	Преимущества	Недостатки
Отпечаток пальца	Высокая точность аутентификации Надежность	Биометрическая характеристика может быть утрачена или повреждена Обязательная процедура ввода данных
ДНК	Высокая точность аутентификации Надежность Необязательная процедура ввода данных	Сложная процедура анализа Сложная процедура ввода данных
Голос	Простота использования Простота сбора данных	Возможность подмены Обязательная процедура ввода данных

Из таблицы видно, что в большинстве способов аутентификации требуется обязательная процедура ввода данных, поэтому их невозможно применять в данной системе. Без процедуры ввода данных можно обойтись в случае аутентификации по ДНК, но ДНК анализ является сложным и долгим, поэтому эта технология также исключается из рассмотрения.

Метод аутентификации с использованием акустических свойств сердца

В качестве метода аутентификации в рассмотренной системе предлагается использовать биометрическую технологию, основанную на акустических свойствах сердца. В тонах сердца содержится уникальная информация, поэтому они могут быть использованы, как биометрическая характеристика человека.

Данная технология отличается от остальных следующими свойствами:

- тоны сердца не могут быть утрачены в течение жизни;
- тоны сердца сложно подделать;
- аутентификация может производиться без действий пользователя.

Но сложность данной технологии состоит в том, что звуки сердца могут изменяться в течение жизни.

На рисунке 2 представлены сигналы сердцебиения для пяти разных людей. Они обладают периодичностью и отличаются друг от друга.

Спектры сигнала сердцебиения одного человека, отличаются характерной формой, которая сохраняется при изменении интенсивности или темпе сигнала. Это важно, потому что частотные характеристики человека могут быстро меняться во времени в зависимости от его физического или эмоционального состояния. Но неизменной остается «мелодия» сердцебиения, то есть последовательность смены частот в спектре.

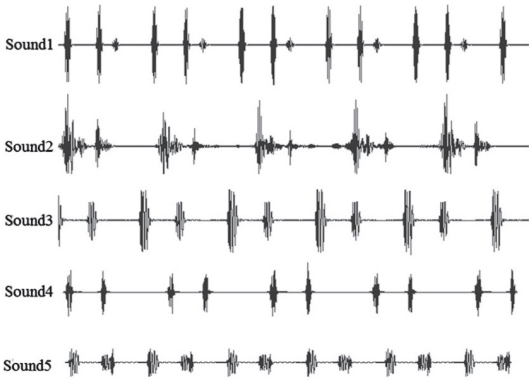


Рис. **2.** Звуки сердцебиения для пяти разных людей

При аутентификации важно решить две задачи:

- отличить сигналы сердцебиения для двух разных людей;
- распознать сигнал сердцебиения для одного человека, но с изменившимися характеристиками;

Алгоритм сравнения двух сигналов сердцебиения:

1. Разбиение спектра сигнала на области (1… *n*)
2. Нахождение максимума спектра
3. Разбиение спектра на уровни (1… *q*)
4. Нахождение максимума для каждой области спектра
5. Составление кодового слово, в котором длина равна *n*, а значение равно номеру уровня, в который попадает максимум спектра
6. Анализ кодовых слов, с целью определения формы спектра

На рисунке 3 представлена схема разбиения спектра на области и на уровни.

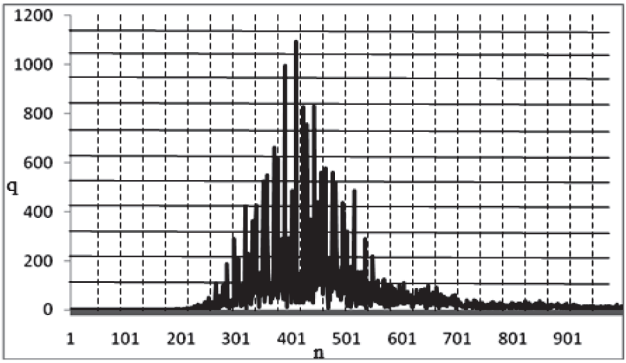


Рис. **3.** Спектр сигнала сердцебиения разбитый на области и уровни

Номер уровня определяет значение в ячейке кодового слова. В таблице 2 представлены кодовые слова для разных сигналов. Первые три последовательности являются частями кодового слова для сигнала сердцебиения одного человека. Сигналы для одного человека взяты с разными частотными характеристиками. Видно, что значения в ячейках не сильно отличаются даже при использовании грубого метода оцифровки сигнала. Четвертая последовательность получена для сигнала другого человека. Она отличается от первых трех, что позволяет отличить данный сигнал от остальных.

Таблица 2

Части кодовых последовательностей звуков сердцебиений

Набор звуков	Значения q для областей n на частотах от 300 до 600						
Звук сердцебиения №1 Нормальные характеристики	14	15	16	17	20	19	18
Звук сердцебиения №1 Ускорение темпа	9	12	18	20	15	10	9
Звук сердцебиения №1 Увеличение громкости	8	11	16	18	20	15	16
Звук сердцебиения №2	2	5	0	0	9	9	9

Заключение

В статье предложена система аутентификации, с помощью которой можно осуществить непрерывный контроль авторизованного пользователя в системе.

Показан метод использования акустических свойств сердца для осуществления процедуры аутентификации. С помощью данного метода можно проводить аутентификацию пользователя независимо от его действий.

Также данная биометрическая технология дает возможность контролировать состояния пользователя при работе в системе. Изучение данного вопроса является задачей для дальнейших исследований.

Литература

1. *F. Beritelli* and *A. Spadaccini*. «Human Identity Verification Based on Heart Sounds: Recent Advances and Future Directions»; University of Catania, Italy 2010.
2. *Koksoon Phua, Tran Huy Dat, Jianfeng Chen* and *Louis Shue*. «Human identification using heart sound»; Institute for Infocomm Research, Singapore 2008.
3. *Nigam V., Priemer R*. Cardiac Sound Separation. University of Illinois at Chicago, 2004.
4. Biometrics: Publications http://biometrics.cse.msu.edu

ИССЛЕДОВАНИЕ АЛГОРИТМА ВСТАВКИ ВОДЯНЫХ ЗНАКОВ В СЖАТЫЙ ВИДЕОПОТОК В ФОРМАТЕ H.264

Д. О. Иванов

студент кафедры комплексной защиты информации;
denis.ivo@vu.spb.ru

А. В. Афанасьева

ассистент кафедры комплексной защиты информации;
alra@vu.spb.ru

Санкт-Петербургский государственный университет аэрокосмического приборостроения

Аннотация: В данной работе рассмотрен метод вставки цифровых знаков, адаптированный для сжатого видеопотока в формате H.264. Для метода вставки приведен выбор параметров на основе уровня безопасности и вносимых искажений. Для увеличения стойкости к атакам показано применение мажоритарного декодера.

Введение

Развитие информационных технологий привело к увеличению незаконного распространения цифровых данных. Вследствие чего любой пользователь, имея цифровую копию, может легко распространить ее через Интернет. Такая ситуация не устраивает производителей контента и возникает задача, определить, кто из пользователей является нечестным, чтобы в дальнейшем прекратить взаимодействие с ним. Примером цифровых данных, которые наиболее подвержены незаконному распространению, являются видеофильмы. Для сжатия видеофильмов высокого разрешения в настоящее время применяется стандарт ITU-T H264 [1]. Поэтому в данной работе будет обсуждаться метод вставки цифровых знаков в видеофильмы, закодированных с помощью данного стандарта.

Постановка задачи

Вставка цифровых знаков должна производиться в сжатый видеопоток, так как для определения пиратов необходимо внедрять уникальную метку в каждую копию, а кодирование видеофильма для каждого пользователя по отдельности является трудоемкой задачей. Так как копии из-за уникальности вставленных меток будут отличаться, то несколько пиратов, объединившись в коалицию и сравнивая свои копии, будут пытаться снять метку.

Поэтому уникальные метки разных пользователей будут вставляться в одни и те же позиции (общий ключ), а борьба против коалиции будет происходить с помощью специальных кодов [2].

Из выше сказанного можно описать следующую модель атакующего:

- Атаки на один кадр (пример: фильтры пост-обработки, уменьшение размеров кадра).
- Атаки на кадровую структуру (пример: удаление, добавление кадров).
- Коалиционные атаки;

Такая модель атакующего предполагает наличие следующих этапов алгоритма:

1. Построение антикоалиционных последовательностей
2. Вставка последовательности в сжатый поток
3. Извлечение последовательности из сжатого потока
4. Определение участников коалиции по извлеченной последовательности

В данной работе более детально будут рассмотрены 2 и 3 этапы. Стоит также отметить, что 1 и 4 этап решают задачу борьбы с коалиционными атаками.

Метод вставки

На рис. 1 рассмотрена типовая схема кодера H.264 для выявления возможных мест для вставки цифрового водяного знака.

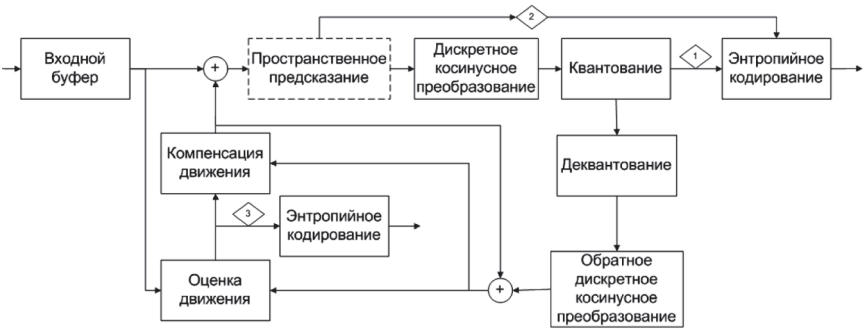


Рис. 1. Типовая схема кодера H.264

При вставке цифровых знаков в сжатый поток нет возможности использовать трудоемкие операции (например, косинусное преобразование). Поэтому возможны следующие методы для вставки цифровых знаков:

- Изменение векторов движения [3]
- Изменение режима пространственного предсказания [4]
- Изменение квантованных частотных коэффициентов [5]

Первый подход является наименее стойким. В частности, после атаки перекодированием с пересчетом векторов движения метку невозможно детектировать. Метод изменения частотных коэффициентов является более гибким с точки зрения минимизации вносимых искажений по сравнению с методом изменения режима пространственного предсказания. Поэтому его и будем использовать.

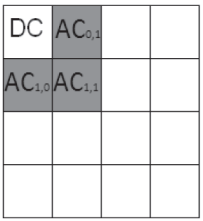


Рис. 2. Блок 4×4 после ДКП

Рассмотрим блок изображения 4×4 (см. рис. 2) после применения дискретного косинусного преобразования и квантования.

Низкочастотные коэффициенты содержат большую часть энергии, поэтому изменять нужно именно их. Алгоритм вставки основан на следующем правиле:

$$AC_{i,j} = AC_{i,j} \pm level,$$

где *level* — параметр алгоритма, а знак говорит о передаваемом бите. Стоит отметить, что для детектирования понадобится исходный блок. Такой алгоритм имеет две степени свободы: глубина продавливания *level* и количество изменяемых коэффициентов. Фиксируя две эти степени свободы необходимо определить информационную емкость одного кадра. Для этого построим график (см. рис. 3) *psnr* (Peak Signal to Noise Ratio) от количества изменяемых блоков и проведем некоторую условную границу качества. После чего на кривых можно легко выбрать точку, лежащую на одной из кривых выше условной границы. Эта точка определит значения для 3-х параметров алгоритма: глубины продавливания коэффициентов, количества изменяемых коэффициентов в блоке, количества изменяемых блоков.

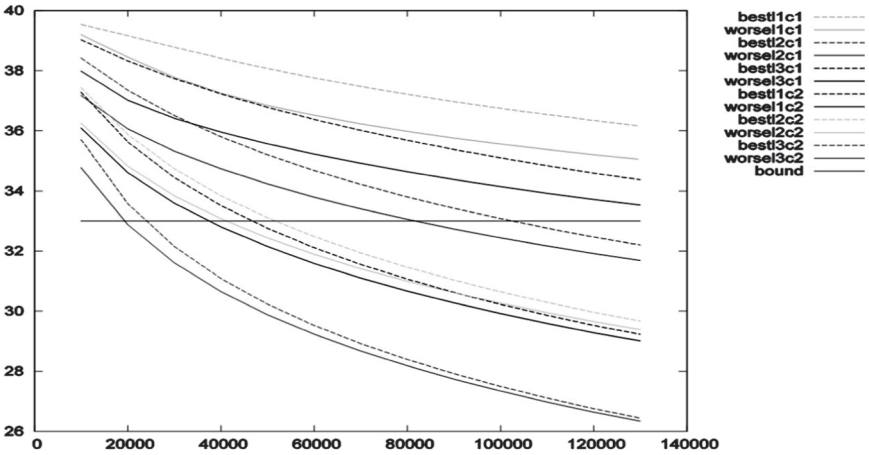


Рис. 3. *PSNR* (количество блоков)

Но такой метод выбора параметров не достатоно хорош, так как если мы будем изменять нетекстурный блок, то искажения будут сильно выделяться (см. рис. 4), что является недопустимым.

Рис. 4. Пример заметных искажений

Для того чтобы вносимые искажения нельзя было заметить, необходимо в качестве блоков для вставки выбирать текстурные блоки. Такие блоки должны содержать в себе мало ненулевых коэффициентов. Выбор таких блоков можно сделать на этапе предвычислений. Если выбирать блоки, в которых больше половины ненулевых коэффициентов, то в кадре оказывается в среднем 800 подходящих блоков для вставки. Отметим, что метод выбора блоков открыт (т.е. известен атакующему), а ключом является позиция изменяемого коеффициента внутри блока. Задачей атакующего будет угадывание знака изменения и позиции коэффициента, так как если он изменит все, то произойдет значительная потеря в качестве.

Использование мажоритарного декодера

В видеофильмы заложена огромная избыточность, которой можно воспользоваться. Будем дублировать вставляемую метку каждые несколько кадров (т. е. применим код с повторением). Тогда при извлечении последовательности понадобится мажоритарный декодер. Общая схема приведена на рис. 5.

После мажоритарного декодера необходимо по извлеченной последовательности определить участников коалиции. Отметим, что такой подход позволит бороться с различными атаками на один кадр, т. к. вероятность одинаково изменить один и тот же символ кодового слова в различных кадрах уменьшается с уменьшением скорости кода.

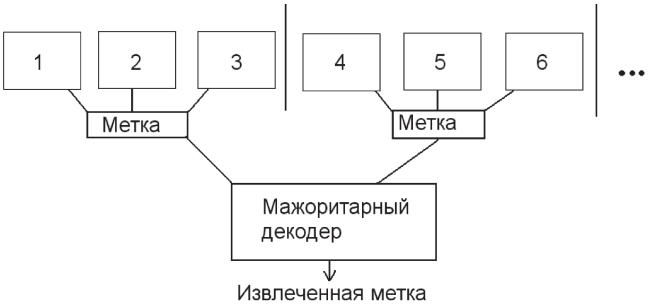


Рис. 5. Повторение метки

Таким образом, можно произвести следующие расчеты. Если мы воспользуемся антикоаллиционными кодами [2] с параметрами 1 000 000 пользователей и максимально допустимым размером коалиции 10, то такой код будет иметь длину 210 000 бит. Также выше уже объяснялось, что информационная емкость в среднем 800 бит на кадр. Из чего следует, что для вставки последовательности необходимо 263 кадра.

С другой стороны предположим, что вставить цифровой знак необходимо в фильм длительностью 10 минут и кадровой скоростью 24 кадра в секунду. Такой фильм содержит 14 400 кадров.

Из этих подсчетов следует, что при таких параметрах фильма метка может быть повторена 54 раза (т.е. может быть использован код с повторением со скоростью 1/54).

В результате исследований различных атак были выбраны следующие параметры: глубина продавливания 2, количество изменяемых коэффициентов 1. Ниже представлены результаты различных атак с применением и без применения кода с повторением. Отметим, что был использован код со скоростью 1/31. Вставка метки велась только в I-кадры при QP = 28.

Таблица 1

Сравнение воздействия атак

Атака	Средний % ошибки по кадрам	Средний % ошибки после применения мажоритарного декодера ($R = 1/31$)
Уменьшение размеров кадра до 1280 × 720 с перекодированием	21.3	0.03
Фильтр Blur с перекодированием	19.9	0.03
Перекодирование с использованием *B*-кадров с *QP* 33–34	14.1	0
Перекодирование с $QP = 30$	5.3	0
Перекодирование с $QP = 32$	11.8	0

Заключение

В данной работе подобраны параметры для алгоритма вставки на основе параметров безопасности и визуальных искажений. Помимо этого рассмотрены различные атаки на одиночный кадр. Для повышения устойчивости к таким атакам предложено использовать мажоритарный декодер, который дает хороший выигрыш. В дальнейшем необходимо решить ещё 2 задачи. Первой задачей является минимизация распространения ошибки в кадры, которые предсказны по кадрам со вставкой. Возможными решениями являются: вставка только в блоки, которые не распространяют ошибку или компенсация вносимой ошибки. Вторая задача: борьба с атаками на кадровую структуру. Цель данной атаки состоит в нарушение синхронизации между кадрами из исходного фильма и фильма с меткой. Возможное решение данной задачи — использование слепых методов [6, 7], т.е. методов, которые не нуждаются в исходном фильме для извлечения метки.

Литература

1. H.264/mpeg-4 part 10 avc iso/iec 14496–10, coding of audio-visual objects — part 10. Advanced video coding. 2003.
2. *G. Tardos* Optimal probabilistic fingerprint codes // Proc. Acm symposium on theory of computing, New York, NY, USA, 2003. Pp.116–125.
3. *G. Qui* et all. A hybrid watermarking scheme for H.264/AVC video // Proceeding ICPR 04 Proceedings of the Pattern Recognition, 17th International conference on (ICPR»04). Vol. 4.
4. *D. Zou* and *J. A. Bloom*. H.264/AVC Stream Replacement Technique for Video Watermarking // IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2008.
5. *P.-C. Su, M.-L. Li, I.-F. Chen*. A content-adaptive digital watermarking scheme in H.264/AVC Compressed videos // Proc. International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 2008.
6. *S. Saryazdi, M. Demehri*. A blind dct domain digital watermarking // Sciences of Electronic, Technologies of Information and Telecommunicatios, 2005.
7. *A. Mansouri, A. M. Aznaveh, F. Torkamani-Azar, F. Kurugollu* A low complexity video watermarking in H.264 compressed domain // IEEE Transactions on Information Forensics and Security. Vol. 5. No. 4. December 2010.

К ВОПРОСУ О ВЫБОРЕ ОПТИМАЛЬНОЙ СКОРОСТИ КОДА ДЛЯ КОДИРОВАНИЯ НА ТРАНСПОРТНОМ УРОВНЕ

Д. А. Маличенко

асс. кафедры комплексной защиты информации;
dml@vu.spb.ru

Санкт-Петербургский государственный университет аэрокосмического приборостроения

Аннотация: В данной статье описывается методика выбора скорости кода для транспортного кодирования, при которой достигается наименьшая задержка сообщения при использовании транспортного кодирования. Выбор скорости кода выполняется для модели сети Клейнрока, производится подбор закона распределения задержки пакета.

Введение

Транспортное кодирование позволяет уменьшить задержку сообщения при передаче по вычислительной сети с коммутацией пакетов [1]. Значение задержки сообщения зависит от параметров сети, от количества информационных символов и от скорости кода, выбранного для транспортного кодирования. Скорость кода, минимизирующая значение задержки при использовании транспортного кодирования, будем называть оптимальной. В работе [2] приводится расчет оптимальной скорости кода для экспоненциальной модели сети. В данной статье описывается методика выбора оптимальной скорости кода для модели сети Клейнрока [3]. Модель сети Клейнрока представляет собой менее идеализированную модель, чем экспоненциальная [2].

Если известен закон распределения задержки пакета, то среднюю задержку сообщения можно вычислить с помощью техники порядковых статистик [4]. В данной работе подбирается распределение для задержки пакетов для модели сети Клейнрока. Далее выполняется расчет средней задержки сообщения для разных скоростей кодирования. Из всех значений скоростей кода выбирается то значение, которое дает наименьшую задержку сообщения.

Модель сети Клейнрока

Рассмотрим модель Клейнрока для вычислительной сети с коммутацией пакетов. Топология сети задается с помощью графа. Узлы представляют собой вычислительные устройства, которые передают сообщения другим вычислительным устройствам. Узлы передают как свои сообщения, так и сообщения от других узлов. Интервалы между появлениями новых сообщений

распределены по экспоненциальному закону распределения. Интенсивность, с которой появляются новые сообщения на узлах задается матрицей интенсивности, в которой на i-й строке и j-м столбце стоит интенсивность появления сообщения от узла i к узлу j. Ребра на графе представляют собой каналы. Время передачи по каналу распределено по экспоненциальному закону с параметром μC, где $1/\mu$ — это длина пакета, а C — емкость канала. Каналы считаются бесшумными и абсолютно надежными. Так как в сети осуществляется коммутация пакетов, то все сообщения перед отправкой разбиваются на равные части, которые называются пакетами. Затем пакеты по очереди отправляются адресату. Передача пакетов осуществляется путем последовательной пересылки от узла к узлу согласно заданному маршруту. Маршруты задаются в таблице маршрутизации, при построении которой для каждой пары источник-адресат выбирается маршрут с минимальной метрикой. В данной работе в качестве метрики выбрано количество промежуточных узлов.

Транспортное кодирование

При использовании транспортного кодирования для передачи сообщений по сети каждый пакет рассматривается как отдельный символ некоторого (n, k) кода. Перед отправкой сообщение кодируется. В результате по сети передается не k пакетов, а $n = k/R$, где R скорость кода. Если использовать код с максимальным достижимым расстоянием (МДР-код), то по любым k пакетам можно восстановить сообщение. В данном способе передачи, кроме процедуры кодирования используется многопутевая маршрутизация. Пакеты одного и того же сообщения отправляются по разным маршрутам. Приняв первые k пакетов, по которым можно декодировать сообщение, адресат восстанавливает сообщение и сообщение считается принятым. Таким образом, для приема сообщения нет необходимости дожидаться прихода всех n пакетов. Так как пакеты передаются по разным маршрутам, то при таком способе передачи, возможно снижение времени доставки сообщения. Обозначим через через $T_{k:n}$ — среднее время приема первых k пакетов из n. При использовании МДР кода $T_{k:n}$ будет равно средней задержке сообщения.

Выбор скорости кода

Если известен закон распределения задержки пакета $F(x)$, то среднюю задержку сообщения $T_{k:n}$ можно вычислить с помощью техники порядковых статистик по следующей известной формуле [4]:

$$T_{k:n} = nC_{n-1}^{k-1} \int_{-\infty}^{\infty} x[F(x)]^{k-1}[1-F(x)]^{n-k}\, dF(x), \qquad (1)$$

где $n = k/R$. Выполняя перебор по R можно найти такое значение скорости кода, при котором задержка сообщения $T_{k:n}$ будет минимальной.

В качестве закона распределения для задержки пакета для вычислений по формуле (1) предлагается использовать распределение Эрланга. Строго говоря, задержка пакета не распределена по закону Эрланга. По этому закону распределена задержка одного пакета, который передается по отдельному маршруту, когда никакие другие пакеты не передаются по каналам этого маршрута. В этом случае задержка каждого пакета будет равна сумме одного и того же числа случайных величин распределенных по одному и тому же закону распределения. В ситуации, когда передается много пакетов, которые идут вперемешку, задержки пакетов будут подчиняться закону распределения Эрланга с разными параметрами. Перемешивание этих распределений даст другой закон распределения задержек. Распределение Эрланга в данной работе используется в качестве усредненного распределения для задержки сообщения в сети. Оно выбрано по следующим причинам:

1. Распределение Эрланга дает неплохой результат для задачи определения оптимальной скорости кода.
2. С увеличением средней длины пути распределение задержки пакетов будет стремиться к нормальному закону, а распределение Эрланга также стремится к нормальному с ростом порядка.

Закон распределения Эрланга имеет два параметра:

$$F_r(x,\alpha) = 1 - e^{-\alpha x}\sum_{i=0}^{r-1}\frac{(\alpha x)^i}{i!},$$

где r порядок Эрланга, а α параметр экспоненциального распределения. Для вычислений по формуле (1) необходимо знать оба параметра. Порядок распределения предлагается определять с помощью программы имитационного моделирования модели Клейнрока. Для этого необходимо один раз запустить моделирующую программу для сбора статистики. Второй параметр связан с математическим ожиданием задержки пакета формулой:

$$M[X] = \frac{r}{\alpha}.$$

Для средней задержки однопакетных сообщений известна следующая формула Клейнрока:

$$T_p = \sum_{i=1}^{M}\frac{\lambda_i}{\gamma}\frac{1}{\mu C - \lambda_i},$$

где M — количество каналов, λ_i — суммарная интенсивность трафика проходящего по i-му каналу, а γ — суммарная интенсивность появления новых сообщений в сети. В данной статье рассматривается многопакетная передача сообщений, поэтому среднюю задержку сообщений предлагается вычислять следующим образом:

$$M[X]=\left(\sum_{i=0}^{[n/nr]-1} T_p+\frac{1}{\mu}i\right)\frac{1}{[n/nr]-1}, \qquad (2)$$

где *nr* — это среднее число маршрутов между парой узлов. Идея вычисления (2) состоит в том, чтобы учесть эффект от попадания в буфер канала пачки из нескольких пакетов. Этот эффект вызван тем, что время между вбросами пакетов одного сообщения в сеть считается нулевым. Размер пачки равен [*n*/*nr*]. Формула Клейнрока хорошо подходит только для первого пакета. Задержка следующих за первым пакетов будет выше.

Используя формулу (2) можно получить второй параметр распределения Эрланга. Затем по формуле (1) можно вычислить среднюю задержку сообщения. Выполняя перебор по скорости кода можно найти наименьшую задержку. Для оценки работы методики выбора скорости кода выполнено построение графиков зависимости средней задержки сообщения от скорости кода, на которых представлены для сравнения две кривые. Одна кривая получена вычислением как было описано выше, а вторая кривая получена с помощью программы имитационного моделирования. В качестве примера ниже приведены два таких графика для *k*-связных сетей.

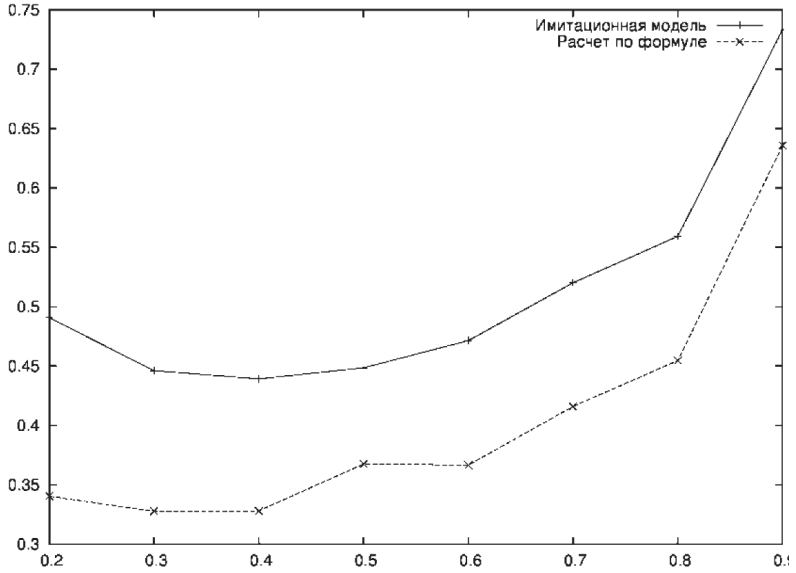


Рис**. 1.** График зависимости скорости кода от средней задержки сообщения для 9-связной сети

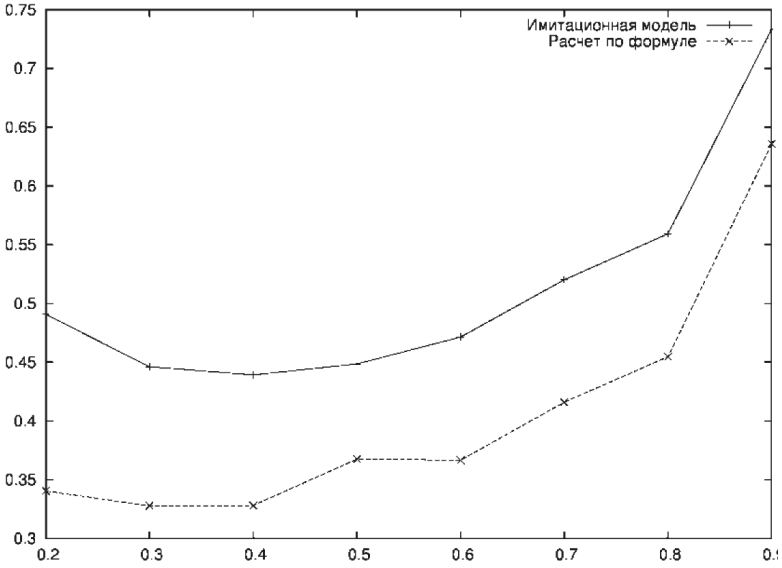


Рис. **2.** График зависимости скорости кода от средней задержки сообщения для 7-связной сети

На графиках видно, что обе кривые по форме повторяют друг друга и описанная выше методика выбора скорости кода для выбранных сетей позволяет получить то значение скорости кода, при котором задержка сообщения минимальна. У данной методики есть границы применения. Описанная методика проверялась на k-связных сетях и на нескольких прямоугольных решетках. Иногда данная задержка сообщения при скорости кода вычисленной по предложенной методике не является минимальной, но находится поблизости от минимальной. Чем выше связность сети, тем более точные результаты можно получить.

Заключение

В работе описана методика выбора оптимальной скорости кода, при котором минимизируется задержка сообщений при передаче по сети с помощью транспортного кодирования. Представлены графики зависимости средней задержки сообщения от скорости кода для случая имитационного моделирования сети Клейнрока и для вычисления, предложенного в данной статье. Для выбранных k-связных сетей данная методика позволила выбрать оптимальное значение скорости кода. В дальнейшем планируется лучше изучить использование методики для разных типов топологий и различных значений загрузки сети.

Литература

1. *Kabatiansky G., Krouk E., Semenov E*. Error Correcting Coding and Security for Data Networks: Analysis of the Superchannel Concept / G. Kabatiansky, E. Krouk, E. Semenov, P.: John Wiley & Sons, Ltd, 2005. 290 p.
2. Крук *Е.* А. Комбинаторное декодирование линейных блоковых кодов / Е. А. Крук; ГУАП. СПб., 2007. 238 с.
3. Клейнрок *Л.* Коммуникационные сети: Стохастические потоки и задержки сообщений / Пер. с англ. Л. Клейнрок. М.: Наука, 1970. 255с.
4. Дэйвид *Г.* Порядковые статистики / Пер. с англ. / Г. Дэйвид. М.: Наука, 1979. 336 с.

РАЗРАБОТКА МЕТОДОВ КАНАЛЬНО-СЕТЕВОГО КОДИРОВАНИЯ

А. С. Мишина

студент кафедры распределенных вычислений и компьютерных сетей;
mishinalina@gmail.com

Санкт-Петербургский государственный университет аэрокосмического приборостроения

Аннотация: В данной статье представлено построение канально-сетевого кода и декодера для него, на основе подпространственных кодов Кеттера — Кшишанга — Силвы и полярных кодов. Получена теоретическая оценка корректирующей способности построенных кодов. Произведено сравнение практических и теоретических зависимостей вероятности ошибки от отношения сигнал/шум для данного канально-сетевого кода.

Введение

Сетевое кодирование — это раздел теории информации, который изучает вопрос оптимизации передачи данных по сети с использованием техник изменения пакетов данных на промежуточных узлах. Сеть связи может быть представлена в виде конечного направленного графа, в котором узлы соединены ребрами. Ребра соответствуют каналам связи и помечены числами, означающими пропускную способность соответствующего канала. Примером такой сети является сеть Интернет. Задача состоит в том, чтобы передать данные от узла-источника к заранее заданному набору узлов-стоков. Необходимо установить можно ли решить эту задачу при заданных ограничениях на каналы связи и, если это возможно, то как это сделать наиболее эффективно.

В работе [1] было показано, что методами «сетевого кодирования» можно получить лучшие результаты, чем традиционными методами. Это тема представляет большой интерес и широко исследуется в настоящее время.

Построение канально-сетевого кода

Рассматривается модель сети, в которой ошибки контролируются только на узлах-источниках и узлах-стоках с помощью внешнего кода, — модель сквозного кодирования. При этом внутренние узлы создают случайную линейную комбинацию входящих пакетов, это позволяет потенциально увеличить пропускную способность и сделать работу сети более устойчивой. Каждое ребро представляет собой канал передачи информации, в котором происходит зашумление информации аддитивным белым гауссовским шу-

мом (АБГШ). Каждому ребру присваивается некоторое действительное значение отношения сигнал/шум.

Для построения внешнего сетевого кода были использованы подпространственные коды Кеттера — Кшишанга — Силвы. Кодовые слова кода в ранговой метрике являются матрицами $n \times m$ и ранговое расстояние между двумя кодовыми словами принимается равным рангу разности соответствующих им матриц. Кеттер, Кшишанг и Силва предложили специальную конструкцию подпространственных кодов, в которой матрица ранга, не превосходящего n, преобразуется в матрицу фиксированного ранга путем добавления единичной матрицы — построения лифтинговой конструкции. В частности, близкий к оптимальному подпространственный код может быть получен из оптимального ранго-метрического кода. Для декодирования данного кода был использован обобщенный алгоритм Габидулина [2], исправляющий ошибки, стирания и отклонения.

В качестве внутреннего канального кода было предложено использовать полярные коды. На выходе из узла новый пакет, представляющий собой линейную комбинацию, кодируется полярным (n_p, k_p, d_p)-кодом. Поступающая последовательность бит разбивается на блоки длины k_p. Если входная последовательность не кратна k_p, то оставшиеся биты дополняются последовательностей нулей до длины k_p. Далее закодированные полярным кодом n_p бит передаются по ребру. Во время передачи по ребру информация зашумляется, путем суммирования с АБГШ. При достижении следующего узла, полученные данные декодируются полярным кодом, при этом производится попытка исправить ошибки, внесенные в ходе передачи по ребру. Затем полученные блоки длины k_p восстанавливаются в n сетевых пакетов, где n — длина сетевого кода.

Таким образом, передача по сети осуществляется до тех пор, пока все пакеты не будут переданы — в источнике нет сообщений и все ребра пустые. После завершения передачи производится декодирование на сетевом уровне. На каждом из стоков полученные пакеты информации декодируются обобщенным ранго-метрическим кодом Габидулина. При этом выполняется попытка исправить ошибки канального уровня и стирания и отклонения, возникающие при формировании линейных комбинаций.

Оценка корректирующей способности

Для построенного канального-сетевого кода была проведена оценка корректирующей способности кода. Рассматривается сеть с одинаковыми потоками f в каждом ребре. Коэффициент связанности сети равен 3.

Построенный код исправляет ошибки, стирания и отклонения. В статье [2] показано, что сетевой код Габидулина исправляет $2\varepsilon + \mu + \delta < d$, где ε — количество ошибок, μ — количество стираний, δ — количество отклонений и d — ранговое расстояние кода.

Корректирующая способность канального (полярного) (n_p, k_p, d_p) — кода оценивается по формуле обобщенной верхней границы:

$$P_p \le \sum_{i=1}^{2^m} A_i Q\left(\sqrt{\frac{2E_b}{N_0} R_c i}\right),$$

где $\frac{E_b}{N_0} = \frac{E_s}{N_0 R_c}$ — отношение сигнал/шум на бит; $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-t^2/2} dt =$ $= \frac{1}{2} erfc\left(\frac{x}{\sqrt{2}}\right)$, где *erfc* — дополнительная функция ошибок; R_c — скорость кода и $(A_0, ..., A_2^m)$ – весовой спектр кода Рида — Маллера, т. е. A_i равно числу слов веса i в коде.

В каждом ребре декодирование одного пакета полярным кодом происходит ν раз, где $\nu = (n + N) \log_2(q)/k_p$, (q, n, N) — параметры кода Габидулина. Таким образом, вероятность возникновения одной ошибки при передаче по ребру равна $P_{pe} = 1 - (1 - P_p)^\nu$.

Можно считать, что каждая ошибка, возникающая на канальном уровне, приводит к одной ранговой ошибке. Тогда вероятность возникновения t ошибок в сети

$$P[\varepsilon = t] = (1 - P_{pe})^{(fn_e - t)} P_{pe}^t,$$

где n_e — количество ребер в сети.

В свою очередь, каждая ошибка приводит к появлению максимум одного отклонения. Для оценки числа отклонений можно воспользоваться следующей формулой $\delta = \max\{\min\{\varepsilon, 3f - n\}, 0\}$, где ε — количество ошибок, значение потока умножается на 3, так как коэффициент связанности графа равен 3.

Для оценки количества стираний необходимо оценить ранг матрицы Y, которая приходит на сток. Если матрица имеет полный ранг, то стираний не происходит; если ранг равен $n - 1$, то произошло одно стирание и т. д. Так как на узлах в сети формируются линейные комбинации со случайными коэффициентами из поля $\mathbb{F}_q$, то матрицу можно считать произвольной. Для оценки вероятности того, что матрица размера $n \times n$ имеет полный ранг, может быть использована следующая формула:

$$P(\mu = 0) = P(\text{rank} = n) = \prod_{i=1}^{n} (1 - q^{-i}).$$

При больших значениях q можно считать, что вероятность появления одного стирания равна $1 - P(\mu = 0)$. Вероятностью появления большего числа стираний можно пренебречь.

Для того, чтобы вычислить вероятность успешного декодирования канально-сетевым кодом необходимо просуммировать найденные вероятности с учетом выражения $2\varepsilon + \mu + \delta < d$.

На рисунке 1 представлены графики зависимости вероятности ошибки декодирования некоторых канально-сетевых кодов на основе кода Габидулина и полярных кодов от отношения сигнал/шум с помощью описанного метода, полученные на практике и теоретическая оценка (сплошная линия).

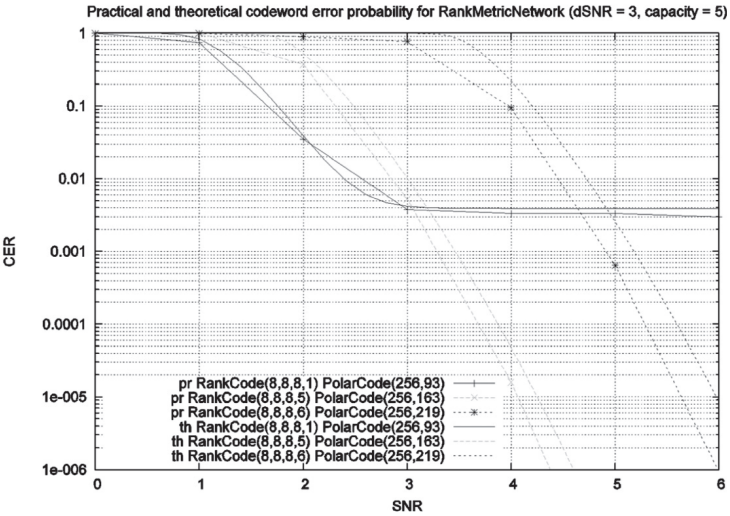


Рис. 1. Вероятность ошибки декодирования канально-сетевым кодом

Из графика на рисунке 1 видно, что при малых отношениях сигнал/шум (ОСШ) лучшие результаты показывает канально-сетевой код, корректирующая способность полярного кода у которого выше. При малых ОСШ большее влияние оказывает сетевой ранговый код. Для кода с параметрами RankCode (8,8,8,1) PolarCode (256,93) для значений ОСШ больших 2,5 дБ зависимость становится линейной, это объясняется тем, что полярный код способен исправить все возникающие ошибки и, следовательно, отклонений не возникает, в то время как сетевой код с ранговым расстоянием равным 1 не исправляет ни одного стирания, следовательно, ошибка декодирования возникает при внесении хотя бы одного стирания.

Таким образом, лучшая корректирующая способность достигается, когда скорость полярного кода меньше, а скорость сетевого, соответственно, больше.

Полученные на практики графики подтверждают состоятельность теоретической оценки. Они показывают несколько лучшие результаты, так как для теоретической оценки была использована верхняя граница.

Заключение

В данном обзоре представлены принципы случайного сетевого кодирования, которое позволяет улучшить пропускную способность сети. Описаны методы сетевого кодирования с помощью подпространственных кодов Кеттера — Кшишанг — Силвы, которые построены на основе ранговых кодов Габидулина и в которых применяется обобщенный алгоритм декодирования ранговых кодов. Предложен метод построения канально-сетевого кода на основе обобщенного алгоритма декодирования Габидулина и полярных кодов. Проведена теоретическая оценка корректирующей способности построенного кода и доказана ее состоятельность, путем сравнения с практически полученными результатами.

Литература

1. *R. Ahlswede, N. Cai, S.-Y. R. Li*, and *R. W. Yeung*. Network information flow. IEEE Transactions On Information Theory, 46 (6), July 2000.
2. *D. Silva, F. R. Kschischang*, and *R. Koetter*. A rank-metric approach to error control in random network coding. IEEE Transactions On Information Theory, 54 (9), September 2008.

ИССЛЕДОВАНИЕ КОДОВ, СТОЙКИХ К КОАЛИЦИОННЫМ АТАКАМ

Д. А. Рыжов

студент кафедры комплексной защиты информации;
dr@vu.spb.ru,

Е. М. Линский

к.т.н. доц. кафедры комплексной защиты информации;
evlinsky@vu.spb.ru

Санкт-Петербургский государственный университет аэрокосмического приборостроения

Аннотация: Описаны принципы построения и использования антикоалиционных кодов Тардоша, используемых для водяного знака, вставляемого в видеопоток. Рассмотрены различные виды атак и экспериментально показана нестойкость выбранного кода к шумовым атакам. Предложен метод адаптации кода для противодействия шумовым атакам.

Введение

Фильмы, а также другая видеопродукция, часто подвергаются нелицензионному распространению, когда пользователи скачивают их через многочисленным пиринговые сети или просматривают он-лайн на видеохостингах или в социальных сетях, при этом правообладатели не получают никакой прибыли.

Для борьбы с таким неконтролируемым распространением видеопродукции можно предложить метод внесения индивидуального водяного знака в каждую продаваемую копию фильма, содержащего идентификационную информацию о покупателе копии. Тогда по каждой не лицензионно распространяемой копии можно будет установить её покупателя и прекратить дальнейшее сотрудничество с ним.

Недобросовестные покупатели будут пытаться уничтожить водяной знак, поэтому он должен быть устойчив к различным видам атак. Особое внимание необходимо обратить на коалиционные атаки, при которых несколько недобросовестных покупателей по своим различным копиям создают новую производную копию.

Задача защиты от нелицензионного распространения видеопродукции

Этапы защиты от нелицензионного распространения

Использование механизма индивидуальных водяных знаков предусматривает следующие этапы защиты от нелицензионного распространения:

1. построение бинарных антикоалиционных последовательностей;
2. вставка последовательности в сжатый видеопоток;
3. извлечение последовательности из видеопотока;
4. определение участников коалиции по извлеченной последовательности.

В данной статье будут подробнее рассмотрены первый и четвертый этапы защиты.

Атаки на идентификационные последовательности

Возможные атаки на идентификационные последовательности можно разделить на следующие типы:

1. коалиционные атаки:
   - 1.1. наиболее частый символ;
   - 1.2. наименее частый символ;
   - 1.3. случайный символ;
   - 1.4. все единицы;
   - 1.5. все нули;
2. шумовые атаки;
3. смешанные атаки.

При коалиционной атаке несколько пользователей системы поэлементно сравнивают копии своих фильмов и на различающихся позициях расставляют производные значения, придерживаясь какого-то алгоритма (см. рис. 1).

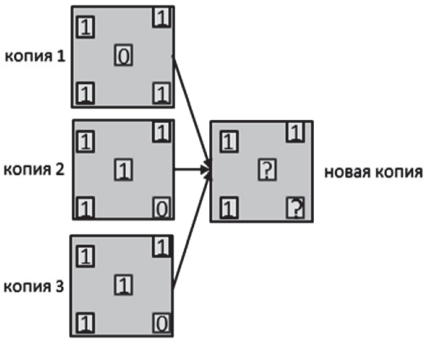


Рис. 1. Пример коалиционной атаки по нескольким копиям фильма

Коалиционная атака на идентификационные последовательности выглядит следующим образом: на одинаковых во всех последовательностях позициях символ сохраняется, а на различающихся символ выбирается по некоторому алгоритму. Например, таким алгоритмом может быть правило «наименее часто встречающийся символ» (см. табл. 1).

Таблица 1

Пример атаки на идентификационные последовательности «наименее частый символ»

Последовательность 1	1	1	1	0	0
Последовательность 2	1	0	1	0	1
Последовательность 3	1	1	0	0	0
Итоговая последовательность	1	0	0	0	1

При шумовой атаке на идентификационные последовательности символ в каждой позиции меняется на противоположный с вероятностью, соответствующей уровню шума.

При смешанной атаке используются как коалиционная атака, так и шумовая атака.

Антикоалиционные коды

Существует несколько антикоалиционных кодов, стойкость которых к атакам с заданным размером коалиции доказана (см табл. 2). Основными параметрами таких кодов являются: n — количество пользователей в системе, c — предполагаемый максимальный размер коалиции, ε — параметр безопасности системы.

Таблица 2

Сравнение длин антикоалиционных кодов для $n = 1\,000\,000$ и $\varepsilon = 10$

Метод	Длина	Длина ($c = 16$)	Длина ($c = 8$)
Boneh-Shaw [1]	$L = 2c \ln\left(\frac{2n}{\varepsilon}\right)$ $m = 32c^4 \ln\left(\frac{2n}{\varepsilon}\right) \ln\left(\frac{8cL}{\varepsilon}\right)$	8 Гб	512 Мб
Tardos [2]	$m = 100c^2 \ln\left(\frac{2n}{\varepsilon}\right)$	2 Мб	512 Кб

Поскольку коды Тардоша имеют гораздо меньшую длину последовательностей, то для исследований были выбраны именно они.

Для построения кода Тардоша необходимо:

1. исходя из параметров системы (n, c, ε) вычислить длину кода m;
2. сгенерировать m случайных вероятностей $P(i)$, распределенных по закону, представленному на рис. 2;

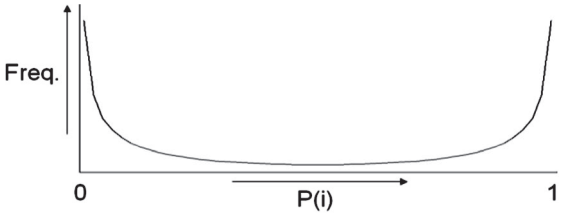


Рис. 2. Распределение вероятностей $P(i)$

3. сгенерировать случайную матрицу $S(m \times n)$, где $S(i, j) = 1$ с вероятностью $P(i)$ (см. рис. 3);

Особенностью такой матрицы является то, что в одних её строках много единиц и мало нулей, а в других — мало единиц и много нулей. Это происходит из-за заданного распределения вероятностей единиц в строке (см. рис. 2).

Рис. 3. Матрица S $(m \times n)$

4. идентификационная последовательность i-го пользователя — i-й столбец матрицы S.

Алгоритм поиска участников устроен следующим образом. По некоторой извлеченной последовательности Z необходимо установить, какие из пользователей принимали участие в построении данной последовательности.

Для этого для каждого пользователя системы при помощи матрицы S и вектора вероятностей P вычисляется коэффициент виновности по формуле:

$$A_j = \sum_{i=1}^{m} U(Z(i), S(i, j), P(i)),$$

где

$$U(1, 1, p) = \sqrt{(1-p)/p}, \quad U(1, 0, p) = -\sqrt{p/(1-p)},$$
$$U(0, 0, p) = \sqrt{p/(1-p)}, \quad U(0, 1, p) = -\sqrt{(1-p)/p}.$$

Каждый пользователь, коэффициент виновности которого превышает порог обвинения $treshold = 20c \log\ (n/\varepsilon)$, считается виновным, т. е. принимавшим участие в создании последовательности Z.

Принцип поиска участников коалиции основывается на поиске редких символов в последовательности. Согласно распределению вероятностей P, на одних позициях вероятность единиц очень высока, в то время как на других — высока вероятность нулей. Таким образом если на некоторой позиции в последовательности Z встретился редкий символ, значит те пользователи, у которых в их последовательностях так же встречается этот редкий символ, получат значительную прибавку в коэффициент виновности, и, наоборот, если у некоторого пользователя в некоторой позиции был редкий символ, а в извлеченной последовательности встретился частый символ, то этот пользователь уменьшит свой коэффициент виновности.

Воздействие шума на идентификационные последовательности кода Тардоша

Коды Тардоша — антикоалиционные коды, но среди возможных атак присутствуют шумовые атаки, поэтому была промоделирована атака шумом на одну случайную последовательность Тардоша и построен график вероятности того, что в результате работы алгоритма поиска виновных, будет обвинен невиновный пользователь при различных уровнях шума (см. рис. 4).

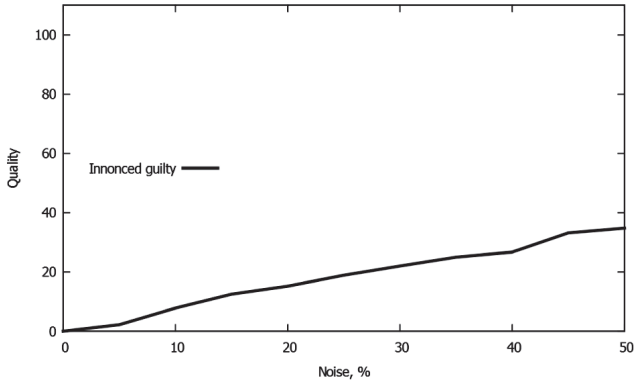


Рис. 4. Влияние шума на обнаружение невиновных пользователей

Результаты показывают, что даже с небольшим процентом шума, вероятность обвинения невиновного пользователя существенна.

Для того чтобы данные коды можно было использовать для реальной системы, нужно уменьшить влияние шума на обвинение невиновных участников.

Для решения этой задачи был введен новый символ — стирание. Алгоритм извлечения метки из фильма для каждой позиции сравнивает значение

контрольной переменной с эталонным. Если значение разности положительно — извлекается символ 1, иначе — 0. При введении порога стираний, всякое значение модуля разности контрольной переменной и эталонного значения, не превышающее порога, извлекается как стирание. При подсчете коэффициентов виновности пользователей в позициях, в которых стоит символ стирания, изменения коэффициента виновности не производится, т.е. такое слагаемое пропускается.

Было промоделировано воздействие различных уровней шума и стираний на извлеченную последовательность и построены графики зависимости обвинения невиновного пользователя (см. рис. 5) и обвинения виновного пользователя (см. рис. 6) в зависимости от различных уровней шума и стираний.

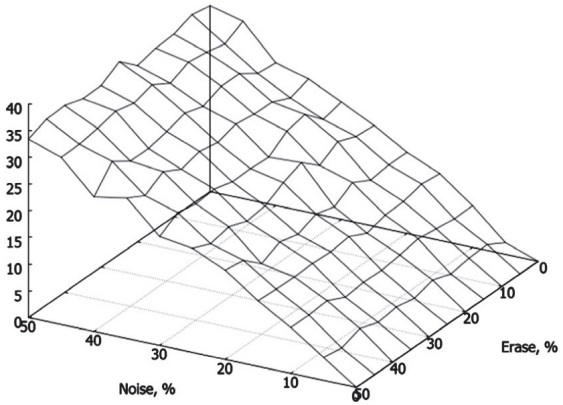


Рис. 5. Обвинение невиновного пользователя
в зависимости от различных уровней шума и стираний

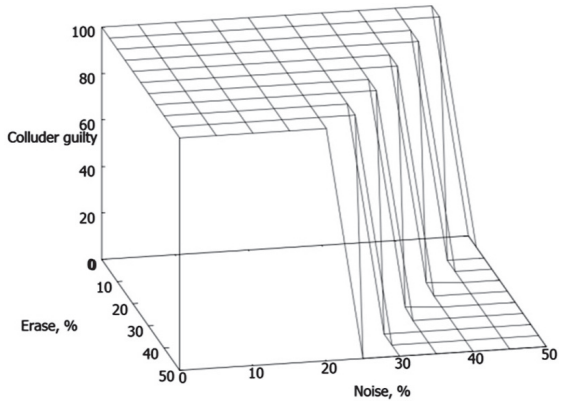


Рис. 6. Обвинение виновного пользователя
в зависимости от различных уровней шума и стираний

Результаты показывают, что с ростом уровня шума вероятность обвинения невиновного участника увеличивается, однако при росте уровня стираний — примерно сохраняется. Таким образом, если из, к примеру, 30% шума на последовательность, 10% шума удастся заменить на стирания, то вероятность обнаружения невиновного пользователя уменьшается более чем на 25%. Данная замена так же не уменьшает вероятность обвинения виновного пользователя.

Заключение

Как видно из результатов исследования использование символов стираний в кодах Тардоша дает снижение вероятности обнаружения невиновных пользователей. Алгоритм извлечения последовательности позволяет ставить степень уверенности в символе. В дальнейшем можно адаптировать алгоритм поиска виновных с учетом данных о степени уверенности.

Так же необходимо изучение возможности ускорения вычисления коэффициентов обвинения, т.к. сейчас при параметрах 1000000 пользователей и 10 пользователей в предполагаемой коалиции вычисление коэффициентов обвинения будет занимать порядка 8 часов на обычном компьютере.

Можно подвести следующие итоги:

- коды Тардоша позволяют успешно бороться с коалициями пользователей, однако они совершенно не стойки к шумовым атакам;
- применение стираний позволило повысить стойкость системы к шумовым атакам.

Литература

1. *D. Boneh, J. Shaw.* Collusion-Secure Fingerprinting for Digital Data // IEEE Transactions on Information Theory. Vol. 44. No. 5. Pp. 1897–1905. Sep 1998.
2. *G. Tardos.* Optimal probabilistic fingerprint codes // Proc. Acm symposium on theory of computing. New York, 2003. Pp.116–125.

СПИСОЧНЫЙ ДЕКОДЕР ОБОБЩЕННЫХ КАСКАДНЫХ КОДОВ С ВНУТРЕННИМИ ПОЛЯРНЫМИ КОДАМИ

П. К. Семенов

аспирант кафедры распределенных вычислений и компьютерных систем;
spk@dcn.ftk.spbstu.ru

Научный руководитель: ***Трифонов П. В***

Санкт-Петербургский государственный университет аэрокосмического приборостроения

Аннотация: В работе рассматривается идея списочного декодирования обобщенных каскадных кодов с внутренними полярными кодами. Представленный декодер такой конструкции позволяет получить вероятность ошибки декодирования, меньшую на 0.1 Дб по сравнению с LDPC-кодом из стандарта WiMax IEEE 802.16e.

Введение

В работе [2] были предложены конструкция полярных кодов на основе ядра Арикана, асимптотически достигающая пропускной способности симметричного канала без памяти с двоичным входом, и алгоритм декодирования. Но при практически значимых длинах (~1000) полярные коды показывают значительно большую вероятность ошибки декодирования по сравнению с традиционными кодами. В [4] было показано, что использование внешних кодов позволяет улучшить корректирующую способность полярных кодов. Списочное декодирование для конструкции обобщенных каскадных кодов с внутренними полярными кодами позволяет улучшить их корректирующую способность.

Обобщенные каскадные и полярные коды

Обобщенные каскадные коды [1, 3] основаны на семействе внешних (ограничимся случаем линейных внешних кодов) (N, K_i, D_i) кодов A_i над $GF(2^{m_i}), 1 \le i \le l,$ и на семействе таких вложенных внутренних (n, k_i, d_i) кодов B_j над GF (2), что $k_j = \sum_{i=j}^{l} m_i,\ 1 \le j \le l$. Кодирование данных проводится следующим образом:

1. Данные кодируются внешними кодами: пусть ($c_{1,1}$, ..., $c_{1,n}$,), ..., ($c_{l,1}$, ..., $c_{l,n}$,) — полученные при этом кодовые слова.

2. Для всех $j = 1 \ldots N$ символы $c_{i,j} \in GF(2^{m_i})$, $1 \le i \le l$ преобразуются в кортежи длин m_i с использованием некоторых фиксированных базисов $GF(2^{m_i})$. Эти кортежи кодируются с (n, k_1, d_1) внутренним кодом.

Если все используемые коды A_i и B_j являются линейными, то множество получаемых таким образом векторов образует линейный блоковый $\left(nN, \sum_{i=1}^{l} K_i m_i, \ge \min(D_1 d_1, \ldots, D_l d_l) \right)$-код.

Рассмотрим симметричный канал без памяти с двоичным входом с функцией плотности вероятности выходных символов $W(y|u)$. Согласно [2], он может быть преобразован в векторный канал $W^N(y_0^{N-1} | B_N G^{\otimes s} u_0^{N-1})$ с выходом $y_0^{N-1} = (y_0\ y_1 \ldots y_{N-1})$, где $G = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $G^{\otimes s}$ — ее s-ая Кронекеровскую степень, $N = 2^s$, а B_N — это $N \times N$-матрица такой перестановки для N-битной последовательности, что номер новой позиции -ого бита получается как обратная запись представленного в виде -битной последовательности числа i. Например, $2 \equiv (1\ 0\ 0) \to (0\ 0\ 1) \equiv 1, \quad 3 \equiv (0\ 1\ 1) \to (1\ 1\ 0) \equiv 6$. Векторный канал $W^N\left(y_0^{N-1} | B_N G^{\otimes s} u_0^{N-1}\right)$ может быть декомпозирован на эквивалентные подканалы $W_N^{(i)}\left(y_0^{N-1}, u_0^{i-1} | u_i\right)$, $0 \le i < N$ двоичных каналов, обладающего следующим свойством: при $N \to \infty$ относительная доля $W_N^{(i)}$ таких, что их пропускная способность стремится к 1, стремится к пропускной способности исходного канала. Таким образом, при $N \to \infty$ каналы $W_N^{(i)}$ будут либо абсолютно без шума, либо абсолютно ненадежными. Поэтому информационные символы u_i, передаваемые по каналам плохого качества, можно считать всегда фиксированными («замороженными»). И если сопоставить i-ую строку матрицы $G^{\otimes s}$ каналу $W_N^{(i)}\left(y_0^{N-1}, u_0^{i-1} | u_i\right)$, то r наиболее надежным каналам соответствует линейный код длины 2^s и размерности r, называемый полярным кодом.

Стоит отметить, чтобы передавать символ u_i через $W_N^{(i)}\left(y_0^{N-1}, u_0^{i-1} | u_i\right)$ необходимо достоверное знание предшествующих ему символов u_0^{i-1}. Эта особенность отражается и на последовательном декодере [2] полярных кодов: оценка символа u_i будет ошибочной, если среди u_0^{i-1} есть хотя бы один неверный символ.

Списочное декодирование обобщенных каскадных кодов с внутренними полярными кодами

Чтобы преодолеть основной недостаток последовательного декодера [2] необходимо ввести дерево вариантов, хранящее значения всех предшествующих символов. На рис. 1, *а* и 1, *б* представлено дерево всех возможных

вариантов для полярного (4,3)-кода. Для того, чтобы ограничить рост такого дерева необходимо на каждой стадии ранжировать текущие ветви и отсекать соответствующие наименее надежным вариантам (рис. 1, *б*, где ненадежные ветви обозначены прерывистой линией). Для выбора ветвей предлагается использовать списочные декодеры внешних кодов.

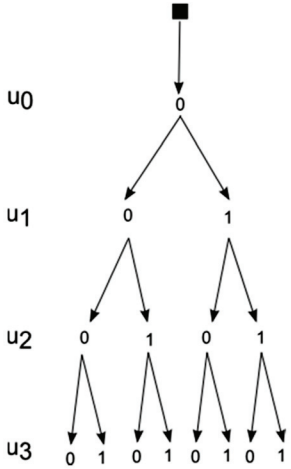


Рис. 1, *а*. Идеальное дерево вариантов для полярного (4,3)-кода

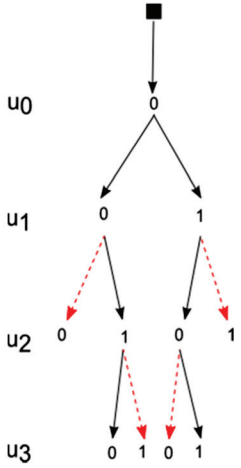


Рис. 1, *б*. Выбор только двух ветвей на каждой стадии для полярного (4,3)-кода

На основе получаемого списка можно построить дерево решений внешних кодов, с помощью которого будут корректироваться деревья вариантов/состояний декодеров внутренних полярных кодов, пример которого представлен на рис. 1, *б*. Для определенности, если речь идет о внутренних кодах, то будет использоваться название «дерево вариантов», а если о внешних — то название «дерево решений».

На рис. 2 приведен пример дерева решений внешних кодов: в его узлах расположены возможные кодовые слова, все вершины одного уровня соответствует списку кодовых слов внешнего кода A_i, полученному соответственно на i-ой стадии декодирования обобщенного каскадного кода. Все внешние коды являются кодами с одной проверкой на четность. Проекции дерева соответствуют состояниям декодеров внутренних кодов. Отсечение наименее надежных элементов списка приводит к удалению в дереве решений каких-то ветвей. Чтобы при этом сделать обновление состояний декодеров внутренних кодов, необходимо хранить ссылки на ветви (на рис. 2 они обозначены в фигурных скобках): тогда обновление деревьев вариантов соответствует простому обновлению списков ссылок (см. рис. 3).

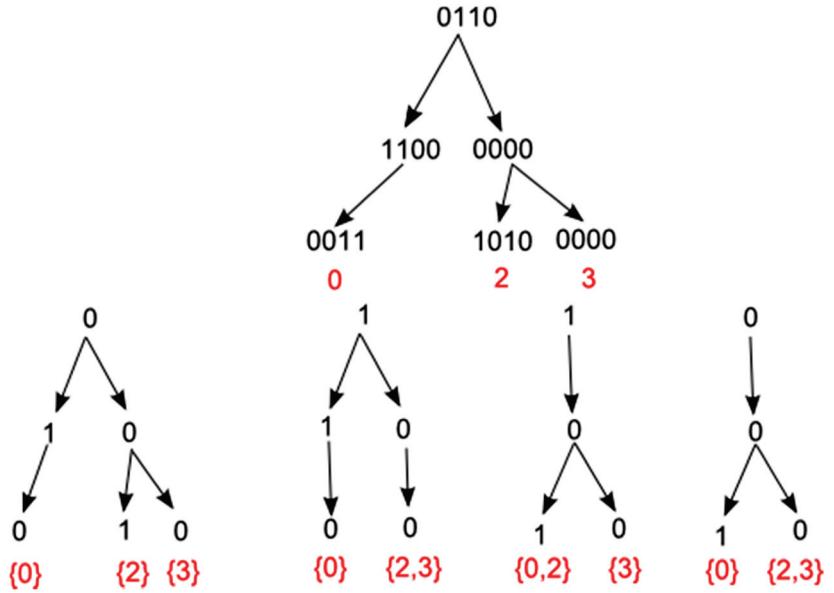


Рис. 2. Дерево решений и его проекции

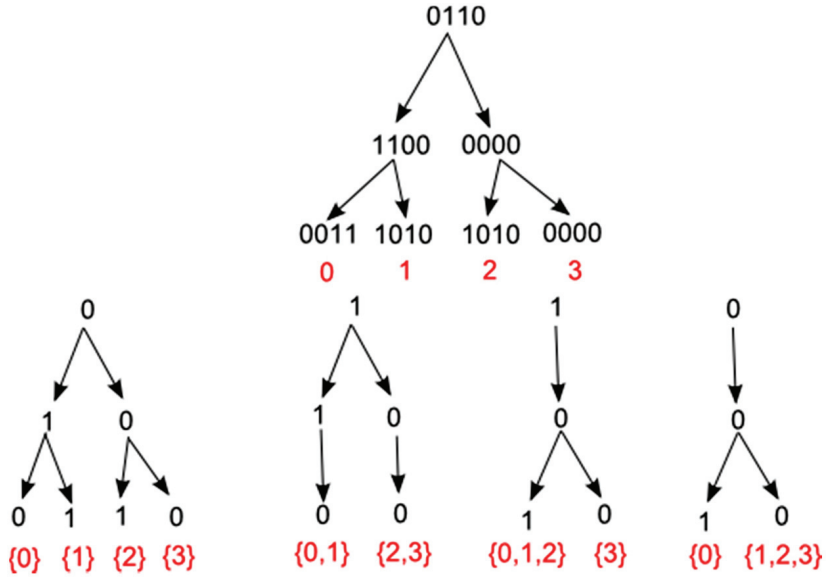


Рис. 3. Удаление ветви из дерево решений и обновление его проекции

Заключение

В статье был представлен метод декодирования обобщенных каскадных кодов с внутренними полярными кодами, основанный на списочном декодировании компонентных кодов. Имитационное моделирование показало, что (1016,508) — код, полученный из 8 внешних БЧХ-кодов длины 127 и минимальными расстояниями 0, 30, 43, 8, 47, 9, 12, 3 и внутреннего полярного кода порядка 3, и продекодированный представленным декодером с размером списка $L = 1, 2, 4$ достигает меньшей на 0.1 Дб вероятности ошибки, чем низкоплотностный (1024,512) — код из станадарта WiMax IEEE 802.16e (см. рис. 4).

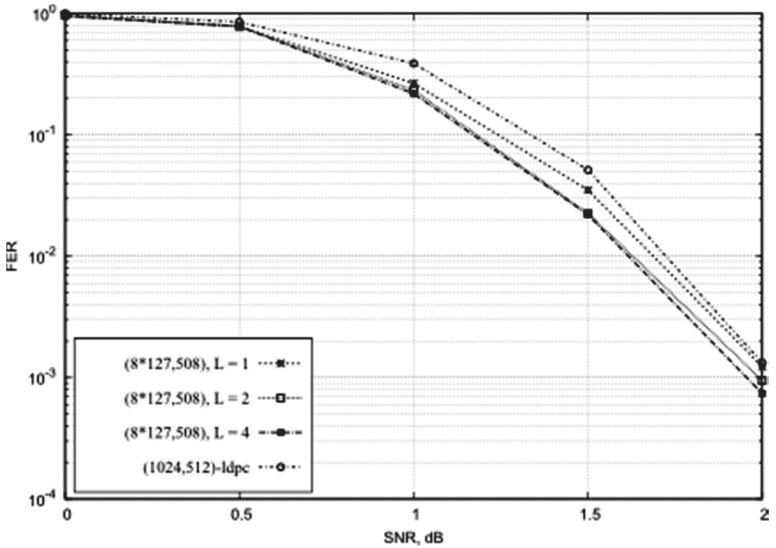


Рис. 4. Вероятность ошибки декодирования в зависимости от отношения «сигнал — шум» (Дб)

Литература

1. *Зиновьев В.* Обобщенные каскадные коды // Проблемы передачи информации. 1976. С. 5–15.
2. *Arikan E.* Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels // IEEE Transactions on Information Theory. 2009. № 7 (55). P. 3051–3073.
3. *Bossert M.* Channel coding for telecommunications // Wiley. 1999. 512 p.
4. *Trifonov P., Semenov P.* Generalized Concatenated Codes Based on Polar codes // In Proceedings of IEEE International Symposium on Wireless Communication Systems. 2011. P. 442–446.

ИДЕНТИФИКАЦИЯ ПРОЦЕССОВ ДЛЯ БОРЬБЫ С ВРЕДОНОСНЫМИ ПРОГРАММАМИ

А. Р. Ханов

awengar@gmail.com

Руководитель: ***М. В. Баклановский***

Санкт-Петербургский государственный университет

Аннотация: В данной работе представлен метод, позволяющий идентифицировать процессы программ. Этот метод может быть применен для обнаружения вредоносных кодов, а сама задача идентификации программ является очень важной для информационной безопасности.

Борьба с вредоносными программами является важнейшей задачей информационной безопасности. Для ее решения создаются различные методы выявления угроз и идентификации вредоносных программ.

Первопричиной всех этих проблем является невозможность идентификации программы или ее процесса и проверки функциональной эквивалентности двух программ. Для операционной системы нет различий между различными потоками процессов, в то время как сами эти потоки могут быть вообще не свойственны данной программе.

Методы нахождения угроз в процессе исполнения программы по ее поведению уже существуют. Так в [1] приводится метод поиска угроз по цепочкам API-вызовов, которые производятся при работе программы. Нашей задачей является идентификация программ по их внешним взаимодействиям. Но мы пытаемся научиться различать произвольные потоки процессов. В общем случае мы не ограничивается только лишь идентификацией, а пытаемся получить как можно больше различной информации из собранных данных и возможность идентификации — это лишь часть задачи, которая нами решается.

В качестве информации о работе программы были взяты списки номеров системных вызовов для каждого потока. Их извлечение происходило по методу, описанному в [2]. Эти цепочки были изучены, после чего было сделано предположение о том, что в них существуют последовательности вызовов, наиболее характерные для этого процесса. Эти последовательности названы нами термами. Термы должны содержаться во множестве дубликатов, то есть последовательностей, повторяющихся несколько раз. Был разработан алгоритм, позволяющий с высокой скоростью находить самые длинные цепочки вызовов, содержащиеся в последовательности не менее фиксированного числа раз. Период их встречаемости дает нам приблизительную длину

последовательности, на которой можно ожидать этот терм. Теперь для того, чтобы опознать поток программы, нам просто нужно искать эти термы в последовательности вызовов ее потоков.

Проведенные тесты показали, что мы можем различать программы, основываясь лишь на информации, полученной из главного потока. Так были взяты 12 программ для ОС Windows 7, из них были выделены самые длинные потоки. Для них было получено множество термов. Затем были записаны вызовы тех же программ на других экземплярах их процессов. Для формирования списка термов были взяты последовательности номеров вызовов размером порядка миллиона. Для опознания самых длинных потоков программ по этим термам хватило последовательности длиной в несколько десятков тысяч вызовов.

Теперь мы, например, можем записать особенности работы всех процессов в системе и выявлять у них нестандартное поведение, либо опознавать те процессы, которые еще не были нами изучены, а также выявить, какой из уже изученных программ является наблюдаемый нами процесс.

Данный алгоритм направлен на выделение особенностей работы потоков, и не приспособлен для того, чтобы обнаруживать функциональное соответствие двух программ. Он может помочь выявлять исполнение вредоносных кодов, но и ему можно пытаться противодействовать, применяя различные преобразования по вставке фиктивных вызовов или перестановке вызовов, которые будут нарушать характерные цепочки у потоков. Также можно написать программу, написанную специально для имитации работы другой программы с целью обойти наш алгоритм идентификации.

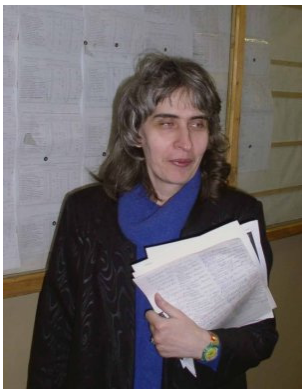
Чтобы оценить способность алгоритма к идентификации был проведен следующий небольшой тест. Были написаны две программы на Perl, которые скачивают страницы из сети и сохраняют их в файл. Они делали это в разном порядке: первая сохраняла страницу после каждой загрузки, а вторая проводила все сохранения после всех загрузок. Но, тем не менее, большая часть термов одной программы обнаруживалась в другой. Такие же программы были написаны и на Python, результат был таким же, как и в предыдущем случае. По набору термов Perl и Python можно идентифицировать, термы одного процесса встречались в другом в малом количестве. Это говорит о том, что термы скорее предназначены для обнаружения общих участков кода, а не общего поведения, что и позволяет с высокой точностью различать программы. Эти объекты представляют большой интерес, и в дальнейшем они будут нами изучены.

В исследовании, посвященном получению различной информации из списков системных вызовов программ, нами уже было сделано несколько шагов, касающихся оценке активности потоков, выявлению схожих участков и описанной выше идентификации процессов. Все это может иметь широкое применение не только в информационной безопасности, но и в других задачах, связанных с анализом работы программ.

Литература

1. *M. Ahmadi, A. Sami, H. Rahimi, B. Yadegari.* Iterative System Call Patterns Blow the Malware Cover // Security for The Next Generation. 2011.
2. *Одеров Р. С., Тенсин Е. Д.* Способы размещения своего кода в ядре ОС Microsoft Windows Server 2008 // Сборник трудов межвузовской научно-практической конференции «Актуальные проблемы организации и технологии защиты информации», СПб., 2011. С. 100–102.

Сплайновые приближения и вопросы распараллеливания в OpenMP



Бурова
Ирина Герасимовна

д.ф.-м.н.
профессор кафедры параллельных алгоритмов СПбГУ

О РАСПАРАЛЛЕЛИВАНИИ ПОСТРОЕНИЯ СРЕДНЕВАДРАТИЧЕСКИХ ПРИБЛИЖЕНИЙ НЕПОЛИНОМИАЛЬНЫМИ СПЛАЙНАМИ МИНИМАЛЬНОГО ДЕФЕКТА

И. Г. Бурова

профессор кафедры вычислительной математики;
BurovaIG@mail.ru

М. П. Винник

студент математико-механического факультета;
saburosakai393@mail.ru

Санкт-Петербургский государственный университет

Аннотация: В данной работе рассматриваеся применение неполиномиальных сплайнов минимального дефекта к задаче построения среднеквадратического приближения. Обсуждаются результаты распараллеливания метода релаксации для решения возникающей системы уравнений.

1. О построении B-сплайнов второй степени. Пусть m — натуральное число, на промежутке $[a, b]$ задана равномерная сетка узлов $a = x_0 < \ldots < x_j < \ldots < x_m = b$, с шагом $h = (b - a)/m$, так что $x_j = a + jh$, $j = 0, 1, \ldots, m$. Базисные сплайны $\omega_j(x)$, удовлетворяющие условиям: $\operatorname{supp}\omega_j = [x_{j-1}, x_{j+2}]$, $\omega_j \in C^1[a, b]$, будем строить решая систему уравнений (см.[1]) относительно $\omega_k(x)$ с параметрами c_{01}, c_{10}, c_{02}, c_{20}, c_{22}:

$$\begin{cases} \omega_{j-1}(x) + \omega_j(x) + \omega_{j+1}(x) = 1, \\ x_{j-1}\omega_{j-1}(x) + x_j\omega_j(x) + x_{j+1}\omega_{j+1}(x) = c_{10}x + c_{01}, \\ x_{j-1}^2\omega_{j-1}(x) + x_j^2\omega_j(x) + x_{j+1}^2\omega_{j+1}(x) = c_{02}x + c_{20}x^2 + c_{22}. \end{cases}$$

При значениях параметров $c_{02} = -h$, $c_{01} = -h/2$, $c_{10} = c_{20} = 1$, $c_{22} = h^2/2$ на промежутке $[x_j, x_{j+1}]$ находим

$$\omega_j(x) = -\frac{(x - jh)^2}{h^2} + \frac{1}{2h}\big(h(2j - 1) + 2x\big),$$

$$\omega_{j-1}(x) = \frac{(x - jh - h)^2}{2h^2},\ \omega_{j+1}(x) = \frac{(x - jh)^2}{2h^2}.$$

Нетрудно получить формулу базисного сплайна

$$\omega_j(x)=\begin{cases}\dfrac{1}{2h^2}(x+h-jh)^2, & x\in[x_{j-1},\,x_j];\\ -\dfrac{(x-jh)^2}{h^2}+\dfrac{1}{2h}\big(h(2j-1)+2x\big), & x\in[x_j,\,x_{j+1}];\\ \dfrac{1}{2h^2}(x-2h-jh)^2, & x\in[x_{j+1},\,x_{j+2}];\\ 0, & x\notin[x_{j-1},x_{j+2}].\end{cases}$$

2. О построении тригонометрических сплайнов минимального дефекта. Для построения тригонометрических базисных сплайнов, таких что: $\operatorname{supp}\omega_j(x)=[x_{j-1},\,x_{j+2}]$ рассмотрим три системы уравнений с параметрами c_{10}, c_{01}, c_{20}, c_{02}.

Таким образом, при $x\in[x_j,\,x_{j+1}]$

$$\omega_{j-1}(x)+\omega_j(x)+\omega_{j+1}(x)=1,$$

$$\sin(x_{j-1})\omega_{j-1}(x)+\sin(x_j)\omega_j(x)+\sin(x_{j+1})\omega_{j+1}(x)=c_{10}\sin(x)+c_{01}\cos(x),$$

$$\cos(x_{j-1})\omega_{j-1}(x)+\cos(x_j)\omega_j(x)+\cos(x_{j+1})\omega_{j+1}(x)=c_{02}\sin(x)+c_{20}\cos(x),$$

на промежутке $[x_{j-1},\,x_j]$

$$\omega_{j-2}(x)+\omega_{j-1}(x)+\omega_j(x)=1,$$

$$\cos(x_{j-2})\omega_{j-2}(x)+\cos(x_{j-1})\omega_{j-1}(x)+\cos(x_j)\omega_j(x)=c_{02}\sin(x)+c_{20}\cos(x),$$

и если $[x_{j+1},\,x_{j+2}]$ —

$$\omega_j(x)+\omega_{j+1}(x)+\omega_{j+2}(x)=1,$$

$$\sin(x_j)\omega_j(x)+\sin(x_{j+1})\omega_{j+1}(x)+\sin(x_{j+2})\omega_{j+2}(x)=c_{10}\sin(x)+c_{01}\cos(x),$$

$$\cos(x_j)\omega_j(x)+\cos(x_{j+1})\omega_{j+1}(x)+\cos(x_{j+2})\omega_{j+2}(x)=c_{02}\sin(x)+c_{20}\cos(x).$$

Значения параметров c_{01}, c_{10}, c_{02}, c_{20} находим из условия: $\omega_j\in C^1(R^1)$. Таким образом, получаем $c_{02}=-c_{01}=\cos(h/2)\sin(h/2)$, $c_{10}=c_{20}=\cos^2(h/2)$, и далее на промежутке $[x_j,x_{j+1}]$ находим

$$\omega_j(x)=\frac{\cos(h)-\cos(x-jh-h/2)\cos(h/2)}{\cos(h)-1},\ \omega_{j-1}(x)=\frac{\cos(x-jh-h)-1}{2(\cos(h)-1)},$$

$$\omega_{j+1}(x)=\frac{\cos(x-jh)-1}{2(\cos(h)-1)}.$$

Объединяя формулы $\omega_j(x)$, найденные на трех соседних промежутках с одинаковым номером j, получаем:

$$\omega_j(x)=\begin{cases}\dfrac{\cos(x-jh+h)-1}{2(\cos(h)-1)}, & x\in[x_{j-1},\,x_j],\\ \dfrac{\cos(h)-\cos(x-jh-h/2)\cos(h/2)}{\cos(h)-1}, & x\in[x_j,\,x_{j+1}],\\ \dfrac{\cos(x-jh-2h)-1}{2(\cos(h)-1)}, & x\in[x_{j+1},\,x_{j+2}].\end{cases}$$

3. Вычисление элементов матрицы. Как известно, задача среднеквадратического приближения приводит к задаче решения системы линейных уравнений с матрицей Грама. В нашем случае структура матрицы Грама M при $n=4$ будет иметь следующий вид:

$$\begin{pmatrix}(\omega_0,\omega_0) & (\omega_1,\omega_0) & (\omega_2,\omega_0) & 0 & 0 & 0\\ (\omega_0,\omega_1) & (\omega_1,\omega_1) & (\omega_2,\omega_1) & (\omega_3,\omega_1) & 0 & 0\\ (\omega_0,\omega_2) & (\omega_1,\omega_2) & (\omega_2,\omega_2) & (\omega_3,\omega_2) & (\omega_4,\omega_2) & 0\\ 0 & (\omega_1,\omega_3) & (\omega_2,\omega_3) & (\omega_3,\omega_3) & (\omega_4,\omega_3) & (\omega_5,\omega_3)\\ 0 & 0 & (\omega_2,\omega_4) & (\omega_3,\omega_4) & (\omega_4,\omega_4) & (\omega_5,\omega_4)\\ 0 & 0 & 0 & (\omega_3,\omega_5) & (\omega_4,\omega_5) & (\omega_5,\omega_5)\end{pmatrix}$$

Вычисляя скалярные произведения, в случае полиномиальных B-сплайнов, получаем следующие значения $(\omega_j,\omega_j)=(11/20)h$, $(\omega_1,\omega_1)=(1/20)\,h$, $(\omega_2,\omega_2)=(1/2)\,h$, $(\omega_N,\omega_N)=(1/2)\,h$, $(\omega_{N+1},\omega_{N+1})=(1/20)\,h$, $(\omega_j,\omega_{j+1})=(13/60)\,h$, $(\omega_1,\omega_2)=(13/120)\,h$, $(\omega_1,\omega_3)=(1/120)\,h$, $(\omega_{N-1},\omega_{N+1})=(1/120)\,h$, $(\omega_N,\omega_{N+1})=$ $=(13/120)\,h$.

Элементы вектора правой части F таковы: $(f,\omega_j)=\int\limits_{x_{j-1}}^{x_{j+2}} f(x)\,\omega_j(x)$.

Вычисляя скалярные произведения, в случае тригонометрических сплайнов минимального дефекта, получаем следующее значение

$$(\omega_j,\omega_j)=(1/4)(\sin(h)\cos(h)-4\sin(h)+3h)/$$
$$/(\cos(h)^2-2\cos(h)+1)+(1/4)(-7\sin(h)\cos(h)+\sin(h)+4h\cos(h)^2+$$
$$+\cos(h)h+h)/(\cos(h)^2-2\cos(h)+1),$$

$$(\omega_1,\omega_1)=(1/8)(\sin(h)\cos(h)-4\sin(h)+3h)/(\cos(h)^2-2\cos(h)+1),$$

$$(\omega_2,\omega_2)=(1/4)(-7\sin(h)\cos(h)+\sin(h)+4h\cos(h)^2+\cos(h)h+h)/$$
$$/(\cos(h)^2-2\cos(h)+1)+(1/8)(\sin(h)\cos(h)-4\sin(h)+3h)/$$
$$/(\cos(h)^2-2\cos(h)+1),$$

$$(\omega_N, \omega_N) = (1/4)(-7\sin(h)\cos(h) + \sin(h) + 4h\cos(h)^2 + \cos(h)h + h)/$$
$$/(\cos(h)^2 - 2\cos(h) + 1) + (1/8)(\sin(h)\cos(h) - 4\sin(h) + 3h)/$$
$$/(\cos(h)^2 - 2\cos(h) + 1),$$

$$(\omega_{N+1}, \omega_{N+1}) = (1/8)(\sin(h)\cos(h) - 4\sin(h) + 3h)/(\cos(h)^2 - 2\cos(h) + 1),$$

$$(\omega_j, \omega_{j+1}) = -(1/8)(-3\sin(h)\cos(h) - 3\sin(h) + 5\cos(h)h + h)/$$
$$/(\cos(h)^2 - 2\cos(h) + 1) - (1/12)(6\cos(h)h + 4h^3 + 6h - 3\sin(h)h^2 - 12\sin(h))/$$
$$/h^2/(\cos(h) - 1),$$

$$(\omega_1, \omega_2) = -(1/12)(6\cos(h)h + 4h^3 + 6h - 3\sin(h)h^2 - 12\sin(h))/h^2/(\cos(h) - 1),$$

$$(\omega_1, \omega_3) = (1/8)(\cos(h)h - 3\sin(h) + 2h)/(\cos(h)^2 - 2\cos(h) + 1),$$

$$(\omega_{N-1}, \omega_{N+1}) = (1/8)(\cos(h)h - 3\sin(h) + 2h)/(\cos(h)^2 - 2\cos(h) + 1),$$

$$(\omega_N, \omega_{N+1}) = -(1/8)(-3\sin(h)\cos(h) - 3\sin(h) + 5\cos(h)h + h)/$$
$$/(\cos(h)^2 - 2\cos(h) + 1).$$

Элементы вектора правой части F таковы: $(f, \omega_j) = \int_{x_{j-1}}^{x_{j+2}} f(x)\omega_j(x)$.

4. Решение системы методом верхней релаксации. Систему уравнений приводим к виду $x = Hx + g$. Расчетная формула имеет вид

$$x_i^{k+1} = x_i^k + w\sum_{j=1}^{i-1} x_j^{k+1}h_{ij} - w\sum_{j=1}^{n} x_j^{k-1}h_{ij}.$$

Для хранения вектора приближения x будем использовать один массив, оба суммирования можно проводить в одном цикле. Оптимальный выбор параметра релаксации w может существенно уменьшить число необходимых итераций. Пусть $\rho(H)$ — спектральный радиус матрицы H, тогда оптимальное значение w вычисляется по формуле:

$$w = \frac{1}{1 + \sqrt{1 - \rho^2(H)}}.$$

Для поиска $\rho(H)$ используем степенной метод.

5. Распараллеливание. Исходя из условия завершения метода, логично использование цикла while, так как количество итераций нам неизвестно. OpenMP не позволяет распараллеливать цикл while, поэтому параллельная секция помещена в тело цикла. Построение матрицы распараллелено с помощью #omp sections.

6. Построение матрицы. Матрица системы имеет пятидиагональный вид, элементы диагоналей строятся независимо друг от друга, поэтому этот процесс оптимизирован с помощью директивы #pragma omp parallel sections. Разделяемыми переменными являются переменная матрицы M и переменная, в которой хранится шаг h. Секций четыре: заполнение главной диагонали, заполнение двух смежных ей, заполнение следующих двух, и заполнение крайних элементов в углах матриц.

```
omp_set_num_threads(4);
  #pragma omp parallel sections shared(M,h) \
    private (j)
  {
      #pragma omp section
      {
          for (j=3;j<=n;j++)
          {
              M[j][j]=41/20.0*h;
          }
      }
      #pragma omp section
      {
          for (j=2;j<=n;j++)
          {
             M[j][j+1]=-47/60.0*h;
             M[j+1][j]=M[j][j+1];
          }
      }
      #pragma omp section
      {
          for (j=1;j<=n;j++)
          {
             M[j+2][j]=31/120.0*h;
             M[j][j+2]=M[j+2][j];
          }
      }
      #pragma omp section
      {
          M[1][1]=31/20.0*h;
          M[2][2]=2*h;
          M[n+1][n+1]=h/2;
          M[n+2][n+2]=1/20.0*h;
          M[2][1]=-77/120.0*h;
          M[1][2]=M[2][1];
          M[n+2][n+1]=-17/120.0*h;
          M[n+1][n+2]=M[n+2][n+1];
      }
  }
```

7. Построение вектора свободных членов. При вычислении вектора свободных членов наиболее трудоемкой частью является расчет элементов $f_j, j = 3, ..., n + 1$. Он производится с помощью цикла for, распараллеленного с помощью parallel for. Решение СЛАУ Алгоритм решения подразумевает использование *while*.

Распараллеливание происходит внутри тела цикла. Параллельная секция представляет собой цикл *for*, оптимизированный с помощью директивы #pragma omp parallel, который зависит от размера матрицы. На данный момент реализованы два варианта распараллеливания. В первом варианте распараллелен полностью весь цикл — производится некоторое количество итераций (цикл идет до параметра *lp*, далее происходит проверка условия продолжения *while*.

```
do
  {
    k2++;
 #pragma omp parallel for shared(M,x,omega,F,n) \
       private(i,j,dtmp2,x1)
     for (int i5=0;i5<lp;i5++)
     {
       k++;
         //запоминаем предыдущую итерацию
         for (int i4=1;i4<=n+2;i4++)
         {
           x1[i4]=x[i4];
         }
         for (i=1;i<=n+2;i++)
         {
             dtmp2=0;
             for (j=1;j<=n+2;j++)
             {
                 dtmp2=dtmp2+M[i][j]*x[j];//x_k+1[i]
             }
             x[i]=omega*F[i]+omega*dtmp2-x1[i]*(omega-1);
         }
         dtemp1=abs(x[1]-x1[1]);
         for (int i3=2;i3<=n+2;i3++)
         {
             dtemp2=abs(x[i3]-x1[i3]);//чтобы не считать 2 раза
             if (dtemp1<dtemp2)
                 dtemp1=dtemp2;
         }
     }
  }
  while(dtemp1>pogr);
```

Во втором варианте распараллелен только двойной цикл расчета приближения.

```
do
  {
      k++;
      for (int i4=1;i4<=n+2;i4++)
      {
        x1[i4]=x[i4];
      }
      #pragma omp parallel for shared(M_elem,x,omega,x1,F,n) private(j,dtmp2)
      for (int i=1;i<=n+2;i++)
      {
          dtmp2=0;
          for (j=1;j<=n+2;j++)
          {
              dtmp2=dtmp2+get_elem(i,j,M_elem)/*M[i][j]*/*x[j];//x_k+1[i]
          }
          x[i]=omega*F[i]+omega*dtmp2-x1[i]*(omega-1);
      }
      dtemp1=abs(x[1]-x1[1]);
      for (int i3=2;i3<=n+2;i3++)
      {
          dtemp2=abs(x[i3]-x1[i3]);
          if (dtemp1<dtemp2)
              dtemp1=dtemp2;
      }
 }
while(dtemp1>pogr);
```

8. Результаты. Целью работы было провести оптимизацию времени вычислений. Были произведены замеры времени на пятидиагональных матрицах различного размера. Результаты приведены в таблице. Время указано в миллисекундах. В четвертом столбце таблицы даны результаты первой оптимизации: если все тело цикла *while* находится внутри цикла *for*; в пятом столбце — вторая оптимизация: распараллеливание происходит на каждой итерации (распараллеливается только вычисление следующего приближения внутри цикла *while*). Все вычисления производились на четырехпроцессорном компьютере.

Размер матрицы	Метод Зейделя	Метод в. р.	Метод в. р. опт. (1)	Метод в. р. опт. (2)
50	297	93	78	76
200	499	530	452	546
500	671	686	390	718
800	983	983	717	1046
1000	1248	1232	983	1248
5000	16832	16915	16754	16661
10000	65952	65907	73604	63710
10000	112971	113272	138653	112694

Литература

1. *Фаддеев Д. К. Фаддеева В. Н.* Вычислительные методы линейной алгебры. М.; Л., 1963. 734 с.
2. *Бурова И. Г., Евдокимова Т. О.* Приближение неполиномиальными сплайнами минимального дефекта. СПб., 2007. 47 с.

О ПОСТРОЕНИИ НЕКОТОРЫХ ИНТЕГРО-ДИФФЕРЕНЦИАЛЬНЫХ СПЛАЙНОВ

И. Г. Бурова

профессор кафедры вычислительной математики;
BurovaIG@mail.ru.

И. А. Морозов

студент 521 группы математико-механического факультета;
morozig@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В этой работе для приближения функций построены некоторые разрывные и непрерывные интегро-дифференциальные базисные сплайны.

1. Введение

При построении сплайновых аппроксимаций использование дополнительной информации, например, такой как величина интеграла по одному или нескольким соседним сеточным промежуткам от приближаемой функции повышает порядок аппроксимации. Сплайны, использующие значения производных функции в узлах сетки и значения интегралов от функции по одному или нескольким сеточным интервалам, называются интегро-диференциальными сплайнами. В работе [1] построены полиномиальные, а в статье [2] неполиномиальные интегро-дифференциальные сплайны. При использовании информации о значениях функции и ее производных в узлах сетки приходим к минимальным интерполяционным сплайнам [3].

Пусть a, b — вещественные числа, n — целое число, $n \geq 2$. Построим на $[a, b]$ упорядоченную сетку узлов $\{x_k\}$: $a < \ldots < x_{k-1} < x_k < x_{k+1} < \ldots < b$, $k = 0, 1, \ldots, n$. Обозначим $h_k = x_{k+1} - x_k$. Функция $u(x)$, $u \in C[a, b]$, задана в узлах сетки $\{x_k\}$ и известны значения интегралов $\int_{x_k}^{x_{k+1}} u(x)\,dx$.

На $[x_k, x_{k+1}]$ будем рассматривать приближения функции $u(x)$ вида:

1. $\tilde{u}(x) = u(x_k)\,\omega_k(x) + \int_{x_k}^{x_{k+1}} u(x)\,dx\,W_k(x)$,
2. $\tilde{u}(x) = u(x_k)\,\omega_k(x) + u(x_{k+1})\,\omega_{k+1}(x) + \int_{x_k}^{x_{k+1}} u(x)\,dx W_k(x)$,
3. $\tilde{u}(x) = u(x_k)\,\omega_{k,0}(x) + u'(x_k)\,\omega_{k,1}(x) + \int_{x_k}^{x_{k+1}} u(x)\,dx W_k(x)$,

где $\omega_k(x)$, $W_k(x)$ — некоторые базисные сплайны, определяемые в дальнейшем.

2. Разрывные приближения

Функцию $u(x)$ на промежутке $[x_k, x_{k+1}]$ будем приближать следующим образом:

$$\tilde{u}(x) = u(x_k)\omega_k(x) + \int_{x_k}^{x_{k+1}} u(x)\,dx W_k(x). \tag{1}$$

2.1. Полиномиальные приближения вида (1). Базисные функции ω_k, $W_k(x)$ находим из соотношения

$$\tilde{u}(x) - u(x) = 0, \;\; u = 1, x.$$

отсюда при $x \in [x_k, x_{k+1}]$ получаем систему уравнений относительно ω_k, $W_k(x)$:

$$\omega_k(x) + (x_{k+1} - x_k)W_k(x) = 1, \tag{2}$$

$$x_k\omega_k(x) + \frac{x_{k+1}^2 - x_k^2}{2}W_k(x) = x. \tag{3}$$

Нетрудно видеть, что определитель Δ этой системы равен $\Delta = (x_{k+1} - x_k)^2/2$, таким образом $\Delta \neq 0$ при $x_{k+1} \neq x_k$, а базисные функции на промежутке $[x_k, x_{k+1}]$ таковы:

$$\omega_k(x) = \frac{x_{k+1} + x_k - 2x}{x_{k+1} - x_k}, \tag{4}$$

$$W_k(x) = 2\frac{x - x_k}{(x_{k+1} - x_k)^2}. \tag{5}$$

Переходя к переменной $t \in [0,1]$, получаем $\omega_k(t) = -2t + 1$, $W_k(t) = \dfrac{2t}{h_k}$.

Из формул (4), (5) получаем оценки

$$|\omega_k(x)| \le 1, \; |W_k(x)| \le \frac{2}{x_{k+1} - x_k}. \tag{6}$$

С помощью разложения функции $u(t)$ в окрестности точки $x \in [x_k, x_{k+1}]$ по формуле Тейлора с остатком в интегральном виде и используя соотношения (2), (3), получим

$$|u - \tilde{u}| \le \left| \int_x^{x_k} \frac{u''(z)(t-z)}{2} dz \omega_k(x) \right| + \left| \int_{x_k}^{x_{k+1}} \int_x^t \frac{u''(z)(t-z)}{2} dz\,dt W_k(x) \right|.$$

С учетом оценок (6), применяя теорему о среднем и интегрируя приходим к оценке погрешности приближения

$$|u(x)-\tilde{u}(x)| \le \frac{7}{12} h_k^2 \, \|u''\|_{C_{[x_k, x_{k+1}]}}.$$

2.2. Построение разрывных экспоненциальных базисных сплайнов. Базисные функции ω_k, $W_k(x)$ на промежутке $[x_k, x_{k+1}]$ находим из соотношений

$$\tilde{u}(x) - u(x) = 0, \;\; u = 1, e^x,$$

которые приводят к системе уравнений

$$\omega_k(x) + (x_{k+1} - x_k) W_k(x)(x) = 1, \tag{7}$$

$$x_k \omega_k(x) + \int_{x_k}^{x_{k+1}} e^x dx W_k(x)(x) = e^x. \tag{8}$$

Определитель системы имеет вид $\Delta = e^{x_{k+1}} - e^{x_k} - h_k e^{x_k}$, очевидно он отличен от нуля при $x_{k+1} \neq x_k$. Базисные функции имеют вид

$$\omega_k(t) = \frac{1 - e^{h_k} + h_k e^{th_k}}{1 - e^{h_k} + h_k}, W_k(t) = \frac{1 - e^{th_k}}{1 - e^{h_k} + h_k}, t \in [0,1].$$

Методом, описанным в работе [2], с учетом (7),(8) находим

$$\tilde{u}(x) - u(x) = u(x_k)\omega_k(x) + \int_{x_k}^{x_{k+1}} u(x)\, dx W_k(x) - u(x) =$$

$$= \omega_k(x) \left(\int_x^{x_k} (u'(y) - u''(y))(1 + e^{x_k - y})\, dy \right) +$$

$$+ W_k(x) \int_{x_k}^{x_{k+1}} \left(\int_x^t (u'(y) - u''(y))(1 + e^{t-y})\, dy \right) dt.$$

Применяя теорему о среднем и интегрируя получаем оценку

$$|u - \tilde{u}| \le K h_k^2 \, \|u'' - u'\|_{C_{[x_k, x_{k+1}]}}, \; K > 0.$$

2.3. Разрывные приближения вида (3). В случае

$$\tilde{u}(x) = u(x_k)\omega_{k,0}(x) + u'(x_k)\omega_{k,1}(x) + \int_{x_k}^{x_{k+1}} u(x)\, dx W_k(x),$$

базисные функции $\omega_{k,0}, \omega_{k,1}, W_k(x)$ на промежутке $[x_k, x_{k+1}]$ определяем из соотношений

$$\tilde{u}(x) - u(x) = 0, \;\; u = 1, x, x^2,$$

и теперь система уравнений принимает вид

$$\omega_{k,0}(x)+(x_{k+1}-x_k)W_k(x)=1,$$

$$x_k\omega_{k,0}(x)+\omega_{k,1}(x)+\int_{x_k}^{x_{k+1}} x\,dx\,W_k(x)=x,$$

$$x_k^2\omega_{k,0}(x)+2x_k\omega_{k,1}(x)+\int_{x_k}^{x_{k+1}} x^2\,dx\,W_k(x)=x^2.$$

Таким образом

$$\omega_{k,0}(x)=-\frac{(3x_k^2-h_k^2-6x\,x_k+3x^2)}{h^2},$$

$$\omega_{k,1}(x)=-\frac{(-6x\,x_k-2xh_k+3x^2+3x_k^2+2x_kh_k)}{2h},$$

$$W_k(x)=\frac{3(x^2-2x\,x_k+x_k^2)}{h_k^3}.$$

Определитель этой системы равен $h_k^3/3$. Оценка погрешности принимает вид

$$|u-\tilde{u}|\le Kh_k^3\,\|u'''\|_{C[x_k,x_{k+1}]},\ K>0.$$

3. Построение непрерывных полиномиальных базисных сплайнов

В случае $\tilde{u}(x)=u(x_k)\omega_k(x)+u(x_{k+1})\omega_{k+1}(x)+\int_{x_k}^{x_{k+1}} u(x)\,dxW_k(x)$ базисные функции $\omega_k,\omega_{k+1},W_k(x)$ на промежутке $[x_k,x_{k+1}]$ определяем из соотношений

$$\tilde{u}(x)-u(x)=0,\ u=1,x,x^2,$$

приводящих к системе уравнений

$$\omega_k(x)+\omega_{k+1}(x)+(x_{k+1}-x_k)W_k(x)=1, \tag{9}$$

$$x_k\omega_k(x)+x_{k+1}\omega_{k+1}(x)+\int_{x_k}^{x_{k+1}} x\,dxW_k(x)=x, \tag{10}$$

$$x_k^2\omega_k(x)+x_{k+1}^2\omega_{k+1}(x)+\int_{x_k}^{x_{k+1}} x^2\,dxW_k(x)=x^2. \tag{11}$$

Определитель системы уравнений равен $\Delta=-h_k^4/6$. Базисные функции имеют вид

$$\omega_k(t)=-4t+3t^2+1,\ \omega_{k+1}(t)=t(-2+3t),\ W_k(t)=\frac{-6t(-1+t)}{h_k},\ t\in[0,1].$$

С помощью формулы Тейлора $u(t) = u(x) + u'(x)(t-x) + u''(x)(t-x)^2/2 + \int_x^t \frac{u'''(z)(t-z)^2}{6}\,dz$ и соотношений (9) — (11) получаем

$$|\tilde{u}(x) - u(x)| \le \left| \int_x^{x_k} \frac{u'''(z)(t-z)^2}{6} dz\omega_k(x) \right| + \left| \int_x^{x_{k+1}} \frac{u'''(z)(t-z)^2}{6} dz\omega_{k+1}(x) \right| +$$

$$+ \left| \int_{x_k}^{x_{k+1}} \int_x^t \frac{u'''(z)(t-z)^2}{6}\, dz\, dt W_k(x) \right|.$$

Отсюда вытекает оценка погрешности на промежутке $[x_k, x_{k+1}]$

$$|u - \tilde{u}| \le \frac{11}{72} h_k^3 \, \|u'''\|_{C_{[x_{k-1}, x_{k+1}]}}.$$

4. Построение непрерывных тригонометрических базисных сплайнов.

В случае $\tilde{u}(x) = u(x_k)\omega_k(x) + u(x_{k+1})\omega_{k+1}(x) + \int_{x_k}^{x_{k+1}} u(x)\,dx W_k(x)$ базисные функции $\omega_k, \omega_{k+1}, W_k(x)$ на промежутке $[x_k, x_{k+1}]$ определяем из соотношений

$$\tilde{u}(x) - u(x) = 0,\ u = 1, \sin(x), \cos(x),$$

приводящих к системе уравнений

$$\omega_k(x) + \omega_{k+1}(x) + (x_{k+1} - x_k) W_k(x) = 1, \tag{9}$$

$$\sin(x_k)\omega_k(x) + \sin(x_{k+1})\omega_{k+1}(x) + \int_{x_k}^{x_{k+1}} \sin(x) dx\, W_k(x) = \sin(x), \tag{10}$$

$$\cos(x_k)\omega_k(x) + \cos(x_{k+1})\omega_{k+1}(x) + \int_{x_k}^{x_{k+1}} \cos(x)\, dx\, W_k(x) = \cos(x). \tag{11}$$

Определитель системы уравнений равен $\Delta = -h\sin(h_k) + 2 - 2\cos(h_k)$. Базисные функции имеют вид

$$\omega_k(t) = -\frac{(-\cos(h_k) + 1 + h_k \sin(th_k - h_k) + \cos(th_k) - \cos(th_k - h_k))}{(h_k \sin(h_k) - 2 + 2\cos(h_k))},$$

$$\omega_{k+1}(t) = \frac{(-\cos(x - x_k) + \cos(-x + x_k + h_k) - h_k \sin(x - x_k) + 1 - \cos(h_k))}{(-2\cos(h_k) + 2 - (x_k + h_k)\sin(h_k) + x_k \sin(h_k))},$$

$$W_k(t) = \frac{(\sin(-x + x_k + h_k) - \sin(h_k) + \sin(x - x_k))}{(-2\cos(h_k) + 2 - (x_k + h_k)\sin(h_k) + x_k \sin(h_k))},\ t \in [0, 1].$$

С помощью метода [2] и соотношений (9) — (11) получаем оценку погрешности на промежутке $[x_k, x_{k+1}]$

$$|u-\tilde{u}| \le Kh_k^3 \, \|u'+u'''\|_{C_{[x_{k-1},x_{k+1}]}}, \; K>0.$$

5. Результаты некоторых вычислений.

В таблице, приведенной ниже, приведены значения погрешностей $|R|=|\tilde{u}(x)-u(x)|$ приближения нескольких функций на промежутке [–1, 1], полученных экспериментально при построении приближений в среде Maple 11 на равномерной сетке узлов с шагом $h=0.25$.

u	1, x	1, e^x	1, x, x^2	1, sin x, cos
$\frac{1}{1+25x^2}$	0.0544	0.0478	0.0423	0.042
$\|2x\|$	0	0.0135	0	0.0003
e^{2x}	0.2432	0.1247	0.0097	0.012
cos 2x	0.0255	0.0296	0.001	0.0007

Литература

1. *Киреев В. И., Пантелеев А. В.* Численные методы в примерах и задачах. М., 2008. 480 с.
2. *Бурова И. Г.* О моделировании неполиномиальных интегро-дифференциальных приближений. Труды СПИИРАН. Вып 4 (19). 2011. С. 176–208.
3. *Бурова И. Г., Демьянович Ю. К.* Минимальные сплайны и их приложения. Изд-во СПбГУ, 2010. 364 с.

РАСПАРАЛЛЕЛИВАНИЕ ПОСТРОЕНИЯ СРЕДНЕКВАДРАТИЧЕСКОГО ПРИБЛИЖЕНИЯ СПЛАЙНАМИ ВОСЬМОГО ПОРЯДКА АППРОКСИМАЦИИ ТРЕТЬЕЙ ВЫСОТЫ

И. Г. Бурова

профессор кафедры вычислительной математики;
BurovaIG@mail.ru

С. В. Полуянов

аспирант кафедры вычислительной математики;
sergeypoluyanov@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Рассматривается построение среднеквадратического приближения с помощью базисных сплайнов восьмого порядка аппроксимации третьей высоты. Приводятся результаты численных экспериментов по ускорению вычислений при распараллеливании алгоритма.

Введение

Пусть на конечном промежутке $[a, b]$, где a и b — вещественные числа, задана некоторая функция $f \in L_2([a,b])$. Возьмём целое число n и построим на этом промежутке равномерную сетку узлов $\{x_j\}$ с шагом $h = \dfrac{(b-a)}{n}$:

$$a = x_0 < \ldots < x_{j-1} < x_j < x_{j+1} < \ldots < x_n = b.$$

Как известно, задача построения среднеквадратического приближения для f в пространстве L_2 (см. [7]) приводится к решению системы линейных алгебраических уравнений. Далее будут показаны особенности реализации метода при построении матрицы и решении системы уравнений в случае использования полиномиальных базисных сплайнов третьей высоты.

1. Построение базисных сплайнов

Интерполяционные полиномиальные базисные сплайны восьмого порядка аппроксимации третьей высоты [1] имеют вид

$$\omega_{j,0}(x) = (x_{j+1}-x)^4 (x_{j+1}-x_j)^7 [20(x_j-x)^3 + 10(x_j-x_{j+1})(x_j-x)^2 +$$
$$+ 4(x_j-x_{j+1})^2 (x_j-x) + (x_j-x_{j+1})^3], \; x \in [x_j, x_{j+1}];$$

$$\omega_{j,0}(x) = (x_{j-1}-x)^4 (x_{j-1}-x_j)^7 [20(x_j-x)^3 + 10(x_j-x_{j-1})(x_j-x)^2 +$$

$$+4(x_j - x_{j-1})^2(x_j - x) + (x_j - x_{j-1})^3],\ x \in [x_{j-1}, x_j);$$

$$\omega_{j,1}(x) = (x - x_j)(x_{j+1} - x)^4 (x_{j+1} - x_j)^6 [10(x_j - x)^2 + 4(x_j - x_{j+1})(x_j - x) +$$
$$+(x_j - x_{j+1})^2],\ x \in [x_j, x_{j+1});$$

$$\omega_{j,1}(x) = (x - x_j)^2 (x_{j-1} - x)^4 (x_{j-1} - x_j)^6 [10(x_j - x)^2 + 4(x_j - x_{j-1})(x_j - x) +$$
$$+(x_j - x_{j-1})^2],\ x \in [x_{j-1}, x_j);$$

$$\omega_{j,2}(x) = (x - x_j)^2 (x_{j+1} - x)^4 2(x_j - x_{j+1})^5 [4(x_j - x) + (x_j - x_{j+1})],\ x \in [x_j, x_{j+1});$$

$$\omega_{j,2}(x) = (x - x_j)(x_{j-1} - x)^4 2(x_j - x_{j-1})^5 [4(x_j - x) + (x_j - x_{j-1})],\ x \in [x_{j-1}, x_j);$$

$$\omega_{j,3}(x) = (x - x_j)^3 (x_{j+1} - x)^4 6(x_j - x_{j+1})^4,\ x \in [x_j, x_{j+1});$$

$$\omega_{j,3}(x) = (x - x_j)^3 (x_{j-1} - x)^4 6(x_j - x_{j-1})^4,\ x \in [x_{j-1}, x_j).$$

Отметим, что с помощью этих базисных сплайнов приближение $\tilde{u}(x)$ для функции $f(x)$, $x \in [x_j, x_{j+1}]$, обладающее свойствами $\tilde{u}^{(\alpha)}(x_k) = f^{(\alpha)}(x_k)$, $\alpha = 0, 1, 2, 3$, строится по формуле:

$$\tilde{u}(x) = \sum_{\alpha=0}^{3} f^{(\alpha)}(x_j)\omega_{j,\alpha}(x) + f^{(\alpha)}(x_{j+1})\omega_{j+1,\alpha}(x),$$

и, согласно [1], имеет место следующая оценка:

$$|f^{(\alpha)}(x) - \tilde{u}^{(\alpha)}(x)| \le Ch^{8-\alpha}\|f^{(8)}\|, \qquad x \in [a, b],\ C > 0.$$

Базисные сплайны $\omega_{j,i}$ обладают свойствами: $\omega_{j,i} \in C^3[a, b]$, $\omega_{j,i}^{(\alpha)}(x_k) = \delta_{j,k}\delta_{i,\alpha}$, где $\delta_{j,k}$ — символ Кронекера, $f(x) - \tilde{u}(x) \equiv 0$ для $f(x) = x^i$, $i = 0, 1, \ldots, 7$.

На промежутке $[x_j, x_{j+1}]$ базисные сплайны могут быть записаны в форме

$$\omega_{j,0}(x) = \frac{(x - x_{j+1})^4}{(x_j - x_{j+1})^7}(x_{j+1}^3 - 7x_j x_{j+1}^2 + 4xx_{j+1}^2 + 21x_j^2 x_{j+1} +$$
$$+10x^2 x_{j+1} - 28x_{j+1}xx_j - 35x_j^3 + 20x^3 - 70x^2 x_j + 84xx_j^2),$$

$$\omega_{j+1,0}(x) = \frac{(x - x_j)^4}{(x_j - x_{j+1})^7}(x_j^3 - 7x_{j+1}x_j^2 + 4xx_j^2 + 21x_{j+1}^2 x_j + 10x^2 x_j -$$
$$-28x_j xx_{j+1} - 35x_{j+1}^3 + 20x^3 - 70x^2 x_{j+1} + 84xx_{j+1}^2),$$

$$\omega_{j,1}(x) = \frac{(x - x_{j+1})^4 (x - x_j)}{(x_{j+1} - x_j)^6}(10x^2 - 24xx_j + 4xx_{j+1} + 15x_j^2 + x_{j+1}^2 - 6x_j x_{j+1}),$$

$$\omega_{j+1,1}(x)=\frac{(x-x_j)^4(x-x_{j+1})}{(x_{j+1}-x_j)^6}(10x^2-24xx_{j+1}+4xx_j+15x_{j+1}^2+x_j^2-6x_{j+1}x_j),$$

$$\omega_{j,2}(x)=\frac{(x-x_{j+1})^4(x-x_j)^2}{2(x_j-x_{j+1})^5}(4x-5x_j+x_{j+1}),$$

$$\omega_{j+1,2}(x)=\frac{(x-x_j)^4(x-x_{j+1})^2}{2(x_j-x_{j+1})^5}(4x-5x_{j+1}+x_j),$$

$$\omega_{j,3}(x)=\frac{(x-x_{j+1})^4(x-x_j)^3}{6(x_j-x_{j+1})^4},$$

$$\omega_{j+1,3}(x)=\frac{(x-x_j)^4(x-x_{j+1})^3}{6(x_j-x_{j+1})^4}.$$

2. Построение матрицы системы линейных алгебраических уравнений

В случае базисных сплайнов третьей высоты матрица M системы уравнений имеет блочный вид

$$M=\begin{pmatrix} M^{0,0} & M^{0,1} & M^{0,2} & M^{0,3} \\ M^{1,0} & M^{1,1} & M^{1,2} & M^{1,3} \\ M^{2,0} & M^{2,1} & M^{2,2} & M^{2,3} \\ M^{3,0} & M^{3,1} & M^{3,2} & M^{3,3} \end{pmatrix}.$$

Здесь через $M^{l,s}$ обозначен блок матрицы Грама, причем вычисление скалярных произведений $m_{i,j}^{l,s}=\int\limits_{x_{j-1}}^{x_{j+1}}\omega_{i,l}(x)\omega_{j,s}(x)\,dx$ в силу выполнения соотношения $\operatorname{supp}\omega_j=[x_{j-1},x_{j+1}]$ сводится к вычислению интегралов

$$m_{j,j}^{l,s}=\int\limits_{x_{j-1}}^{x_{j+1}}\omega_{j,l}(x)\omega_{j,s}^2(x)\,dx,\ m_{j,j+1}^{l,s}=\int\limits_{x_j}^{x_{j+1}}\omega_{j,l}(x)\omega_{j+1,s}(x)\,dx,\ l,s=0,1,2,3,$$

$$j=1,\ldots,n-2,$$

$$m_{0,0}^{l,s}=\int\limits_{x_0}^{x_1}\omega_{0,l}(x)\omega_{0,s}^2(x)\,dx,\ m_{n,n}^{l,s}=\int\limits_{x_{n-1}}^{x_n}\omega_{n,l}(x)\omega_{n,s}^2(x)\,dx.$$

Таким образом, каждый блок $M^{l,s}$ имеет ленточную трехдиагональную структуру.

После вычислений, получаем следующие значения:

1) для элементов $M^{0,0}$

$$m_{0,0}^{0,0} = m_{n,n}^{0,0} = \frac{521}{1287}h, m_{i,i}^{0,0} = \frac{1042}{1287}h, m_{i,i+1}^{0,0} = \frac{245}{2574}h, i = 1, \ldots, n-1,$$

2) для элементов $M^{1,1}$

$$m_{0,0}^{1,1} = m_{n,n}^{1,1} = \frac{5}{273}h^3, m_{i,i}^{1,1} = \frac{10}{273}h^3, m_{i,i+1}^{1,1} = \frac{-373}{36036}h^3, i = 1,\ldots,n-1,$$

3) для элементов блока $M^{2,2}$

$$m_{0,0}^{2,2} = m_{n,n}^{2,2} = \frac{43}{180180}h^5, m_{i,i}^{2,2} = \frac{43}{90090}h^5, m_{i,i+1}^{2,2} = \frac{131}{720720}h^5, i = 1,\ldots,n-1,$$

4) для элементов блока $M^{3,3}$

$$m_{0,0}^{3,3} = m_{n,n}^{3,3} = \frac{1}{1621620}h^7, m_{i,i}^{3,3} = \frac{1}{810810}h^7, m_{i,i+1}^{3,3} = \frac{-1}{1853280}h^7, i = 1,\ldots,n-1,$$

5) для элементов блока $M^{1,0}$

$$m_{0,0}^{1,0} = -m_{n,n}^{1,0} = \frac{151}{2002}h^2, m_{i,i}^{1,0} = 0, m_{i,i+1}^{1,0} = -m_{i+1,i}^{1,0} = \frac{-127}{4004}h^2, i = 1,\ldots,n-1,$$

6) для элементов блока $M^{2,0}$

$$m_{0,0}^{2,0} = m_{n,n}^{2,0} = \frac{137}{18018}h^3, m_{i,i}^{2,0} = \frac{137}{9009}h^3, m_{i,i+1}^{2,0} = m_{i+1,i}^{2,0} = \frac{155}{36036}h^3, i = 1,\ldots,n-1,$$

7) для элементов блока $M^{3,0}$

$$m_{0,0}^{3,0} = -m_{n,n}^{3,0} = \frac{383}{1081080}h^4, m_{i,i}^{3,0} = 0, m_{i,i+1}^{3,0} = -m_{i+1,i}^{3,0} = \frac{-521}{2162160}h^4, i = 1,\ldots,n-1,$$

8) для элементов блока $M^{2,1}$

$$m_{0,0}^{2,1} = -m_{n,n}^{2,1} = \frac{7}{3432}h^4, m_{i,i}^{2,1} = 0, m_{i,i+1}^{2,1} = -m_{i+1,i}^{2,1} = \frac{199}{144144}h^4, i = 1,\ldots,n-1,$$

9) для элементов блока $M^{3,1}$

$$m_{0,0}^{3,1} = m_{n,n}^{3,1} = \frac{1}{10010}h^5, m_{i,i}^{3,1} = \frac{1}{5005}h^5, m_{i,i+1}^{3,1} = m_{i+1,i}^{3,1} = \frac{-1}{13104}h^5, i = 1,\ldots,n-1,$$

10) для элементов блока $M^{3,2}$

$$m_{0,0}^{3,2} = -m_{n,n}^{3,2} = \frac{1}{83160}h^6, m_{i,i}^{3,2} = 0, m_{i,i+1}^{3,2} = -m_{i+1,i}^{3,2} = \frac{-43}{4324320}h^6, i = 1,\ldots,n-1,$$

Всего имеем 34 отличных от нуля и различных между собой элементов матрицы Грама.

3. Решение системы линейных алгебраических уравнений

Среднеквадратическое приближение функции $f(x)$ строим в виде

$$\tilde{u}(x) = \sum_{\alpha=0}^{3} \tilde{n}_j \omega_{j,\alpha}(x) + f^{(\alpha)}(x_{j+1})\,\omega_{j+1,\alpha}(x),\ x \in [x_k, x_{k+1}].$$

Для нахождения коэффициентов $c_{j,\alpha}$ необходимо решить систему линейных алгебраических уравнений $MC = F$, элементы матрицы M даны выше, а правая часть системы уравнений F представима в виде четырех блоков $F_\alpha = \{f_j^\alpha\}, j = 0, 1, \ldots, n,\ \alpha = 0, 1, \ldots, 3,$ элементы которых являются скалярными произведениями исходной функции и соответствующих базисных сплайнов:

$$f_i^\alpha = \int_{x_{i-1}}^{x_{i+1}} \omega_{i,\alpha}(x) f(x)\,dx.$$

Таким образом, правая часть системы линейных алгебраических уравнений F имеет вид:

$$F = (F_0, F_1, F_2, F_3)^T,$$

где элементы f_j^α вектора $F_\alpha, \alpha = 0, 1, 2, 3$ определяются так:

$$f_1^\alpha = \int_{x_0}^{x_1} \omega_{j,\alpha}(x) f(x)\,dx, f_j^\alpha = \int_{x_{j-1}}^{x_{j+1}} \omega_{j,\alpha}(x) f(x)\,dx, f_N^\alpha = \int_{x_{N-1}}^{x_N} \omega_{j,\alpha}(x) f(x)\,dx.$$

Для вычисления элементов вектора F использовалась составная квадратурная формула Симпсона [4], для решения системы линейных алгебраических уравнений $MC = F$ — итерационный метод Зейделя. Программа решения системы уравнений и построения среднеквадратического приближения написана на языке C++,для распараллеливания использовалась библиотека OpenMP [5], [6], вычисления проводились на двухпроцессорном компьютере. Распараллеливание алгоритма производилось при вычислении правой части системы уравнений (для вычисления элементов вектора $F_i, i = 0, 1, 2, 3$).

Время работы программы для различного числа потоков и разных размеров матрицы даны в таблице, приведенной ниже.

Размер матрицы	Число потоков	Время вычисления правой части, сек	Время решения СЛАУ, сек
[12×12]	1	3.04	0.05
[12×12]	2	3.06	0.05
[16×16]	1	4.85	0.06
[16×16]	2	3.22	0.06
[64×64]	1	24.38	0.93
[64×64]	2	13.07	0.93
[400×400]	1	160.69	26.14
[400×400]	2	82.47	25.98

Литература

1. *Бурова И. Г., Демьянович Ю. К.* Теория минимальных сплайнов. Изд-во СПбГУ, 1999. 357 с.
2. *Бурова И. Г.* Приближения неполиномиальными сплайнами максимального дефекта. Изд-во СПбГУ, 2007. 37 с.
3. *Фаддеев Д. К., Фаддеева В. Н.* Вычислительные методы линейной алгебры. Изд-во «Лань», 1960. 736 с.
4. *Самарский А. А.* Введение в численные методы. Изд-во «Наука», 1982. 271 с.
5. *Канг Су Гэтлин, Пит Айсенсие* OpenMP и C++, MSDN Magazine, Русская Редакция, Октябрь 2005 г.
   URL: http://www.microsoft.com/Rus/Msdn/Magazine/2005/10/OpenMP.mspx
6. *Barbara Chapman, Gabtiel Jost, Ruud van der Pas.* Using OpenMP. Portable Shared Memory Parallel Programming. The MIT Press, 2008. 353 с.
7. *Березин И. С., Жидков Н. П.* Методы вычислений. Т. 1. М., 1962.

СПЕЦИАЛЬНЫЕ КВАДРАТУРНЫЕ ФОРМУЛЫ ОБРАЩЕНИЯ ИНТЕГРАЛЬНОГО ПРЕОБРАЗОВАНИЯ ЛАПЛАСА

Н. И. Порошина

аспирантка кафедры вычислительной математики;
m02pni@star.math.spbu.ru

Санкт-Петербургский государственный университет

Аннотация: Построены различные квадратурные формулы наивысшей степени точности обращения интегрального преобразования Лапласа, как вещественные, так и комплексные, приспособленные для обращения изображений, соответствующих длительным медленно изменяющимся оригиналам, характерным для задач линейной вязкоупругости. Указаны явные формулы для вычисления узлов и коэффициентов квадратур, которые могут быть использованы для распараллеливания построения таких формул на многопроцессорных компьютерах.

1. Введение

Интегральное преобразование Лапласа $F(p)$ функции-оригинала $f(t)$,

$$F(p) = \int_0^\infty e^{-pt} f(t)\, dt, \tag{1}$$

представляет собой мощный инструмент для решения широкого класса прикладных задач математической физики. Одним из его главных достоинств является алгебраизация процедур математического анализа, с помощью которой удается свести интегральные и дифференциальные уравнения к более простым. Кроме того, изображение Лапласа является аналитической функцией в некоторой полуплоскости $\operatorname{Re} p > \lambda$, что позволяет привлечь к исследованию решаемой задачи результаты теории функций комплексного переменного.

Как правило, при решении задач операционными методами наиболее трудным этапом является процесс обращения, т. е. определение оригинала по его изображению. Существуют таблицы соответствия функций-оригиналов и их изображений [1], теоремы разложения, формула обращения Римана — Меллина, позволяющие теоретически точно находить оригинал. Но решение практических задач часто приводит к изображениям, к которым не могут быть применены эти классические приемы обращения. Следовательно, возникает необходимость разработки и применения приближенных методов.

Наиболее полно возможные подходы к задаче обращения и их реализация описаны в книге [2]. Обзор других способов обращения, не вошедших в [2], приведен в статье [3]. Теоретические основы операционного исчисления содержатся в классических работах [1], [4], [5], [6]. Вопросам приложения операционного исчисления к решению прикладных задач, среди прочих, посвящены фундаментальные труды [7], [8].

Не существует универсального метода обращения, дающего удовлетворительные результаты для произвольного изображения $F(p)$. Любой конкретный метод обращения должен учитывать специфику поведения изображения (или функции-оригинала), что прежде всего находит отражение в выборе подходящих систем функций в пространствах оригиналов и изображений, с которыми легко работать и с помощью которых могут быть хорошо приближены заданные образы и оригиналы. Выбор метода обращения существенно зависит от способа задания информации об изображении искомого оригинала. Выбор подходящих методов обращения для указанных ситуаций, их описание либо отсылка к соответствующей литературе рассмотрены в работе [3]. Цель настоящей работы состоит в рассмотрении методов обращения с помощью квадратурных формул наивысшей степени точности (к.ф.н.с.т.) специального вида.

2. Обращение преобразования лапласа по значениям изображения на вещественной оси

Один из вариантов метода обращения, использующий значения изображения в равноотстоящих точках вещественной полуоси $p > 0$, был предложен в [9]. В работах [10], [11] проведено его детальное исследование и приведены числовые параметры конкретных частных случаев общей схемы, так что здесь мы не будем повторять их результаты.

Известна следующая

Теорема 1 ([4], с. 385). *Пусть* $f(t) \in L(0, \infty)$ *и ее преобразование Лапласа равно*

$$F(p) = \int_0^\infty e^{-pt} f(t)\, dt.$$

Тогда для почти всех положительных значений t имеет место формула обращения

$$f(t) = \lim_{n\to\infty} \frac{(-1)^n t^{n-1}}{n!(n-2)!} \int_0^\infty e^{-pt} p^{2n-1} F^{(n)}(p)\, dp. \qquad (2)$$

Вхождение в эту формулу производной изображения ограничивает возможности ее применения. Однако интегрированием по частям она может быть представлена иначе:

$$f(t) = \lim_{n\to\infty} \int_0^\infty e^{-pt} P_{2n-1}(pt) F(p)\, dp,$$

где

$$P_{2n-1}(p)=\frac{(-1)^{n-1}(2n-1)!}{n!\,(n-2)!}\sum_{j=0}^{n}\binom{n}{j}\frac{(-p)^{2n-j-1}}{(2n-j-1)!}.$$

Нетрудно проверить, что этот многочлен выражается через многочлены Лагерра общего вида

$$L_n(x\,;\alpha)=\sum_{m=0}^{n}(-1)^m\binom{n+\alpha}{n-m}\frac{x^m}{m!} \tag{3}$$

простой формулой:

$$P_{2n-1}(p)=\frac{p^{n-1}}{(n-2)!}L_n(p;\,n-1).$$

В результате этих преобразований утверждение (2) теоремы запишется в виде

$$f(t)=\lim_{n\to\infty}f_n(t),\quad f_n(t)=\frac{1}{t(n-2)!}\int_0^\infty e^{-p}p^{n-1}L_n(p;\,n-1)\,F(p/t)\,dp.$$

Для приближенного вычисления последнего интеграла применим квадратурную формулу типа Гаусса с весом e^{-p} вида

$$\int_0^\infty e^{-p}\psi(p)dp\approx\sum_{k=1}^{m}A_k\psi(p_k),\tag{4}$$

точную для всех многочленов степени не выше $2m-1$.

Такой метод обращения преобразования Лапласа был предложен и исследован в работе [12]. Скорость его сходимости невелика, однако ее можно увеличить, построив линейные комбинации $f_n(t)$, вычисленные для различных *n Ryab*6. Однако для этого могут потребоваться формулы (4) с большим числом узлов *m* (например, несколько сотен). Покажем, как их можно эффективно построить.

Узлы формулы (4) совпадают с корнями многочленов Лагерра (3), т. е.

$$L_m(p_k)=0,\quad k=\overline{1,m}. \tag{5}$$

Будем находить корни уравнения (5) методом Ньютона:

$$p_k^{j+1}=p_k^j-L_m(p_k^j)/L_m'(p_k^j),\quad j=0,\ 1,\ \ldots \tag{6}$$

Начальные приближения ко всем корням многочленов (3), для которых метод (6) сходится, берем из приближенных равенств [13] (нам нужен лишь случай $\alpha=0$, поскольку $L_m(p)=L_m(p;0)$

$$p_1\approx\frac{(1+\alpha)(3+0.92\alpha)}{1+2.4m+1.8\alpha},\quad p_2-p_1\approx\frac{15+6.25\alpha}{1+0.9\alpha+2.5m},$$

$$\frac{p_{k+2}-p_{k+1}}{p_{k+1}-p_k}\approx\frac{1}{1+0.3\alpha}\left(\frac{1+2.55k}{1.9k}+\frac{1.26k\alpha}{1+3.5k}\right),\quad k=\overline{1,m-2}.$$

После нахождения с требуемой точностью всех узлов вычисляем коэффициенты формулы (4):

$$A_k = \frac{p_k}{(L'_m(p_k))^2}, \quad k = \overline{1, m}. \tag{7}$$

Заметим, что в данном случае узлы квадратурной формулы вещественны, что облегчает работу. Для вычисления узлов и коэффициентов таких квадратурных формул могут быть использованы параллельные алгоритмы. Так в случае нахождения коэффициентов к.ф. для них есть явные независимые формулы (7), а в случае нахождения узлов задача разбивается на несколько подзадач, которые могут быть решены параллельно и независимо друг от друга — что и является классическим применением теории параллельных алгоритмов. Так в нашем случае, такими независимыми подзадачами можно назвать уточнение значений корней уравнения (5) методом Ньютона после нахождения начальных приближений к ним. Эти процессы могут быть запущены параллельно и существенно ускорили бы решение исходной задачи обращения.

3. Комплексные квадратурные формулы наивысшей степени точности

Как известно, обращение преобразования Лапласа задается интегралом Римана — Меллина

$$f(t) = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} e^{pt} F(p)\, dp, \;\; c > \gamma, \;\; t > 0, \tag{8}$$

где γ — абсцисса сходимости интеграла Лапласа (1). Напомним, что интеграл (8) понимается в смысле главного значения, он не зависит от c и в случае разрыва оригинала в точке t мы получаем полусумму предельных значений оригинала слева и справа от точки t.

Далее для простоты считаем $\gamma = 0$, чего всегда можно добиться домножением оригинала на соответствующую экспоненту.

Пусть при некотором $s > 0$ функция $\varphi_s(p) = p^s F(p)$ регулярна в полуплоскости $\operatorname{Re} p > 0$. Преобразуем интеграл (8) с учётом предложенной замены и для приближённого вычисления получившегося интеграла выберем попарно различные точки $p_1, \ldots, p_n$ в полуплоскости $\operatorname{Re} p > 0$ в качестве узлов квадратурной формулы (к.ф.) вида

$$\frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} e^p p^{-s} \varphi(p)\, dp \approx \sum_{k=1}^{n} A_k \varphi(p_k) \tag{9}$$

и потребуем, чтобы формула (9) была точна для функций $\varphi(p) = p^{-j}$, $j = \overline{0, 2n-1}$, т. е. имела наивысшую степень точности (в таком случае будем говорить, что к.ф. обладает $(2n-1)$ -свойством). Это требование равносильно выполнению равенств

$$\sum_{k=1}^{n} A_k p_k^{-j} = \frac{1}{\Gamma(s+j)}, \quad j = \overline{0, 2n-1}. \tag{10}$$

Такая к.ф. существует и единственна, все её узлы p_k различны и лежат в правой полуплоскости $\operatorname{Re} p > 0$ [2].

По построению формула (9) точна для оригиналов вида $t^{s-1} Q_{2n-1}(t)$, где Q_{2n-1} — любой многочлен степени не выше $2n-1$.

К сожалению, к.ф.н.с.т. (9) плохо приспособлены для обращения изображений, соответствующих медленно протекающим длительным процессам. Так, в задачах линейной вязкоупругости [14], описывающих напряженное состояние на основе определяющего соотношения Больцмана — Вольтерра (пространственные координаты ниже для простоты опущены), деформации ε и напряжения σ связаны соотношением (обобщённый закон Гука)

$$\varepsilon(t) = \frac{1}{E}\left(\sigma(t) + \lambda \int_0^t K(t-\tau)\sigma(\tau) d\tau\right). \tag{11}$$

Первое слагаемое справа в (11) соответствует мгновенной деформации, а второе — наследственной деформации. Как правило, из эксперимента определяется функция ползучести материала — значение правой части (11) при $\sigma = \text{const}$, т. е.

$$\varepsilon(t) = \frac{c}{E}\left(1 + \lambda \int_0^t K(\tau)\, d\tau\right). \tag{12}$$

Важнейшей задачей становится выбор подходящего ядра K интегрального уравнения (11), определяющего функцию ползучести (12). Ядро K должно иметь интегрируемую особенность в точке $t = 0$. Чаще всего в качестве такового берут дробно-экспоненциальную функцию Работнова [14] (резольвента ядра Абеля)

$$Э_\alpha(\beta, t) = t^\alpha \sum_{k=0}^{\infty} \frac{(\beta t^{1+\alpha})^k}{\Gamma((1+\alpha)(1+k))}, \quad -1 < \alpha \leqslant 0. \tag{13}$$

Способ определения параметров дробно-экспоненциальной функции по измеренной функции ползучести описан в работе [15].

Интеграл от этого ядра по полуоси $t \geqslant 0$ должен быть конечным, для чего необходимо $\beta < 0$. Не умаляя общности, далее считаем $\beta = -1$, и пусть символ $Э_\alpha(t)$ означает $Э_\alpha(-1, t)$.

В наследственной механике твердого тела наряду с функцией (13) широко используется и интеграл от нее с переменным верхним пределом. Для облегчения использования этих величин составлены таблицы функций [14]

$$F_1(\alpha, x) = t^{-\alpha} Э_\alpha(x), \quad F_2(\alpha, x) = t^{-\alpha-1} \int_0^t Э_\alpha(\tau) d\tau, \quad x = t^{\alpha+1}.$$

Однако при решении конкретных задач необходимо вводить в память вычислительной машины части этих таблиц, соответствующие найденным

параметрам $Э_\alpha$ — функций, которые заранее неизвестны и определяются в процессе решения задачи (и в итоге таковых в таблице может не оказаться). При изменении параметров приходится эту работу проделывать заново, что неудобно и сопряжено с внесением ошибок.

Применяя преобразование Лапласа к уравнению (11), получаем

$$\bar{\varepsilon}(p) = \frac{1}{E}\left(1 + \frac{\lambda}{p^{\alpha+1} - \beta}\right)\bar{\sigma}(p). \tag{14}$$

В частности, изображение функции ползучести равно

$$\bar{\varepsilon}(p) = \frac{c}{pE}\left(1 + \frac{\lambda}{p^{\alpha+1} - \beta}\right). \tag{15}$$

Применяя преобразование Лапласа к уравнению движения среды, получим второе соотношение между $\bar{\varepsilon}(p)$ и $\bar{\sigma}(p)$, а затем, используя (14), найдём $\bar{\varepsilon}(p)$ и $\bar{\sigma}(p)$. Если искомые функции $\varepsilon(t)$ и $\sigma(t)$ ограничены, то можно положить $s = 1$ и в качестве $\varphi_s(p)$ рассматривать функции $p\bar{\varepsilon}(p)$ и $p\bar{\sigma}(p)$. Они зависят фактически от p^a, $a = \alpha + 1$. Заметим, что для реальных процессов деформирования значение s можно увеличить: так, изображение по Лапласу второго слагаемого в (12), определяющего наследственную деформацию, равно $\lambda/(p(p^a - \beta))$, и можно положить $s = 1 + a$. Искомые решения $\varepsilon(t)$ и $\sigma(t)$ на конечном по t отрезке времени допускают хорошие приближения вида $t^{s-1}Q(t^a)$ при $0 < a \leqslant 1$, где $Q(t)$ — некоторый многочлен, и при уменьшении a скорость их изменения уменьшается.

В таком случае целесообразно вместо к.ф.н.с.т. построить и использовать обобщенные квадратурные формулы наивысшей степени точности (о.к.ф.н.с.т.) вида (9), точные для функций $\varphi(p) = p^{-aj}$, $j = \overline{0, 2n-1}$, или для оригиналов вида $t^{s-1}Q_{2n-1}(t^a)$. (Q — произвольный многочлен).

Такие формулы были введены в работе [16] и исследованы в статье [17]. В частности, в [17] доказана

Теорема 2. *Для того чтобы формула (9) была точна для функций* $\varphi(p) = p^{-aj}$, $j = \overline{0, 2n-1}$, *необходимо и достаточно выполнение двух условий:*

1) формула (9) интерполяционная, т. е. точна для функций $\varphi(p) = p^{-aj}$, $j = \overline{0, n-1}$;

2) построенный по узлам формулы (9) многочлен

$$\omega_n(x) = \prod_{k=1}^{n}(x - p_k^{-a}) \tag{16}$$

удовлетворяет условиям

$$\int_{c-i\infty}^{c+i\infty} e^p p^{-s} \omega_n(p^{-a}) p^{-am}\, dp = 0, \quad m = \overline{0, n-1}, \quad c > 0. \tag{17}$$

В работе [17] показано, что многочлен (16), удовлетворяющий условию (17), существует и определяется однозначно, а его корни, т. е. узлы о.к.ф.н.с.т., удовлетворяют неравенствам $\mathrm{Re}(p_k^a) > 0,\ k = \overline{1, n}$. Отметим, что узлы и коэффициенты формулы (9) суть комплексные числа. Оценки погрешности о.к.ф.н.с.т. для конкретного n получены в работе [18]. Эти формулы оказались весьма эффективными при решении задач линейной вязкоупругости [15], хотя их применение более трудоёмкий процесс (по сравнению с к.ф.н.с.т.).

Классические к.ф.н.с.т. [2] получаются в случае $a = 1$.

Применение методов параллельных алгоритмов в описанных выше способах обращения преобразования Лапласа возможно лишь в случае построения к.ф.н.с.т., поскольку для о.к.ф.н.с.т. нет явных формул для вычисления узлов и коэффициентов. В случае же к.ф.н.с.т. методы теории параллельных алгоритмов применимы для вычисления коэффициентов квадратурной формулы, поскольку они зависят лишь от значений узлов к.ф. и для них есть явные формулы [18].

Литература

1. *Диткин В. А., Прудников А. П.* Интегральные преобразования и операционное исчисление. М., 1961. 524 с.
2. *Крылов В. И., Скобля Н. С.* Методы приближенного преобразования Фурье и обращения преобразования Лапласа. М., 1974. 224 с.
3. *Порошина Н. И., Рябов В. М.* Об обращении преобразования Лапласа некоторых специальных функций // Вестн. С.-Петерб. ун-та. Сер. 1. 2011. Вып. 3. С. 55–64.
4. *Widder D. V.* The Laplace transform. Princeton, 1946. 406 p.
5. *Дёч Г.* Руководство к практическому применению преобразования Лапласа и Z-преобразования. М., 1971. 288 с.
6. *Лаврентьев М. А., Шабат Б. В.* Методы теории функций комплексного переменного. М., 2002. 688 с.
7. *Лурье А. И.* Операционное исчисление и его приложение к задачам механики. М.-Л., 1951. 433 с.
8. *Слепян Л. И., Яковлев Ю. С.* Интегральные преобразования в нестационарных задачах механики. Л., 1980. 344 с.
9. *Рябцев И. И.* Приближенное вычисление оригинала по значениям изображения в равноотстоящих точках действительной оси // Математика. 1966. №3. С. 139–143.
10. *Рябов В. М.* Метод обращения преобразования Лапласа, использующий значения изображения на вещественной оси // Вестн. Ленингр. ун-та. 1982. №1. С. 48–53.
11. *Рябов В. М.* О точности некоторых методов обращения преобразования Лапласа // Методы вычислений. Вып. 14. Л., изд-во Ленингр. ун-та, 1985. С. 59–71.
12. *Рябов В. М.* Формула обращения преобразования Лапласа, основанная на теореме Виддера // Вестн. Ленингр. ун-та. 1989. №22. С. 35–39.
13. *Stroud A. H., Secrest D.* Gaussian quadrature formulas. N.-Y., 1966. 374 p.
14. *Работнов Ю. Н.* Элементы наследственной механики твердых тел. М., 1977. 384 с.

15. *Екельчик В. С., Рябов В. М.* Об использовании одного класса наследственных ядер в линейных уравнениях вязкоупругости // Механика композитных материалов. 1981. № 3. С. 393–404.
16. *Рябов В. М.* О многочленах, возникающих при численном обращении преобразования Лапласа // Методы вычислений. Вып. 12. Л., изд-во Ленингр. ун-та, 1981. С. 46–53.
17. *Рябов В. М.* Свойства квадратурных формул наивысшей степени точности, применяемых для обращения преобразования Лапласа // Журн. вычислит. матем. и математ. физ. 1989. Т. 29. № 7. С. 1083–1087.
18. *Матвеева Т. А., Рябов В. М.* Обобщенные квадратурные формулы численного обращения преобразования Лапласа // Вестн. С.-Петерб. ун-та. Сер. 1. 2000. Вып. 4. С. 7–11.

О РАСПАРАЛЛЕЛИВАНИИ РЕШЕНИЯ ЗАДАЧИ ДИРИХЛЕ В КРУГЕ

И. Г. Бурова

профессор кафедры вычислительной математики;
BurovaIG@mail.ru.

В. А. Ракчаев

аспирант кафедры вычислительной математики;
vladimir.rakchaev@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Здесь строится решение задачи Дирихле в круге с помощью гармонических сплайнов и обсуждаются результаты распараллеливания вычислений при представлении решения с помощью ряда

1. Тригонометрические ряды и формулы Филона

Традиционно решение задачи Дирихле в круге радиуса R с центром в начале координат

$$r^2\frac{\partial^2 u}{\partial r^2}+r\frac{\partial u}{\partial r}+\frac{\partial^2 u}{\partial r\theta^2}=0,$$

$0\le r\le R,\ -\infty\le\theta\le\infty.\ u(R,\theta)=f(\theta),\ u(r,\theta+2\pi)=u(r,\theta),$ строится с помощью тригонометрического ряда [1]

$$u(r,\theta)=\frac{\alpha_0}{2}+\sum_{k=1}^{\infty}\left(\frac{r}{R}\right)^k(\alpha_k\cos k\theta+\beta_k\sin k\theta),$$

причем для вычисления интегралов

$$\alpha_k=\frac{1}{\pi}\int_0^{2\pi}f(t)\cos ktdt,\ \beta_k=\frac{1}{\pi}\int_0^{2\pi}f(t)\sin ktdt$$

можно применять квадратурные формулы Филона [2]. В предположении, что промежуток $[a, b]$ конечен, разобьем его на $2n$ частичных отрезков одинаковой длины точками $x_s=a+sh,\ h=(b-a)/(2n),\ s=0,1,\ldots,2n.$

Тогда

$$\int_a^b f(x)\sin kxdx\approx h\left[-\alpha f(a)\cos ka+\alpha f(b)\cos kb+\beta\left(\frac{f(a)\sin ka+f(b)\sin kb}{2}+\right.\right.$$
$$\left.\left.+\sum_{j=0}^{n-2}f(x_{2j+2})\sin kx_{2j+2}\right)+\gamma\sum_{j=0}^{n-1}f(x_{2j+1})\sin kx_{2j+1}\right],$$

где $\theta = kh$, $\alpha = (\theta^2 + \theta\sin\theta\cos\theta - 2\sin^2\theta)/\theta^3$, $\beta = (2\theta(1+\cos^2\theta) - 4\sin\theta\cos\theta)/\theta^3$, $\gamma = 4(\sin\theta - \theta\cos\theta)/\theta^3$.

Для численного решения задачи

$$\Delta u = 0.\ |z| < 1,$$

$$u|_{|z|=1} = \frac{\sin\psi}{5 + 4\cos\psi}.$$

производились вычисления

$$u(r,\theta) = \frac{\alpha_0}{2} + \sum_{k=1}^{m}\left(\frac{r}{R}\right)^k (\alpha_k \cos k\theta + \beta_k \sin k\theta)$$

с применением квадратурных формул Филона с помощью разработанной программы на языке C++. Для распараллеливания вычислений использовалась среда OpenMP, вычисления проводились на двухпроцессорном компьютере. Вычисления по формулам Филона проводились распараллеливанием с помощью прагмы sections на два потока. При достаточно больших значениях параметров m, n (>1000) было получено уменьшение времени счета при распараллеливании вычислений по сравнению с последовательным вариантом программы.

2. Гармонические сплайны

Далее опишем решение задачи Дирихле в круге с помощью гармонических сплайнов [3]. Пусть Γ — окружность радиуса R с центром в нуле.

Как известно, гармоническая функция $u(r,\theta)$ выражаются через $u(r,\varphi)$ следующим образом: Пусть Γ — окружность радиуса ρ с центром в нуле.

$$u(r,\theta) = \frac{1}{2\pi}\int_0^{2\pi} u(\rho,\varphi)\frac{\rho^2 - r^2}{r^2 + \rho^2 - 2r\rho\cos(\theta - \varphi)}d\varphi,$$

Пусть на окружности Γ задана сетка узлов $\{\varphi_j\}$.

Будем аппроксимировать функцию $u(R,\theta)$ на Γ соотношением, в котором участвуют значения функции $u(R,\varphi_j)$ в узлах сетки на границе Γ и базисные тригонометрические функции $\omega_j(\theta)$:

$$\tilde{u}(R,\theta) = \sum_j u(R,\varphi_j)\omega_j(\theta).$$

Тригонометрические базисные функции [4] обеспечивают совпадение на окружности Γ приближаемой функции $u(R,\theta)$ и аппроксимации $\tilde{u}(R,\theta)$, если $u(R,\theta) = \sin(k\theta)$ и $u(R,\theta) = \cos(k\theta)$, $k = 0, 1, \ldots, m+1$, m — определяет порядок погрешности. Функцию

$$\Omega_j(r,\theta) = \int_0^{2\pi}\omega_j(\varphi)\frac{\rho^2 - r^2}{r^2 + \rho^2 - 2r\rho\cos(\theta - \varphi)}d\varphi$$

называем гармоническим сплайном.

Приведем формулы некоторых гармонических сплайнов соответствующих тригонометрическим базисным сплайнам первого и второго порядка. 1. Пусть сплайны $\omega_j(\varphi)$ задаются соотношением

$$\omega_j(\varphi) = \begin{cases} 1, \varphi \in [\varphi_j, \varphi_{j+1}] \\ 0, \varphi \notin [\varphi_j, \varphi_{j+1}] \end{cases},$$

$$\Omega(r, \theta) = \int_{\varphi_j}^{\varphi_{j+1}} \frac{\rho^2 - r^2}{r^2 + \rho^2 - 2r\rho\cos(\theta - \varphi)} d\varphi =$$

$$= 2\left[\frac{\rho + r}{\rho - r}\frac{\varphi_{j+1} - \theta}{2}\right] - 2\left[\frac{\rho + r}{\rho - r}\frac{\varphi_j - \theta}{2}\right].$$

2. Пусть $\omega_j(\varphi)$ задается соотношением

$$\omega_j(\varphi) = \begin{cases} \dfrac{\sin\dfrac{\varphi - \varphi_{j-1}}{2}\sin\dfrac{\varphi - \varphi_{j-2}}{2}}{\sin\dfrac{\varphi_j - \varphi_{j-1}}{2}\sin\dfrac{\varphi_j - \varphi_{j-2}}{2}}, \ \varphi \in [\varphi_{j-1}, \varphi_j]; \\ \dfrac{\sin\dfrac{\varphi - \varphi_{j-1}}{2}\sin\dfrac{\varphi - \varphi_{j+1}}{2}}{\sin\dfrac{\varphi_j - \varphi_{j-1}}{2}\sin\dfrac{\varphi_j - \varphi_{j+1}}{2}}, \ \varphi \in [\varphi_j, \varphi_{j+1}]; \\ \dfrac{\sin\dfrac{\varphi - \varphi_{j+1}}{2}\sin\dfrac{\varphi - \varphi_{j+2}}{2}}{\sin\dfrac{\varphi_j - \varphi_{j+1}}{2}\sin\dfrac{\sin\varphi_j - \varphi_{j+2}}{2}}, \ \varphi \in [\varphi_{j+1}, \varphi_{j+2}]; \\ 0, \ \varphi \notin [\varphi_{j-1}, \varphi_{j+2}] \end{cases}.$$

Соответствующий гармонический сплайн имеет вид

$$\Omega(r, \theta) = \int_{-\pi}^{\pi} \frac{\rho^2 - r^2}{r^2 + \rho^2 - 2r\rho\cos(\theta - \varphi)} d\varphi =$$

$$= \frac{\rho^2 - r^2}{\sin\dfrac{\varphi_j - \varphi_{j-1}}{2}\sin\dfrac{\varphi_j - \varphi_{j-2}}{2}} \int_{\varphi_{j-1}}^{\varphi_j} \frac{\sin\dfrac{\varphi - \varphi_{j-1}}{2}\sin\dfrac{\varphi - \varphi_{j-2}}{2}}{r^2 + \rho^2 - 2r\rho\cos(\theta - \varphi)} d\varphi +$$

$$+ \frac{\rho^2 - r^2}{\sin\dfrac{\varphi_j - \varphi_{j-1}}{2}\sin\dfrac{\varphi_j - \varphi_{j+1}}{2}} \int_{\varphi_j}^{\varphi_{j+1}} \frac{\sin\dfrac{\varphi - \varphi_{j-1}}{2}\sin\dfrac{\varphi - \varphi_{j+1}}{2}}{r^2 + \rho^2 - 2r\rho\cos(\theta - \varphi)} d\varphi +$$

$$+\frac{\rho^2-r^2}{\sin\dfrac{\varphi_j-\varphi_{j+2}}{2}\sin\dfrac{\varphi_j-\varphi_{j+1}}{2}}\int_{\varphi_{j+1}}^{\varphi_{j+2}}\frac{\sin\dfrac{\varphi-\varphi_{j+2}}{2}\sin\dfrac{\varphi-\varphi_{j+1}}{2}}{r^2+\rho^2-2r\rho\cos(\theta-\varphi)}d\varphi.$$

Итак, задавая значения функции на границе круга, получаем решение задачи Дирихле внутри круга. Аналогично нетрудно получить решение во внешности круга. Применяя теорию конформных отображений, получаем решение задачи Дирихле в соответствующих областях. Погрешность метода определяется погрешностью аппроксимации функции на границе круга. Результаты вычислений подтверждают теоретические оценки погрешностей.

Литература

1. *Канторович Л. В., Крылов В. И.* Приближенные методы высшего анализа. Л.; М., 1949.
2. *Мысовских И. П.* Лекции по методам вычислений. СПб., 1998. 472 с.
3. *Бурова И. Г.* Об аппроксимации локальными тригонометрическими и гармоническими сплайнами. Л., 1987. 49 с. / Деп. ВИНИТИ, №132-В87.
4. *Бурова И. Г., Демьянович Ю. К.* Минимальные сплайны и их приложения. Изд-во СПбГУ, 2010. 364 с.

РЕШЕНИЕ ОДНОЙ ОБОБЩЕННОЙ ЗАДАЧИ ЭРМИТА — БИРКГОФА С ПОМОЩЬЮ НЕПОЛИНОМИАЛЬНЫХ ИНТЕГРО-ДИФФЕРЕНЦИАЛЬНЫХ СПЛАЙНОВ

О. В. Родникова

аспирантка кафедры вычислительной математики;
ilmarik.spb@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Изучается решение обобщенной задачи Эрмита — Биркгофа с помощью неполиномиальных интегро-дифференциальных сплайнов.

Хорошо известно решение задачи Эрмита-Биркгофа с помощью интерполяционных полиномов (см., напр. [1]). В ряде случаев получить решение с помощью интерполяционных полиномов невозможно. В работе [2] предлагается решение задачи Эрмита — Биркгофа с помощью неполиномиальных минимальных сплайнов. В книге [3] предложены полиномиальные интегро-дифференциальные сплайны. Неполиномиальные интегро-дифференциальные сплайны рассмотрены в статье [4]. Здесь рассмотрим решение одной обобщенной задачи Эрмита-Биркгофа с помощью неполиномиальных интегро-дифференциальных сплайнов.

Рассматриваем функцию $U(x)$, достаточное количество раз дифференцируемую на конечном отрезке $[a, b]$. Построим на $[a, b]$ упорядоченную по возрастанию сетку узлов $\{x_k\}_{k=0}^{n}$, $n \in \mathbb{Z}$, с переменным шагом h_k на этом отрезке, считая, что точки a, b также являются узлами.

Предполагаем, что в узлах сетки заданы значения производной функции $U'(x_k)$ и на каждом промежутке $[x_k, x_{k+1}]$ известны значения интегралов $\int_{x_k}^{x_{k+1}} U(x)\,dx$. Требуется построить на промежутке $[a, b]$ функцию $\tilde{U}(x)$ так, чтобы на каждом промежутке $[x_k, x_{k+1}]$ выполнялись условия

1. $\tilde{U}'(x_k) = U'(x_k),\ \ \tilde{U}'(x_{k+1}) = U'(x_k).$

2. $\int_{x_k}^{x_{k+1}} \tilde{U}(x)\,dx = \int_{x_k}^{x_{k+1}} U(x)\,dx.$

Решать эту задачу будем следующим образом. На каждом сеточном интервале $[x_k, x_{k+1}]$ строим приближение $\tilde{U}(x)$ для функции $U(x)$ в виде

$$\tilde{U}(x) = U'(x_k)\,\omega_{k,1}(x) + U'(x_{k+1})\,\omega_{k+1,1}(x) + \int_{x_k}^{x_{k+1}} U(x)\,dx\,\omega_k^{<1>}(x),\ x \in [x_k, x_{k+1}]. \quad (1)$$

Базисные функции $\omega_{k,1}(x)$, $\omega_{k+1,1}(x)$, $\omega_k^{<1>}(x)$ находим из условий

$$U(x)-\tilde{U}(x)=0, \text{ если } U(x)=1, \varphi_1(x), \varphi_2(x),$$

где функции $\varphi_1(x), \varphi_2(x)$ достаточное число раз дифференцируемы на отрезке $[a, b]$.

Из условий, приведенных выше получаем линейную систему уравнений

$$(x_{k+1}-x_k)\omega_k^{<1>}(x)=1,$$

$$\varphi_{1'}(x_k)\omega_{k,1}(x)+\varphi_{1'}(x_{k+1})\omega_{k+1,1}(x)+\int_{x_k}^{x_{k+1}}\varphi_1(x)\,dx\omega_k^{<1>}(x)=\varphi_1(x),$$

$$\varphi_{2'}(x_k)\omega_{k,1}(x)+\varphi_{2'}(x_{k+1})\omega_{k+1,1}(x)+\int_{x_k}^{x_{k+1}}\varphi_2(x)\,dx\omega_k^{<1>}(x)=\varphi_2(x).$$

Определитель этой системы имеет вид

$$\Delta=h_k(\varphi_1'(x_k)\varphi_2'(x_{k+1})-\varphi_1'(x_{k+1})\varphi_2'(x_k)). \tag{2}$$

После несложных преобразований для базисных функций получаем следующие выражения:

$$\omega_{k,1}(x)=\frac{1}{\Delta}\left(\varphi_1'(x_{k+1})\left(\int_{x_k}^{x_{k+1}}\varphi_2(x)\,dx-h_k\varphi_2(x)\right)-\right.$$

$$\left.-\varphi_{2'}(x_{k+1})\left(\int_{x_k}^{x_{k+1}}\varphi_1(x)\,dx-h_k\varphi_1(x)\right)\right), \tag{3}$$

$$\omega_{k+1,1}(x)=\frac{1}{\Delta}\left(\varphi_{1'}(x_k)\left(h_k\varphi_2(x)-\int_{x_k}^{x_{k+1}}\varphi_2(x)\,dx\right)\right)-$$

$$\left.-\varphi_{2'}(x_k)\left(h_k\varphi_1(x)-\int_{x_k}^{x_{k+1}}\varphi_1(x)\,dx\right)\right), \tag{4}$$

$$\omega_k^{<1>}(x)=\frac{1}{h_k}.$$

Справедлива следующее утверждение:

Лемма. Построенное таким образом приближение $\tilde{U}(x)$ обладает следующими свойствами:

1. $\tilde{U}'(x_k)=U'(x_k),\ \tilde{U}'(x_{k+1})=U'(x_k).$
2. $\int_{x_k}^{x_{k+1}}\tilde{U}(x)\,dx=\int_{x_k}^{x_{k+1}}U(x)\,dx.$

Рассмотрим несколько частных случаев.

Случай 1. Если $\varphi_1(x)=x,\ \varphi_2(x)=x^2,$ то для определителя (2) получаем выражение $\Delta=2h_k^2,$ и базисные функции (5) принимают вид

$$\omega_{k,1}(t) = -\frac{h_k}{2}\left(t^2 - 2t + \frac{2}{3}\right),$$

$$\omega_{k+1,1}(t) = \frac{h_k}{2}\left(t^2 - \frac{1}{3}\right),$$

$$\omega_k^{<1>}(t) = \frac{1}{h_k}.$$

Случай 2. Если $\varphi_1(x) = e^x$, $\varphi_2(x) = e^{2x}$, то для определителя (2) получаем выражение

$$\Delta = 2h_k\left(e^{3x_k+2h_k} - e^{3x_k+h_k}\right),$$

и базисные функции (5) принимают следующий вид

$$\omega_{k,1}(t) = \frac{1}{\Delta}\left(-\frac{3}{2}e^{3x_k+3h_k} - \frac{1}{2}e^{3x_k+h_k} + 2e^{3x_k+2h_k} + 2h_k e^{3x_k+th_k+2h_k} - h_k e^{3x_k+2th_k+h_k}\right),$$

$$\omega_{k+1,1}(t) = \frac{1}{\Delta}\left(h_k e^{3x_k+2th_k} - \frac{1}{2}e^{3x_k+2h_k} - \frac{3}{2}e^{3x_k} + 2e^{3x_k+h_k} - 2h_k e^{3x_k+th_k}\right),$$

$$\omega_k^{<1>}(t) = \frac{1}{h_k}.$$

Случай 3. Если $\varphi_1(x) = \sin(x)$, $\varphi_2(x) = \cos(x)$, то для определителя (2) получаем выражение $\Delta = -h_k \sin(h_k)$, и базисные функции (5) принимают следующий вид

$$\omega_{k,1}(t) = \frac{1}{\Delta}(\sin(h_k) - h_k\cos(h_k - th_k)),$$

$$\omega_{k+1,1}(t) = \frac{1}{\Delta}(-\sin(h_k) + h_k\cos(th_k)),$$

$$\omega_k^{<1>}(t) = \frac{1}{h_k}.$$

Справедлива следующая теорема:

Теорема. Пусть функция $U \in C^3[a,b]$. Приближение $\tilde{U}(x)$ задается формулой (1). Тогда $\max\limits_{x\in[x_k,x_{k+1}]} |\tilde{U}(x) - U(x)| \le 9h_k^3/24 \max\limits_{x\in[x_k,x_{k+1}]} |LU|$, где функция LU имеет вид

1. Если $\varphi_1(x) = e^x$, $\varphi_2(x) = e^{2x}$, то $LU = U'''(x) - 3U''(x) + 2U'(x)$.
2. Если $\varphi_1(x) = \sin(x)$, $\varphi_2(x) = \cos(x)$, то $LU = U'''(x) + U'(x)$.
3. Если $\varphi_1(x) = x$, $\varphi_2(x) = x^2$, то $LU = U'''(x)$.

В среде Maple получены значения теоретических и экспериментальных погрешностей приближений для различных функций.

Литература

1. *Мысовских И. П.* Лекции по методам вычислений. СПб., 1998. 472 с.
2. *Бурова И. Г, Тимофеев В. А.* Решение задачи Эрмита-Биркгофа с помощью минимальных неполиномиальных сплайнов // Вестник СПбГУ. Сер. 1. Вып 3. 2006. С 50–51.
3. *Киреев В. И., Пантелеев А. В.* Численные методы в примерах и задачах. М., 2008. 480 с.
4. *Бурова И. Г.* О моделировании неполиномиальных интегро-дифференциальных приближений. Труды СПИИРАН, вып. 4(19). 2011. С. 176–208.
5. *Бурова И. Г., Демьянович Ю. К.* Минимальные сплайны и их приложения. Изд-во СПбГУ, 2010. 364 с.

РАСПАРАЛЛЕЛИВАНИЕ ГНЕЗДОВЫХ СПЛАЙН-ВЭЙВЛЕТНЫХ РАЗЛОЖЕНИЙ

И. Д. Мирошниченко

ст. преп. кафедры параллельных алгоритмов;
irina_mir_@mail.ru

Санкт-Петербургский государственный университет

Аннотация: Рассматривается задача построения гнездовых сплайн-вейвлетных разложений, а также анализируются возможности распараллеливания процесса построения этих разложений.

Задача реализации гнездовых сплайн-вейвлетных разложений — это, в первую очередь, хорошо формализуемая задача, решаемая в пространствах матриц больших размерностей независимых данных, но, в то же время, матриц, особой структуры, которая позволяет находить эффективные способы представления таких матриц. Большие размерности матриц, особые способы их представления, независимые блоки данных, — все это обещает хорошие результаты при применении элементов параллельного программирования.

Задача представления гнездовых сплайн-вэйвлетных разложений [2] — это задача представления сплайновых (в общем случае разрывных и неполиномиальных) пространств, получающихся при удалении группы узлов (гнезд).

Гнездом назовем множество узлов, $\Gamma = \Gamma(k,s) \stackrel{\text{def}}{=} \{x_{k-s}, x_{k-s+1}, ..., x_k, x_{k+1}\}$, $x_j^* \stackrel{\text{def}}{=} x_j$ при при $j \le k-s-1$, $x_j^* \stackrel{\text{def}}{=} x_{j+s+2}$, при $j \ge k-s$, $j \in J_{N-s-2}$, на сетке $X_{N\{\Gamma\}}$: $x_0^* < x_1^* < ... < x_{N-s-3}^* < x_{N-s-2}^*$, $0 \le s+1 \le k \le N-2$, где s, k, N — целые.

Для множества Γ (гнезда удаляемых узлов) полагаем

$$S_j^* \stackrel{\text{def}}{=} (x_j^*, x_{j+1}^*) \cup (x_{j+1}^*, x_{j+2}^*), \ j \in J'_{N-s-4}, \quad G^* \stackrel{\text{def}}{=} \bigcup_{j \in J_{N-2-3}} (x_j^*, x_{j+1}^*),$$

задаем цепочку двумерных векторов $\{a_j^*\}_{j \in J'_{N-s-3}}$ и строим систему функций $\{\omega_j^*\}_{j \in J'_{N-s-3}}$. Для таким образом определенного пространства $\mathbb{S}_{N\{\Gamma\}} \stackrel{\text{def}}{=} Cl_p L\{\omega_j^*\}_{j \in J'_{N-s-3}}$ (размерность которого $\dim \mathbb{S}_{\mathbb{N}\{\Gamma\}} = N-s-1$).

При $\forall t \in G$, $j \in J'_{N-s-3}$ справедливы соотношения $a_j^* \omega_j^*(t) = a_j \omega_j(t)$ при $j \le k-s-3$, $a_j^* \omega_j^*(t) = a_{j+s+2} \omega_{j+s+2}(t)$ при $j \ge k-s$, которые и дают искомое разложение. В этих условиях [2] выполняются тождества для $j = k-s-2, \ldots, k+1$

$$\omega_{k-s-2}^*(t) \equiv \sum_{j=k-s-2}^{k+1} p_{k-s-2,j} \omega_j(t), \quad \omega_{k-s-1}^*(t) \equiv \sum_{j=k-s-2}^{k+1} p_{k-s-1,j} \omega_j(t),$$

$$p_{k-s-2,\,j} \stackrel{\text{def}}{=} \frac{\det(a_j,\ a^*_{k-s-1})}{\det(a^*_{k-s-2},\ a^*_{k-s-1})}, \qquad p_{k-s-1,\,j} \stackrel{\text{def}}{=} \frac{\det(a^*_{k-s-2},\ a_j)}{\det(a^*_{k-s-2},\ a^*_{k-s-1})}.$$

Последняя строка формул демонстрирует тот факт, что для нахождения сплайн-вэйвлетных разложений требуется вычислить большое число определителей указанных матриц. А далее полученные значения используются для нахождения матриц больших размерностей: матрицы, представляющей калибровочные соотношения, и связанной с нею матрицы продолжения, размерность которых ($N–s–2 \times N–s–2$), что также позволяет применить методы для распараллеливания больших массивов

Узлы сетки представляют независимые данные или блоки данных. Система функций $\{\omega^*_j\}_{j\in J'_{N-s-3}}$ также независима в том смысле, что разные функции могут одновременно выполняться на различных узлах (понятно, что время их выполнения должно мало отличаться друг от друга).

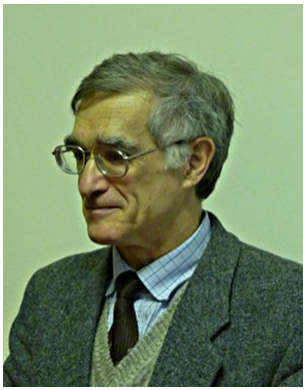
Следует уточнить ранее сделанное замечание об особом представлении матриц в сплайн-вейвлетном разложении: эти матрицы содержат большое количество нулей и ненулевые элементы на нескольких диагоналях, поэтому справедливо хранить такие матрицы в специальном виде, предварительно преобразовав их индексацию (т. е. хранить только элементы диагоналей). Кроме этого, хранимые элементы матрицы предпочтительно использовать блоками одинаковой величины для работы на различных процессорах.

Все вышеизложенное говорит о том, что применение различных приемов и способов распараллеливания должно дать ускорение выполнения задачи на больших размерностях [3].

Литература

1. *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. БХВ-Петербург, 2002. С. 602.
2. *Демьянович Ю. К., Мирошниченко И. Д.* Негладкие сплайн-вэйвлетные разложения на отрезке // Проблемы математического анализа. 2012. Вып. 63. С.23–40.
3. *Бурова И. Г., Демьянович Ю. К., Евдокимова Т. О., Иванцова О. Н., Мирошниченко И. Д.* Параллельные алгоритмы. Разработка и реализация. СПб., 2012. С. 391.

Методы хранения и поиска информации



Новиков
Борис Асенович

д.ф.-м.н.
профессор кафедры информатики СПбГУ

ЭКСПЕРИМЕНТАЛЬНОЕ СРАВНЕНИЕ АЛГОРИТМОВ РАЗДЕЛЕНИЯ ВЕРШИН В *R*-ДЕРЕВЕ НА РАЗЛИЧНЫХ ДАННЫХ

Ерохин Г. А.

george.erokhin@gmail.com

Чернышев Г. А.,

ассистент кафедры информатики;
chernishev@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В данной работе экспериментально оцениваются три метода разделения вершин в *R*-деревьях. Эти методы оцениваются в контексте использования указанной структуры данных для решения задачи построения многомерного индекса с поддержкой транзакций. Операции разделения уделено особое внимание из-за частого ее исполнения при добавлении элементов и, соответственно, ее определяющего влияния на время построения индексов. Еще более серьезное влияние она оказывает на качество разбиения элементов и производительность поисковых запросов. В нашем исследовании мы сравниваем субквадратичные алгоритмы разделения вершин на нескольких синтетических наборах данных, имеющих равномерное и нормальное распределения.

Введение

Одной из наиболее популярных структур данных, используемых для индексирования многомерных объектов в СУБД, является *R*-дерево [6]. Индексирование позволяет значительно сократить время исполнения запросов на поиск данных и повысить производительность системы в целом.

Согласно определению [7] *R*-дерево — это древовидная структура данных, заданная парой (m, M) со следующими свойствами:

- Каждый лист может содержать до M записей, минимально $2 \leq m \leq M/2$. Каждая запись в листе представлена в форме (mbr, oid), где mbr это минимальный ограничивающий прямоугольник, а oid — идентификатор объекта.
- Количество записей хранящихся во внутреннем узле также должно принадлежать [m; M]. Каждая запись в узле представляет собой пару (mbr, p), где p — указатель на ребенка узла, а mbr содержит в себе mbr ребенка.
- Дерево сбалансировано — все листья находятся на одном уровне.

Под минимальным ограничивающим прямоугольником понимается минимальный многомерный прямоугольник, содержащий рассматриваемые объекты.

Алгоритмы выполнения основных операций с данным деревом описаны в [4, 7]. Для нас наиболее интересна операция вставки элемента, а именно случай, когда происходит переполнение листа-кандидата. Тогда следует разделить вершину на несколько, что представляет собой сложную задачу.

В классической статье, где было предложено *R*-дерево [4], автор рассмотрел три типа алгоритмов разделения вершины, различающихся качеством получаемого разбиения и временной сложностью. Эта статья породила целую цепочку исследований на данную тему, некоторые из которых мы рассмотрим.

Формально задача «хорошего» разделения формулируется следующим образом: разделить прямоугольники так, чтобы при поисковом запросе необходимо было проверять как можно меньше прямоугольников-потомков одного уровня (заметим, что в определении дерева отсутствует требование попарного непересечения многоугольников, т. е. даже при поиске точки может требоваться просмотр нескольких путей до листьев). В статье [3] были рассмотрены следующие критерии, определяющие качество разделения вершины:

1. площадь прямоугольника, соответствующего вершине, минус общая площадь прямоугольников, соответствующих её детям;
2. площадь пересечения прямоугольников вершин, получившихся в результате разделения;
3. заполненность вершины;
4. соотношение объем/периметр прямоугольника.

В линейном алгоритме Гуттмана учитывается только первый критерий, в алгоритме Ан и Тань в первую очередь минимизируется площадь перекрытия получившихся прямоугольников, и во вторую очередь площадь самих результирующих прямоугольников, то есть второй и первый критерии соответственно. В алгоритме аль-Бадарне рассматриваются только расстояния между вершинами прямоугольников, что примерно пропорционально периметру объемлющего прямоугольника, то есть можно сказать, что оптимизируется четвертый критерий. Третий критерий никак не учитывается в алгоритме Ан и Тань и в некоторой степени поддерживается с помощью константы m, обозначающей минимальное количество элементов в узле дерева, в алгоритмах Гуттмана и аль-Бадарне.

Кроме структуры данных для построения транзакционного индекса необходим еще и протокол обеспечения корректности параллельного доступа к дереву для защиты от аномалий [8]. За основу нами был выбран GiST (Generalized Search Tree) [5] — алгоритм индустриального уровня, использующийся, например, в системе PostgreSQL [9], ввиду его универсальности для различных вариантов *R*-дерева. Он накладывает определенные правила и ограничения на работу с деревьями.

В данной работе мы исследуем несколько алгоритмов разделения вершин, имеющих малую временную сложность (доквадратичную по M)

в применении к задаче построения транзакционного индекса, основанного на GiST. В нашей постановке задачи требование малой временной сложности диктовалось ограничениями на время построения индекса. В следующих частях мы рассмотрим возможные альтернативы к разделению вершин, вопрос увязывания транзакционного компонента и дерева, а также эксперименты.

Обзор подходов

Гуттман [4] предложил три метода, один из которых был линеен по времени. Данный метод мы возьмем за основу при сравнении. Вкратце его идею можно описать как нахождение двух наиболее удаленных друг от друга точек и наращивание прямоугольников вокруг них так, чтобы увеличение объема было минимально на каждом шаге (при рассмотрении очередной точки). Недостаток этого алгоритма состоит в том, что как только один из улов заполнен наполовину, мы помещаем все оставшиеся прямоугольники во второй, что может привести к плохому распределению.

Ещё один линейный алгоритм был предложен авторами Ан и Тань в работе [2] и состоит в следующем: по каждому измерению разбиваем все прямоугольники на две группы: ближайшие к минимальной стороне объемлющего прямоугольника и ближайшие к максимальной стороне. Затем из всех разбиений выбираем наиболее равномерное, то есть такое, в котором в обеих частях примерно одинаковое количество записей. Если таких разбиений несколько, берем то, в котором объемлющие прямоугольники для двух частей имеют наименьшую площадь пересечения, если даже в этом случае оказалось несколько вариантов, выбираем вариант с наименьшей площадью объемлющих прямоугольников.

Третий алгоритм, выбранный для сравнения, описан аль-Бадарне в статье [1] и состоит в следующем: пусть m — минимальное количество детей узла дерева. Сначала мы, как и в линейном алгоритме Гуттмана, выбираем два прямоугольника, которые станут основой нашего разбиения. Отличие состоит в том, что на этот раз мы выбираем самый левый нижний и самый правый верхний, назовем их MIN и MAX соответственно. Затем сортируем все прямоугольники по возрастанию расстояния до MIN, после чего выбираем из них первые m и помещаем в первый узел, оставшиеся сортируем по возрастанию расстояния до MAX и помещаем первые m во второй узел. Все остальные распределяем по узлам в зависимости от расстояния до MIN и MAX.

Есть и другие методы разделения вершин, однако мы их не рассматриваем по различным причинам:

$R+$ деревья (и их алгоритм разбиения) не работают с GiST [5], кроме того их высота может быть больше [2].

Алгоритмы для деревьев типа *R** сложны в реализации и требовательны к ресурсам (необходимо удалять и вставлять вершину заново) [7], что может понизить их привлекательность для транзакционных систем.

Гильбертово дерево также требует дополнительной работы по обеспечению упорядоченности среди узлов.

Эксперименты

В наших экспериментах мы не рассматриваем впрямую качество полученных разбиений, но рассматриваем пропускную способность системы (измеряемую в количестве выполненных запросов в секунду) как оценку качества алгоритма. Прямые оценки качества алгоритмов, такие как степень пересечения полученных прямоугольников, измерялись, например, в работе [2]. Также, наши эксперименты включают в себя измерение времени построения индекса.

В нашем прототипе индекса отсутствовала функциональность по восстановлению и журналированию, он был ориентирован на нахождение в памяти целиком (in-memory indexing), замками защищались отдельные записи, а не страницы. Был использован уровень изоляции read committed, удаление осуществлялось с помощью пометки значения как удаленного.

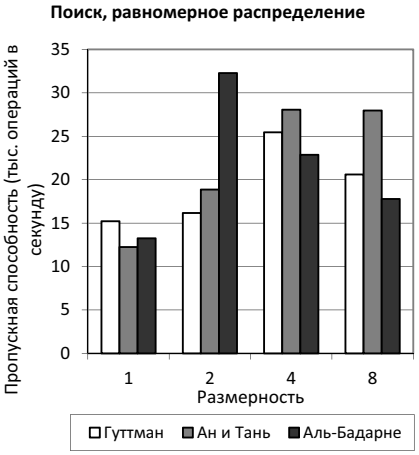
Данные и запросы были взяты из задачи на соревнования ACM SIGMOD Programming Contest 2012 [10]. Испытательный стенд (генераторы данных и сборщик статистики) был предоставлен организаторами. Эксперименты проводились на сгенерированных данных объемом 32 Мб размерности 1, 2, 4 и 8, с равномерным и нормальным распределением (по 3 набора, равномерное распределение, нормальное распределение, и смешанное, в котором половина измерений подчинялись равномерному, а половина нормальному распределению значений). Эксперименты проводились в 2 фазы: сначала индекс заполнялся одним потоком и замерялось время его построения, затем в течение 60 секунд восемью потоками (по количеству логических ядер) осуществлялись запросы и замерялось количество операций в секунду. В каждом случае проводилось по 4 испытания и вычислялось среднее значение, которое и было представлено на графике.

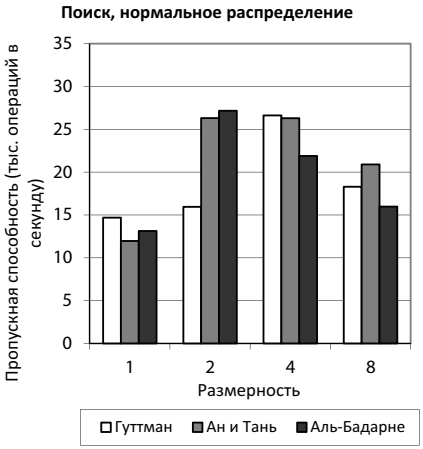
Конфигурация тестового оборудования:

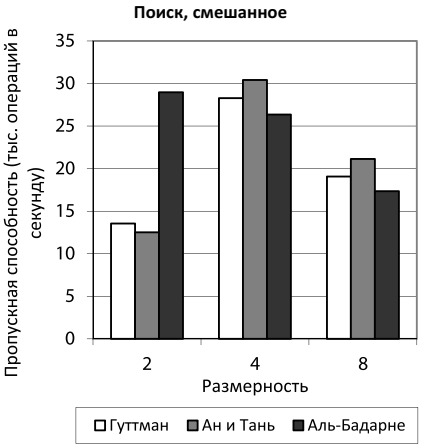
Процессор	Intel Core i7-2670QM 2.20 GHz
Hyper-Threading	Enabled
ОЗУ	6 Гб
Размер L3 кэша	6 Мб
Размер L2 кэша	256 Кб на ядро
Размер L1 кэша	(32 + 32) Кб на ядро
Операционная система	GNU/Linux, kernel 3.0.0-12

Результаты

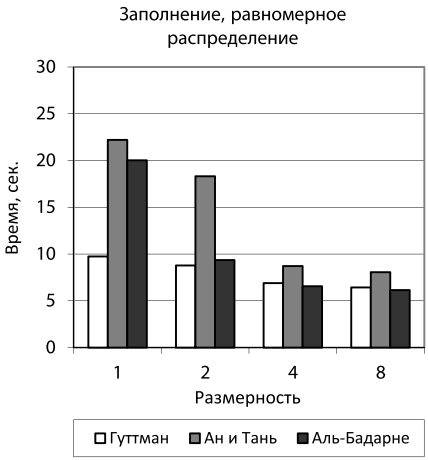
Пропускная способность при исполнении запросов на диапазон:

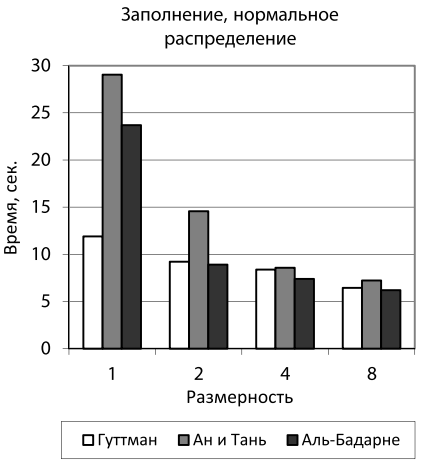


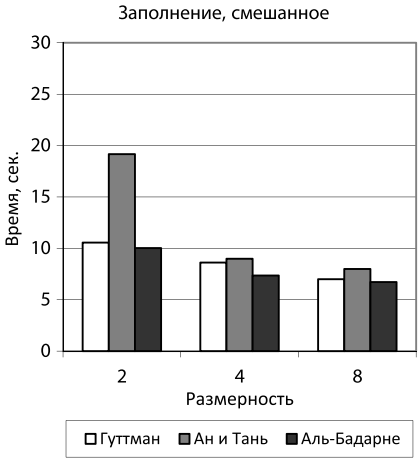




Скорость заполнения индекса:







Повышение пропускной способности и снижение времени построения индекса с увеличением размерности объясняется тем, что тесты работают с данными фиксированного объема, при этом размер одной записи пропорционален размерности. Соответственно чем больше размерность, тем меньше записей вставляется в индекс в одном тесте, что и приводит к улучшению производительности. Однако этот факт не влияет на чистоту эксперимента, так как при фиксированной размерности алгоритмы работают с одинаковым объемом данных. В то же время, каждый алгоритм является линейным по количеству измерений, поэтому при больших размерностях уменьшение количества вставляемых записей не так сильно влияет на скорость построения. К тому же с повышением размерности ухудшается качество прямоугольников, получающихся в результате разделения, что приводит к ухудшению дерева в целом, и как следствие поиск работает медленнее.

Заключение

В качестве итога можно сказать что алгоритм, предложенный Ан и Тань, является наилучшим с точки зрения качества построенного разбиения, однако он является довольно дорогостоящим по времени. Необходимо отметить алгоритм аль-Бадарне, который показал наилучшие результаты среди всех на данных размерности 2 при всех распределениях. Также, данный алгоритм является одним из самых быстрых на этапе заполнения индекса.

Литература

1. *Amer F. Al-Badarneh, Qussai Yaseen*, and *Ismail Hmeidi*. 2010. A new enhancement to the R-tree node splitting. J. Inf. Sci. 36, 1 (February 2010), 3–18.
2. *Chuan-Heng Ang* and *T. C. Tan*. 1997. New Linear Node Splitting Algorithm for *R*-trees. In Proceedings of the 5th International Symposium on Advances in Spatial Databases (SSD»97), Michel Scholl and Agnès Voisard (Eds.). Springer-Verlag, London, UK, 339–349.
3. *Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger*. 1990. The *R**-tree: an efficient and robust access method for points and rectangles. In Proceedings of the 1990 ACM SIGMOD international conference on Management of data (SIGMOD»90). ACM, New York, NY, USA, 322–331.
4. *Antonin Guttman*. 1984. R-trees: a dynamic index structure for spatial searching. In Proceedings of the 1984 ACM SIGMOD international conference on Management of data (SIGMOD»84). ACM, New York, NY, USA, 47–57.
5. *Joseph M. Hellerstein, Jeffrey F. Naughton*, and *Avi Pfeffer*. 1995. Generalized Search Trees for Database Systems. In Proceedings of the 21th International Conference on Very Large Data Bases (VLDB «95), Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 562–573.
6. Ling Liu, M. Tamer Özsu (Eds.): Encyclopedia of Database Systems. *R*-Tree (and Family). 2453–2459. Springer US 2009, ISBN 978-0-387-35544-3, 978-0-387-39940-9.
7. *Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos,* and *Y. Theodoridis*. 2005. R-Trees: Theory and Applications (Advanced Information and Knowledge Processing). Springer-Verlag New York, Inc., Secaucus, NJ, USA.
8. *Gerhard Weikum* and *Gottfried Vossen*. 2001. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
9. http://www.postgresql.org/docs/8.1/static/gist.html, [дата просмотра 18.04.2012]
10. http://wwwdb.inf.tu-dresden.de/sigmod2012contest/#task-overview, [дата просмотра 18.04.11]

РЕАЛИЗАЦИЯ УРОВНЯ ИЗОЛЯЦИИ READ COMMITTED ДЛЯ ДРЕВОВИДНЫХ СТРУКТУР ДАННЫХ[1]

П. В. Федотовский

pavel.fedotovsky@math.spbu.ru

Г. А. Чернышев

ассистент кафедры информатики;
chernishev@gmail.com

К. К. Смирнов

kirill.k.smirnov@math.spbu.ru

Санкт-Петербургский государственный университет

Аннотация: В данной работе мы исследуем метод обеспечения корректности параллельного доступа к древовидным структурам данных для уровня изоляции read committed. Для экспериментальной оценки был реализован прототип многомерного индекса с поддержкой транзакций. Мы исследуем производительность разработанного метода для индексов, построенных по модели GiST.

Введение

Согласно [1], индексом называется структура данных, и метод размещения данных в таблице, что индексируется. Данный объект позволяет значительно ускорить время выполнения различных запросов в зависимости от типа индекса. Например, использование B-деревьев позволяет снизить оценку временной сложности поиска элемента в таблице с линейной до логарифмической. Существует два основных подхода к построению индексов [1]: использовать хеш-таблицы или деревья поиска. В настоящем исследовании мы сконцентрируемся на последнем типе.

Кроме реализации структуры данных и ее интеграции с исходными данными (таблицей), необходимо также обеспечить корректность параллельного доступа для нескольких потоков. Для решения этой задачи применяются различные методы, например для B+ деревьев существует несколько классических алгоритмов [2]. Однако они слабо применимы для случая многомерных индексов и, соответственно, многомерных деревьев поиска. Индекс такого типа строится для набора атрибутов и позволяет осуществлять поиск не только по отдельным значениям, но сразу по кортежам из нескольких значений. Такие индексы широко применяются, например, для построения геоинформационных систем, CAD систем (computer-aided design), мультимедиа СУБД [3].

[1] Работа частично поддержана РФФИ, грант № 10-07-00156.

Корректность при параллельном доступе обеспечивается за счет выбора определенного уровня изоляции. В свою очередь, уровень изоляции может определяться как набор допустимых аномалий транзакционного доступа [2]. Различают четыре уровня изоляции: read uncommitted, read committed, repeatable read, serializable (перечислены в порядке усиления требований корректности). Последний из этих уровней обеспечивает полную корректность, однако им не всегда пользуются на практике из-за его «дороговизны» [4]. Вместо него часто используется уровень изоляции read committed, согласно [5] можно получить выигрыш в скорости в 2.5–3 раза.

Одной из популярных моделей обеспечения транзакционного доступа в деревьях для многомерных индексов является GiST (Generalized Search Tree). Данная модель позволяет наладить транзакционный доступ при работе с любым деревом, отвечающим определенным требованиям [6]. Она является достаточно популярным решением и используется, например в PostgreSQL [8].

В нашей работе мы, с помощью построенного прототипа многомерного транзакционного индекса, исследуем производительность алгоритма параллельного доступа с уровнем изолированности read commited. В качестве структуры данных для индекса было использовано *R*-дерево [3], в индексе отсутствовало журналирование и восстановление, и прототип был ориентирован на нахождение в памяти целиком (in-memory indexing). Замки брались не на страницы, но на отдельные записи.

Предложенный алгоритм

Как известно, для обеспечения уровня изоляции read committed необходимо предотвратить две аномалии: грязное чтение и потерянное обновление [2]. Для этого используется менеджер блокировок и специальный механизм меток, гарантирующий, что транзакция не «увидит» незафиксированных данных.

Основное отличие от алгоритма, описанного в [7], состоит в том, что замки берутся только на сами данные. Каждая запись хранит идентификатор текущей WR транзакции и состояние. Всего состояний пять:

- ProcessInsert — запись вставляется.
- ProcessDelete — запись удаляется.
- ProcessUpdate — запись обновляется (изменяется только значение, ключ остается прежним).
- Deleted — запись удалена.
- Valid — запись в текущий момент не изменяется.

Алгоритм удаления — логический, то есть выставляется флаг «запись удалена». Рассмотрим алгоритмы исполнения основных операций:

- Получить записи (GetRecords).

Этот метод получает итератор на записи, которые удовлетворяют запросу (в данном случае подходят все записи, содержащиеся в n-мерном прямоугольнике). Сначала происходит поиск необходимых записей с помощью древовидной структуры данных (при этом не учитывается текущее состояние вершины). Затем пользователь последовательно вызывает метод GetNext для получения очередной записи. При этом проверяется ее текущее состояние. Опишем дальнейшие действия в зависимости от состояния:

- ProcessInsert — запись пропускается, так как транзакция, ее вставляющая, еще не была зафиксирована. Происходит переход к следующей подходящей под запрос записи (если ее нет, то пользователь информируется об этом).
- ProcessDelete — пользователь получает запись, поскольку изменяющая ее транзакция еще не была зафиксирована или отменена (иначе состояние было бы Deleted или Valid соответственно).
- ProcessUpdate — возвращается старое значение записи.
- Deleted — запись была удалена, происходит переход к следующей записи (см. пункт ProcessInsert).
- Valid — пользователь получает запись, так как в текущий момент ее не изменяет ни одна транзакция, и она не удалена.

К сожалению, формат работы не позволяет детально изложить реализацию всех операций, поэтому ниже приводится краткое описание остальных методов.

- Вставка (Insert).
  - Запись помечается состоянием ProcessInsert и текущим идентификатором транзакции.
  - Непосредственная вставка в древовидную структуру данных.
- Обновление/удаление (Update/Delete).
  - Нахождение в древовидной структуре нужных записей.
  - Получение блокировок на вершины у менеджера блокировок.
  - Изменение состояние вершины.

Данная работа является работой практического типа, корректность алгоритма подтверждалась unit-тестами, однако ниже кратко объясняется отсутствие аномалий многопоточного доступа.

- Потерянное обновление (Lost Update).
  Возникает в случае, когда две транзакции изменяют одну и ту же запись. Это предотвращается с помощью менеджера блокировок. Каждый раз при изменении записи транзакция запрашивает блокировку, что гарантирует изменение записи только одной транзакцией.
- «Грязное» чтение (Dirty Read).
  Возникает в случае, если транзакция читает данные, которые еще не были зафиксированы. Для борьбы с данной аномалией используется предло-

женный механизм маркировки записей. Каждый раз при чтении записи анализируется ее состояние. Если происходит изменение, то пользователь получает старую версию записи, которая уже была зафиксирована. Чуть более детально этот процесс описан выше при рассмотрении операции «получить записи».

Об измерениях

Данные, запросы и их вероятности были взяты из постановки задачи на соревновании ACM SIGMOD Programming contest 2012 [9]. Испытательный стенд (генераторы данных и сборщик статистики) был предоставлен организаторами. Эксперименты проводились на сгенерированных данных различных объемов (32, 64 и 128 Мб) размерности 2, 4 и 8, с равномерным распределением. Одновременно исполнялось 8 потоков (по количеству логических процессоров), все измерения проводились на уже построенном индексе. В каждом случае проводилось по 3 испытания и вычислялось среднее значение.

В своих измерениях мы не используем шаблоны, то есть запросы, содержащие весь диапазон по какому — либо из измерений.

Конфигурация тестового оборудования представлена в таблице 1, результаты экспериментов отображает диаграмма 1.

Таблица 1

Конфигурация тестового оборудования

Процессор	Intel Core i7-2630QM 2.00 GHz
Hyper-Threading	Enabled
ОЗУ	6 Гб
Размер L3 кэша	6 Мб
Размер L2 кэша	256 Кб на ядро
Размер L1 кэша	(32 + 32) Кб на ядро
Операционная система	GNU/Linux, kernel 2.6.38-8-generic

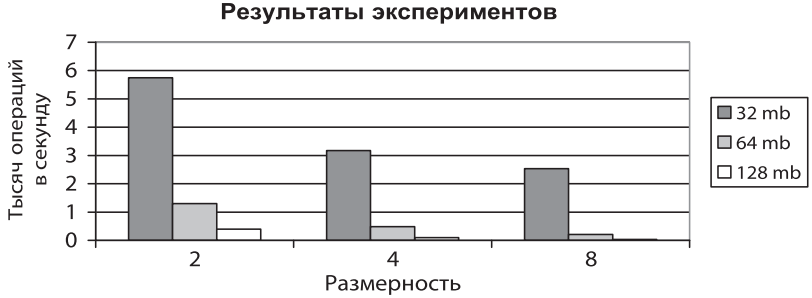


Диаграмма 1. Результаты экспериментов

Можно заметить, что производительность системы сильно падает при увеличении, как размера, так и размерности индекса. Также этот график демонстрирует, что текущий прототип справляется с задачей многомерного индексирования, в дальнейшем планируются работы над его улучшением.

Заключение

В данной работе был представлен алгоритм для обеспечения уровня изоляции read committed. С его помощью нами был построен прототип многомерного транзакционного индекса с данным уровнем изоляции. Прохождение unit-тестов организаторов соревнования ACM SIGMOD Programming Contest 2012 [9] свидетельствует в пользу корректности нашей реализации. Это первая стадия в нашем исследовании, предварительный результат которой — корректный алгоритм и функционирующая реализация. В дальнейшем мы планируем повышать производительность нашей системы, а так же представить формальное доказательство корректности применяемого алгоритма.

Литература

1. Ling Liu, M. Tamer Özsu (Eds.): Encyclopedia of Database Systems. Index Tuning. 1433–1435. Springer US 2009, ISBN 978-0-387-35544-3,978-0-387-39940-9.
2. *Gerhard Weikum* and *Gottfried Vossen*. 2001. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
3. Ling Liu, M. Tamer Özsu (Eds.): Encyclopedia of Database Systems. *R*-Tree (and Family). 2453–2459. Springer US 2009, ISBN 978-0-387-35544-3, 978-0-387-39940-9.
4. Ling Liu, M. Tamer Özsu (Eds.): Encyclopedia of Database Systems. SQL Isolation Levels. 2761–2762. Springer US 2009, ISBN 978-0-387-35544-3,978-0-387-39940-9.
5. *Paul M. Bober* and *Michael J.* Carey. 1992. On Mixing Queries and Transactions via Multiversion Locking. In Proceedings of the Eighth International Conference on Data Engineering, Forouzan Golshani (Ed.). IEEE Computer Society, Washington, DC, USA, 535–545.
6. *Joseph M. Hellerstein, Jeffrey F. Naughton*, and *Avi Pfeffer*. 1995. Generalized Search Trees for Database Systems. In Proceedings of the 21th International Conference on Very Large Data Bases (VLDB «95), Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 562–573.
7. *Marcel Kornacker, C. Mohan*, and *Joseph M. Hellerstein*. 1997. Concurrency and recovery in generalized search trees. In Proceedings of the 1997 ACM SIGMOD international conference on Management of data (SIGMOD «97), Joan M. Peckman, Sudha Ram, and Michael Franklin (Eds.). ACM, New York, NY, USA, 62–72.
8. http://www.postgresql.org/docs/8.1/static/gist.html [дата просмотра 18.04.2012]
9. http://wwwdb.inf.tu-dresden.de/sigmod2012contest/#task-overview [дата просмотра 18.04.11]

МОДЕЛИ СТОИМОСТИ ПРИБЛИЖЕННЫХ АЛГОРИТМОВ ДЛЯ ОПЕРАЦИЙ СИНТЕЗА

О. А. Долматова

студентка кафедры информатики;
oxana.dolmatova@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В этой работе представлены модели стоимости, для приближенных алгоритмов, которые позволяют не только выполнить запрос, но и решают задачу ранжирования. Предложенные модели стоимости дают возможность управлять соотношением затраченных ресурсов к качеству получаемого результата. Для этого для каждого приближенного алгоритма выведена формула, связывающая входящий ресурс, в качестве которого взято время, и качество результата, измеряемое полнотой.

Введение

Одним из важных составляющих системы, выполняющей запросы, является оптимизатор. Он является неотъемлемой частью, в особенности, когда работа ведется над сложными запросами и с большим объемом данных. Задача оптимизатора состоит в том, чтобы выбрать алгебраическое выражение минимальной стоимости среди других аналогичных выражений.

Абстрактное понятие стоимости может включать в себя различные меры производительности выполнения запросов. Как правило, наиболее важными являются время выполнения, количество I/O или, в распределенной мобильной среде, использованная энергия. В данной работе будет рассмотрено время выполнения.

В настоящее время разработано большое количество оптимизаторов, которые в основном опираются на модели стоимости для точных алгоритмов выполнения запросов. Мы же остановимся подробнее на приближенных алгоритмах.

Так как выполнение запроса приближенное, качество результата может варьироваться. Главная задача нашей работы — создание модели стоимости, способной поддерживать компромисс между стоимостью и качеством результата. На основе нашей модели оптимизатор может либо обеспечить максимально возможное качество для данной стоимости, либо свести к минимуму стоимость обеспечения желаемого качества.

Связанные работы

Точные алгоритмы, взятые за основу для написания приближенных, с замечаниями и некоторыми оценками подробно рассмотрены в [2].

Оптимизация запросов стала необходима в середине 70-х годов, с момента появления декларативных языков высокого уровня. В связи с этим стали развиваться исследования моделей стоимости для различных алгоритмов, операций и сред выполнения запросов.

Модель стоимости, основанная на оценке операции выборки и мощности, представлена в [1] для выбранного набора операций: объединение, пересечение, разность, соединения, слияния и выборки.

Отсутствие алгебраических эквивалентностей подталкивает к развитию оптимизации, основанной на приближенных алгоритмах и управлении соотношением между производительностью и качеством.

Обширный обзор связанных работ находится в [4]. Так же последняя статья представляет набор «top k» алгоритмов, основанных на вероятностных гарантиях. В отличие от данной работы, в [4] делается упор на получение конечного результата с заданным качеством, без управления соотношением производительность/качество.

Алгоритмы выполнения запросов

Для выполнения запросов и ранжирования рассмотрены канонические алгоритмы [2, 3], с предположением об ограничении количества входящий потоков оценок до двух.

Первый приближенный алгоритм — алгоритм Рональда Фейгана [2], ориентированный на обработку потоков, ниже будет обозначаться FA_S. Второй алгоритм — алгоритм NRA [2], так же ориентированный на обработку потоков, будет обозначаться NRA_S. Вследствие того, что система может быть распределенной, и оценки поступают потоками из двух независимых друг от друга источников, дорогостоящий, а порой и невозможный, random access убран из реализации алгоритмов и заменен последовательным просмотром. Оба алгоритмы были дополнены возможностью задания порогов для оценок и носят названия FA_ST и NRA_ST соответственно.

В качестве агрегирующих функций были рассмотрены пересечение и объединение.

Модели стоимости

Представим модели стоимости для разных операций в случае, когда мы можем регулировать входные ресурсы и качество выходящего результата снаружи системы.

Стоимость зависит от различных типов входных данных, таких как энергия, объем входных и выходных данных, время выполнения запроса. Мы будем использовать последнее, как наиболее удобный входной параметр. Говоря же о выходном результате, мы использовали абстрактное понятие качества. В данной работе определим абстрактное качество как полнота. Так же сделаем предположение о входящих оценках, будем считать, что все оценки

независимы, нормированы и имеют равномерное распределение в нашей модели стоимости.

Остановимся подробнее на входящих параметрах (все нижеприведенное мы считаем известным):

t_0 — время, отведенное на выполнение данного запроса;

k_0 — желаемое количество оценок в результате;

a — количество оценок, для рассматриваемого случая равно удвоенному количеству объектов;

thr_1 — порог для левой оценки, в случае, если алгоритм с порогом;

thr_2 — порог для правой оценки, в случае, когда алгоритм с порогом;

c — системные параметры, время затрачиваемое на:

- чтение/запись;
- сравнение;
- поиск (при условии использования расширяемого хэша).

Не умаляя общности, мы можем сказать, что все системные параметры равны одной константе c.

Для каждой из приближенных моделей мы определим два граничных значения входящих ресурсов — минимальное и максимальное время. Минимальное время необходимо для получения хотя бы одного объекта в результате, то есть, если данное время t_0 будет меньше минимального, то не имеет смысла выполнять запрос. Максимальное же время — время, за которое система выдаст все k_0 желаемых объектов с оценками, и при увеличении времени, данного системе, качество результата не улучшится.

Остановимся на формулах подробнее. Для начала нам понадобятся переменные, которые позже будут оценены исходя из особенностей и реализации алгоритмов и выражены через входящие параметры:

1. t — ожидаемое количество оценок, которое нужно прочитать для получения конечного результата, содержащего k_0 объектов
2. ks — ожидаемое количество оценок, нужных для того, чтобы мощность пересечение прочитанных множеств правых и левых оценок была равна k_0
3. ns — ожидаемое количество оценок, нужных для того, чтобы мощность пересечения прочитанных множеств правых и левых оценок была равна t
4. ts — количество оценок, удовлетворяющее порогу, в случае, когда алгоритм с порогом

Мы будем использовать математическое ожидание, независимость оценок и равномерное распределение для вычисления первого, второго и третьего параметра. Для последнего же нужно только посчитать площадь под графиком равномерного распределения с заданными параметрами.

Теперь несложно написать формулы, выражающие зависимость затраченного времени от входящих параметров:

FA_S:

$$5ct + ca + ct \log t = \text{time}.$$

FA_ST:

$$2c * \min(ts, ns) + c(ns + t) + ct \log t = \text{time}.$$

NRA_S:

$$2ct(2 + t) + ct \log t * \frac{(ns - ks)}{2} + ca = \text{time}.$$

NRA_ST:

$$2c * \min(ts, ns) + cns + 2ct^2 + ct \log t * \frac{(\min(ts, ns) - ks)}{2} = \text{time}.$$

Рассмотрим немного подробнее, как получились формулы.

FA: слагаемые отражают запись оценок, поиск места для вставки оценки, вычисление агрегированной оценки, считывание потоков, сортировку соответственно.

NRA: слагаемые учитывают запись оценок, поиск места для вставки, вычисление worst case score и best case score, сортировку, считывание потоков соответсвенно.

Алгоритмы с порогом: отличие — наличие функции минимума. Они ограничивают количество оценок, доступных для просмотра порогом сверху.

Так как все переменные, входящие в формулы, выражаются через входящие параметры, то сейчас мы получили зависимость времени от заданного желаемого количества объектов в результате. Но мы хотим, чтобы по заданному времени можно было получить количество оценок, которое система сможет выдать, поэтому преобразуем зависимости, имея в виду, что time это t_0, а количество оценок — k. После оценки всех переменных и упрощения получаются такие выражения:

FA_S:

$$\left(\frac{15}{4}c + \frac{3}{4}c \log a - \frac{3}{2}c\right)k + \frac{9}{4}ca + \frac{1}{4}ac(\log a - 2) = t_0.$$

FA_ST:

1) Эта формула имеет место, когда $k < a * (2 - thr1 - thr^2)/3$

$$\left(\frac{21}{4}c + \frac{3}{4}c \log a - \frac{3}{2}c\right)k + \frac{1}{4}ca + \frac{1}{4}ac(\log a - 2) = t_0.$$

2) А эта когда $k > a * (2 - thr1 - thr^2)/3$

$$\left(\frac{3}{4}c \log a + \frac{3}{4}c\right)k + ca(2 - thr1 - thr2) + \frac{1}{4}ca(\log a - 2) + \frac{1}{4}ca = t_0,$$

NRA_S:

$$\left(\frac{9}{32}+\frac{9}{8}c-\frac{9}{64}\log a\right)k^2+$$
$$+\left(\frac{3}{8}ca+\frac{3}{2}c\left(2+\frac{1}{4}a\right)-\frac{3}{64}ac(\log a-2)+\frac{3}{16}\left(\frac{3}{4}\log a-\frac{3}{2}\right)ac\right)k+$$
$$+\frac{1}{2}ca\left(2+\frac{1}{4}a\right)+ca+\frac{3}{64}a^2c\left(\log a-2\right)=t_0.$$

NRA_ST:

1) Эта формула имеет место, когда $k > a*(1–4*thr1–4*thr^2)/9$

$$\left(\frac{9}{8}c+\frac{3}{4}-\frac{3}{8}\log a\right)k^2+$$
$$+\left(\frac{3}{4}ca+\frac{9}{8}c-\frac{1}{8}ac(\log a-2)+\frac{1}{4}c\left(\frac{3}{4}\log a-\frac{3}{2}\right)a(2-thr1-thr2)\right)k+$$
$$+ca(2-thr1-thr2)+\frac{1}{16}ca^2(\log a-2)(2-thr1-thr2)+\frac{1}{8}ca^2+\frac{3}{8}ca=t_0.$$

2) А эта когда $k < a*(1–4*thr1–4*thr^2)/9$

$$\left(\frac{3}{64}c\log a-\frac{3}{32}+\frac{9}{8}c\right)k^2+$$
$$+\left(\frac{1}{64}ca(\log a-2)+\frac{3}{16}c\left(\frac{3}{4}\log a-\frac{3}{2}\right)a+\frac{27}{8}c+\frac{3}{4}ca\right)k+$$
$$+\frac{9}{8}ca+\frac{1}{8}ca^2+\frac{3}{64}ca^2(\log a-2)=t_0.$$

Случаи в формулах для алгоритмов с порогом возникают из-за того, что раскрыта функция минимума.

Из вышеприведенных формул нетрудно выразить зависимость k от t_0, но мы воздержимся от приведения этого в данной работе. Для получения граничных оценок достаточно вместо k подставить 1 в случае минимального значения и k_0 в случае максимального. И если нам не нужно много, то достаточно по этим двум значениям построить функцию от k, используя линейную интерполяцию.

В итоге мы имеем k_0 — желаемое количество объектов и k — возможное за заданное время. Рассмотрим соотношение между этими значениями и получим полноту. Тогда, задавая различное время ожидания и желаемый результат, мы сможем управлять производительностью системы.

Заключение

В этой работе были представлены несколько моделей стоимости для приближенных алгоритмов выполнения запросов. Интуитивно понятно, что, чем меньше входных ресурсов используется — тем менее аккуратным будет полученный результат, и наоборот, чем больше ресурсов — тем точнее результат. Представленные модели стоимости позволяют преобразовать интуитивные рассуждения в четкие и ясные формулы и формализуют соотношение между производительностью и конечным результатом.

Литература

1. *S. Adali, P. Bonatti, M. L. Sapino*, and *V. S. Subrahmanian*. A multi-similarity algebra. In Proc of the 1998 ACM SIGMOD intrn conf on Management of data, SIGMOD'98, pages 402–413. NY, USA, 1998. ACM.
2. *Ronald Fagin, Amnon Lotem* and *Moni Naor*. Optimal aggregation algorithms for middleware // Journal of Comp and Syst Sciences. 2003.
3. *Peter Gurský* and *Peter Vojtáš*. On top-*k* search with no random access using small memory.
4. *Martin Theobald, Gerhard Weikum* and *Ralf Schenkel*. Top-*k* query evaluation with probabilistic guarantees.

ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ В МНОГОПОТОЧНОМ ОБРАБОТЧИКЕ ТРАНЗАКЦИЙ

К. Е. Чередник
toskira@gmail.com

К. К. Смирнов
kirill.k.smirnov@math.spbu.ru

Санкт-Петербургский государственный университет

Аннотация: В данной работе изучается один из аспектов транзакционной системы, такой как использование динамической памяти. Делается краткий обзор готовых библиотек-распределителей. Строятся и анализируются журналы использования памяти на разных этапах работы системы. Проводятся эксперименты по использованию различных распределителей памяти.

Введение

Работа с динамической памятью является важной задачей при разработке приложений. Как показывает практика [6], распределитель памяти, используемый в стандартной библиотеке glibc, демонстрирует плохую производительность на конкретных классах задач — это неизбежная плата за его универсальность. Поэтому в тех случаях, где работа с памятью особенно важна, разработчики вынуждены использовать альтернативные решения, пригодные для каждого конкретного приложения.

В данной работе делается краткий обзор существующих библиотек, разбирается использование памяти многопоточным обработчиком транзакций, проводится экспериментальное сравнение производительности системы при использовании различных распределителей. В качестве изучаемой системы использовалась система многомерного индексирования в оперативной памяти, разработанная в рамках соревнования при конференции ACM SIGMOD 2012.

Описание системы

В качестве изучаемой системы использовалась система многомерного индексирования в оперативной памяти, разработанная в рамках соревнования при конференции ACM SIGMOD 2012. Система была реализована как библиотека на языке C++, динамически связываемая с тестовой программой организаторов.

Система работает в два этапа: построение индексов и поиск по ним. В качестве основных структур данных, используемых для организации индексов, использовались R-деревья и B-деревья. Поддержка одновременного доступа к индексам из разных нитей была реализована с использованием GiST.

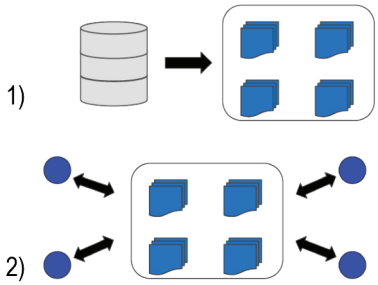


Рис. 1. Описание работы системы

Схема работы системы представлена на рисунке 1.

Обзор готовых технических решений

В настоящее время рынок программного обеспечения располагает некоторым выбором библиотек, реализующих различные алгоритмы распределения памяти. Ниже приведен краткий обзор подобных свободно распространяемых программных средств.

1. Стандартная библиотека Linux glibc использует улучшенную реализацию ptmalloc [3] в качестве многопоточного распределителя памяти. В свою очередь, ptmalloc разрабатывался на основе dl-malloc (Doug Lee Malloc) с улучшенной поддержкой многопоточности.
2. Разрабатываемый компанией Google tcmalloc (Thread-Caching Malloc) [1] демонстрирует лучшую, чем у ptmalloc, производительность при работе в многопоточной среде. В частности, в некоторых работах [2] утверждается, что использование tcmalloc вместо стандартного распределителя памяти позволяет увеличить пропускную способность транзакционной системы вдвое.
3. Jemalloc [5] используется в *BSD как системный распределитель памяти. Также является распределителем в проектах Mozilla Firefox и Facebook.
4. Hoard malloc [4] — еще один кросс-платформенный распределитель памяти, предназначенный для многопоточных приложений в многопроцессорной среде.

Для обоснованного выбора распределителя памяти необходимо изучить то, как системой используется память: какого размера блоки и с какой частотой выделяются.

Предварительно можно сформулировать две гипотезы:

1. Ожидается выделение большого числа блоков фиксированного размера при небольшой вариации самих размеров.
2. Различные нити должны демонстрировать схожее потребление динамической памяти.

Экспериментальная часть

Эксперименты проводились в два этапа: получение журнала использования памяти и сравнение различных распределителей.

Использовалась следующая программно-аппаратная конфигурация:

- HW: Intel(R) Pentium(R) D CPU 3.00GHz, 3Gb RAM.
- SW: OS GNU/Linux, kernel 2.6.38.7.

Строились 4 многомерных индекса с целочисленными координатами. Характеристики индексов приведены в таблице 1.

Таблица 1

Описание используемых индексов

Размер индекса (в Mb)	Размерность индекса	Полезные данные (в байтах)
2	3	8
1	1	8
2	4	64
4	8	64

Данные строились на двух различных типах нагрузки транзакционной системы:

- построение индексов;
- поиск по индексу, редкие обновления индекса.

Анализ журналов использования памяти

Зависимость количества выделяемых блоков от их размера, полученная при построении индекса, изображена на диаграмме 1:

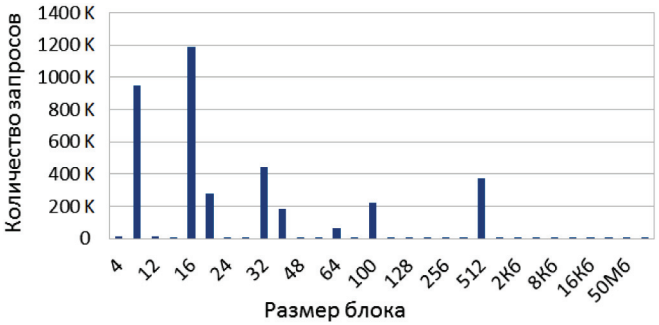


Диаграмма 1. Выделение блоков при построении индексов

Как видно на диаграмме, существенное количество запросов на выделение памяти происходит на блоки, размеры которых принадлежат довольно малому множеству (8 элементов).

Более ярко выраженная ситуация проявляется при поиске по индексу (диаграмма 2):

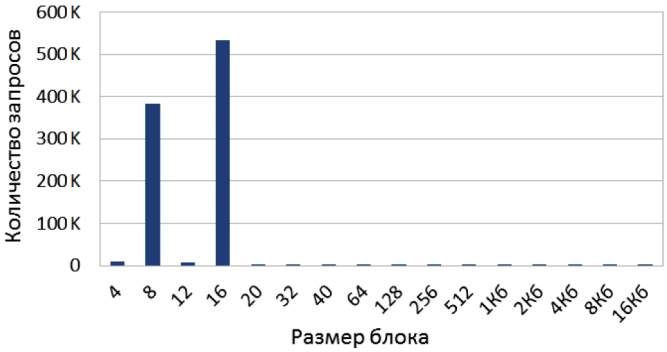


Диаграмма 2. Выделение блоков при поиске по индексам

Таким образом, экспериментально подтверждается первая гипотеза о распределении количества выделяемых блоков по размерам этих блоков.

Количества блоков, выделенных каждой нитью в отдельности, имеет распределение, схожее с изображенным на диаграмме 2. Это подтверждает вторую гипотезу о распределении выделенных блоков между нитями.

Сравнение распределителей памяти

Таблица 2

Версии рассматриваемых распределителей

Распределитель	Версия
Glibc	2.14.1
Tcmalloc	2.0
Hoard	38
Jemalloc	2.2.5

Был проведен ряд экспериментов, в ходе которых была измерена пропускная способность системы в зависимости от количества клиентских нитей и используемого распределителя памяти. Версии распределителей приведены в таблице 2. Для каждого из четырех распределителей памяти и количества нитей от 1 до 20 было проведено 10 замеров, по которым были вычислены итоговые значения с доверительным интервалом 95%. На диаграмме 3 изображена производительность системы на этапе построения индексов.

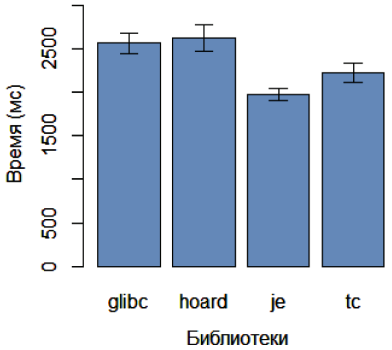


Диаграмма 3. Построение индекса

Разница между распределителями glibc, hoard статистически незначима, jemalloc демонстрирует лучшую производительность, tcmalloc занимает промежуточную позицию. При этом разброс значений для jemalloc меньше, чем у остальных.

На диаграммах 4–8 отражена пропускная способность системы при поисковых запросах.

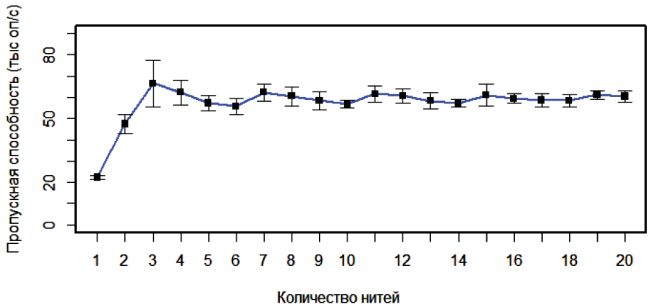


Диаграмма 4. Glibc

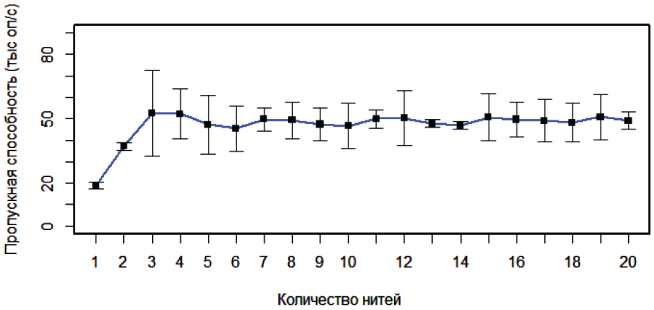


Диаграмма 5. Hoard

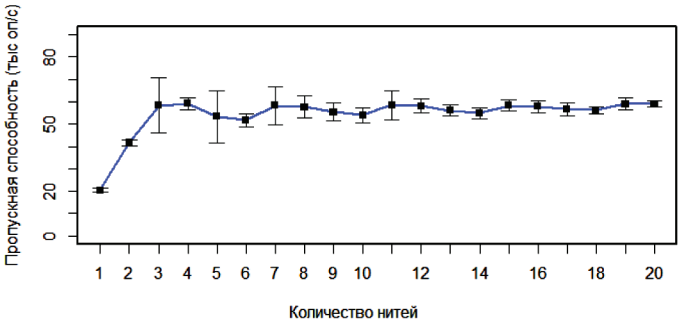


Диаграмма 6. Jemalloc

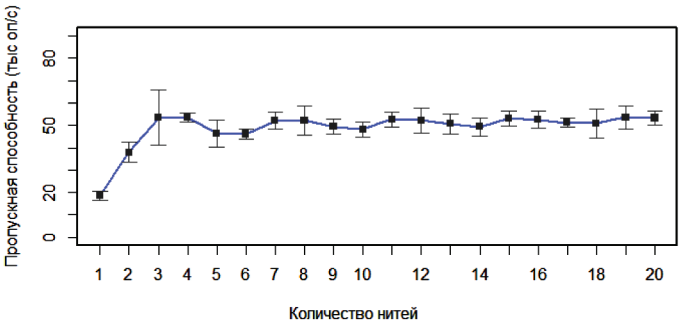


Диаграмма 7. Tcmalloc

Основываясь на этих измерениях, можно предположить, что при увеличении числа нитей пропускная способность системы при использовании конкретного распределителя стабилизируется. Для подтверждения этого построим линейную регрессию с функцией $y = A + Bx$, где x — количество клиентских нитей, y — пропускная способность системы (в 1000 оп/с).

Таблица 3

Коэффициенты регрессии

Allocator	A	B	σA	σB
Glibc	61.04	–0.10	1.48	0.12
Hoard	49.30	–0.02	1.13	0.09
Jemalloc	55.58	0.11	1.23	0.10
Tcmalloc	49.61	0.13	1.34	0.11

Анализ результатов

По данным отдельных распределителей:

1. Две нити дают почти 100% прирост производительности по сравнению с одной нитью. Это объясняется тем, что система двухъядерная.
2. Добавление еще одной нити также дает некоторый прирост, т.к. в случае блокировки одной нити, процессор выполняет другую нить, а не простаивает.
3. Стабилизация пропускной способности системы с ростом числа нитей, начиная с трех. Это можно объяснить полной занятостью обоих ядер. Более точно можно судить об этом по результатам линейной регрессии.

По данным линейной регрессии:

1. Действительно, имеет место стабилизация пропускной способности (значение 0 для наклона прямой согласуется с полученными результатами).
2. Glibc показал лучшую производительность.
3. Hoard и Tcmalloc отстают и разница между ними статистически незначима.
4. Jemalloc занимает промежуточную позицию.

Такие результаты можно частично объяснить следующим образом:

В данной ситуации можно предположить, что распределитель из Glibc показывает лучшую производительность за счёт малого числа ядер, так как его конкуренты (jemalloc и tcmalloc) нацелены на работу с их большим числом.

Разработка распределителя hoard была приостановлена некоторое время назад, поэтому он не использует некоторые приемы. Этим можно объяснить его не очень хорошие показатели. Тем не менее, он был включен в наши эксперименты по причине частого его фигурирования в различных обзорах [6].

Распределитель jemalloc использует основные успешные идеи, примененные в некоторых других популярных распределителях. Это объясняет как достигнутую производительность при интенсивном обращении к памяти во время первой фазы работы системы, так и его неплохое поведение на этапе поиска по индексу.

Распределитель tcmalloc проигрывает jemalloc по той причине, что в нашем приложении нити создаются единожды и выполняются в течение всего времени работы приложения, а в такой ситуации jemalloc показывает лучшие результаты, чем tcmalloc.

Заключение

Различные этапы работы транзакционной системы характеризуются различным использованием динамической памяти. Было экспериментально показано, что те распределители памяти, которые демонстрируют лучшие

результаты при построении индексов, могут проигрывать в производительности стандартной реализации glibc при выполнении операций поиска.

Из проведенных экспериментов также можно сделать вывод, что к данной системе в качестве распределителя памяти лучше всего подходит стандартная библиотека.

Литература

1. *Sanjay Ghemawat, Paul Menage*. TCMalloc: Thread-Cashing Malloc, http://goog-perftools.sourceforge.net/doc/tcmalloc.html [дата просмотра: 17.04.2012]
2. *Mark Callaghan*. High Availability MySQL: Double sysbench throughput with TCMalloc, http://mysqlha.blogspot.com/2009/01/double-sysbench-throughput-with_18.html [дата просмотра: 17.04.2012].
3. *Wolfram Gloger*. Ptmalloc3 — a multi-thread malloc implementation, June 2006. http://svn.netlabs.org/repos/qt4/branches/4.5.1/src/3rdparty/ptmalloc/README [дата просмотра: 17.04.2012]
4. *Emery D. Berger, Kathryn S. McKinley, Robert D. Blumofe*, and *Paul R. Wilson*. Hoard: A Scalable Memory Allocator for Multithreaded Applications. ACM SIGPLAN Notices Volume 35 Issue 11, Nov. 2000, NY, USA
5. *Jason Evans*. A Scalable Concurrent malloc(3) Implementation for FreeBSD, Proceedings of BSDCan 2006, http://www.bsdcan.org/2006/papers/jemalloc.pdf [дата просмотра: 17.04.2012]
6. *Joseph Attardi, Neelakanth Nadgir*. A Comparison of Memory Allocators in Multiprocessors, 2003. http://developers.sun.com/solaris/articles/multiproc/multiproc.html [дата просмотра: 17.04.2012]
7. *Steven Fuerst*. Benchmarks of the Lockless Memory Allocator [дата просмотра: 10.05.2012] http://locklessinc.com/benchmarks_allocator.shtml

ПРИБЛИЖЕННЫЙ АЛГОРИТМ И МОДЕЛЬ СТОИМОСТИ ДЛЯ ОПЕРАЦИИ СОЕДИНЕНИЯ НА ОСНОВЕ ПОДОБИЯ

М. О. Кулешова

студент; kmo58@mail.ru

Санкт-Петербургский государственный университет

Аннотация: В данной статье предложен приближенный алгоритм для операции соединения на основе подобия и построена модель стоимости для данного алгоритма, которая впоследствии может быть применена в оптимизаторе запросов для поисковой системы. За основу алгоритма был взят алгоритм соединения вложенными циклами из реляционной базы данных.

Введение

Целью поисковой системы является предоставление информации, максимально соответствующей введенному пользовательскому запросу. При этом для получения такой информации необходимо выполнять сложные виды поиска.

Из-за природы поиска в запросах присутствуют неточности и неопределенности. В связи с этим применяются сложные виды поиска на основе подобия [3].

При сложном поиске необходимо уметь соединять полученную информацию о разных объектах из разных источников в единый результат, отвечающий на запрос поиска.

Целью данной работы является разработка приближенного алгоритма и модели стоимости для операции соединения на основе подобия для наиболее эффективного и качественного поиска информации, соответствующей введенному пользовательскому запросу.

В рамках поставленной цели сформулированы следующие основные задачи:

- создать алгоритм настраиваемым на пороги по оценкам во всех множествах;
- оценить затраты времени на выполнение алгоритма, а также определить потери данных при заданных значениях порогов;
- оценить затраты времени на выполнение алгоритма при значениях порогов, обеспечивающих полноту данных.

Приближенный алгоритм операции соединения на основе подобия

Исходными данными для алгоритма являются:

- Два множества, состоящие из трех компонентов: запрос q_i, множество объектов B_i и множество их оценок S_i (оценки принимают значения от 0 до 1).
- Предикат *pred,* отвечающий за подобие.

Предикат *pred* — это нечеткий предикат, который принимает на вход атрибуты объектов, сравнивает их и выдает некоторую оценку в интервале от 0 до 1 для данных объектов.

В результате работы приближенного алгоритма операции соединения на основе подобия будет получено множество, состоящее из трех компонентов:

- запрос q: $q = join(pred, q_1, q_2)$;
- множество объектов B: $B = B_1 \times B_2$, т. е. для всех $b \in B \quad b.id = (b_1.id, b_2.id)$ и $b.Attrs = b_1.Attrs \times b_2.Attrs$;
- множество оценок S для множества объектов B:

$$S(b) = pred(b_1, \ldots, b_n) \times S_1(b_1) \times S_2(b_2) \tag{1}$$

или

$$S(b) = \min\{pred(b_1, b_2), S_1(b_1), S_2(b_2)\}. \tag{2}$$

Таким образом, в общем виде операция соединения представима следующим образом:

$$join\big(pred,\ (q_1, B_1, S_1),\ (q_2, B_2, S_2)\big) = (q, B, S).$$

Основными алгоритмами выполнения операции соединения являются: алгоритм соединения вложенными циклами, алгоритм соединения слиянием сортированных списков, алгоритм соединения хешированием. Все данные алгоритмы изложены в [1].

Для реализации операции соединения был выбран в качестве основы алгоритм вложенных циклов, так как он не требует упорядочивания данных, которые могут быть полностью неизвестны. А также алгоритм вложенных циклов является общим алгоритмом для операции соединения и позволяет реализовать любое условие соединения.

Предлагаемый алгоритм операции соединения на основе подобия приближенный и выдает не все результаты, полученные соединением входящих множеств как алгоритм выполнения операции соединения вложенными циклами, а только часть наиболее релевантных данному запросу. Это возможно за счет следующих параметров:

1. θ_1 — порог на входящие оценки первого множества;
2. θ_2 — порог на входящие оценки второго множества;

3. θ_{pred} — порог по предикату;
4. θ_{res} — порог на оценки итогового множества.

Все параметры принимают значения от 0 до 1, как и оценки входящих множеств. Меняя данные параметры, мы можем выбирать пары объектов с необходимой нам точностью.

В статье [2] рассмотрен аналогичный алгоритм для операции соединения. Данный алгоритм работает с независимыми внешними источниками информации в отличие от предложенного алгоритма, который использует таблицы.

Модель стоимости для приближенного алгоритма операции соединения на основе подобия

При некоторых значениях параметров в результате работы алгоритма мы можем получить пустое множество, то есть исходные данные могут быть такие, что не найдется ни одной пары объектов, удовлетворяющей поставленным условиям.

Оптимизатор решает какие необходимо взять значения порогов, сколько при этом будет получено пар объектов в итоговом множества, а также сколько при этом будет затрачено времени, используя модель стоимости.

Модель стоимости включает в себя:

1. затраты на чтение данных из входящих множеств;
2. затраты на запись данных на диск;
3. затраты на чтение данных с диска;
4. затраты на вычисление предиката;
5. затраты на запись результата.

А также модель стоимости для приближенного алгоритма включает в себя вычисление потерь данных в зависимости от значений порогов.

Для каждого компонента модели стоимости были построены формулы, исходя из предположения о распределение оценок входящих множеств и значений предиката.

Рассмотрим, как вычисляются все составляющие модели стоимости на примере, когда оценки входящих множеств и значения предиката имеют равномерное распределение на интервале от 0 до 1.

1. Затраты на чтение данных из входящих множеств (*cr*)

$$cr = (n_1 + n_2) * t_{read},$$

где n_1 — размер первого входного множества, n_2 — размер второго входного множества, t_{read} — время чтения.

2. Затраты на запись данных на диск (*cdw*)

$$cdw = \left((1-\theta_1) * n_1 + (1-\theta_2) * n_2\right) * t_{disk\_write},$$

где n_1 — размер первого входного множества; n_2 — размер второго входного множества; t_{disk_write} — время записи на диск.

3. Затраты на чтение данных с диска (*cdr*)

$$cdr = \left((1-\theta_1)*n_1 + (1-\theta_2)*n_2\right)*t_{disk\_read},$$

где n_1 — размер первого входного множества; n_2 — размер второго входного множества; t_{disk_read} — время чтения с диска.

4. Затраты на вычисление предиката (*ccp*)

$$ccp = acp * t_{pred},$$

где t_{pred} — среднее время вычисления предиката из некоторой статистики; *acp* — это количество вычислений предиката:

$$acp = \left((1-\theta_1)*n_1*(1-\theta_2)*n_2\right),$$

где n_1 — размер первого входного множества; n_2 — размер второго входного множества.

5. Затраты на запись результата (*cwr*)

$$cwr = AO * t_{disk\_write},$$

где t_{disk_write} — время записи на диск; AO — количества пар объектов в итоговом множестве.

Пара объектов будет записана в итоговом множестве, если:

a) оценка объекта из первого входящего множества больше или равна θ_1;
b) оценка объекта из второго входящего множества больше или равна θ_2;
c) значение предиката для данных двух объектов больше или равно θ_{pred};
d) итоговая оценка для данных двух объектов больше или равна θ_{res}.

Условия a), b) и c) образуют в трехмерном пространстве параллелепипед P^*. Условие d) и формула (1) образуют в трехмерном пространстве гиперболоид H. Условие d) и формула (2) образуют в трехмерном пространстве параллелепипед P.

Для того, чтобы найти количество пар объектов в итоговом множестве, необходимо найти объем общей части, полученной при пересечении параллелепипеда P^* и гиперболоида H или параллелепипеда P^* и параллелепипеда P.

Для того, чтобы вычислить потери данных, необходимо найти разность между количеством данных, оценки которых удовлетворяют условию d) (объем гиперболоида H или объем параллелепипеда P), и количеством данных в итоговом множестве (AO).

Эксперименты

Для выявления точности моделей стоимости были проведены эксперименты с коллекцией строк для операции соединения на основе подобия. Все оценки входящих множеств, а также значения оценок предиката генерировались в соответствие с выбранным распределением (равномерным, нормальным, экспоненциальным). Измерялось время выполнения приближенного алгоритма операции соединения в секундах ($time_{pract}$), а также количество пар объектов в итоговом множестве ($full_{pract}$). Ниже представлена таблица результатов для равномерного распределения входящих оценок и значений предиката на интервале от 0 до 1, когда итоговая оценка вычислялась согласно формуле (1).

Таблица 1

Результаты экспериментов для равномерного распределения на интервале от 0 до 1

№	θ_1	θ_2	θ_{pred}	θ_{res}	$time_{pract}$, с	$time_{theory}$, с	$full_{pract}$	ful_{ltheor}у
1	0.7	0.3	0.4	0.6	112.42	112.35	165	148
2	0.3	0.7	0.4	0.6	84.04	74.96	137	126
3	0.3	0.8	0.7	0.6	83.11	74.81	60	51
4	0.7	0.3	0.8	0.6	111.77	112.30	133	122
5	0.5	0.4	0.3	0.6	152.96	160.38	168	152

Результаты экспериментов показали, что процентное соотношение полученной стоимости к реальной стоимости равно 89%. Из этого можно заключить, что оценка предложенной модели стоимости для операции соединения на основе подобия близка к реальности.

Заключение

В работе был реализован приближенный алгоритм, а также построена модель стоимости для данного алгоритма.

Литература

1. *К. Дж. Дейт.* Введение в системы баз данных: научно-популярная литература. 7-е изд. М.: Вильямс, 2001.
2. *Daniele Braga, Stefano Ceri, Michael Grossniklaus*. Join Methods and Query Optimization. SeCO Workshop 2009.
3. *P. Zezula, G. Amato, V. Dohnal, M. Batko*. Similarity Search — The Metric Space Approach. Springer, 2006.

РАСШИРЕНИЕ СТРУКТУР, ВЫЧИСЛЯЮЩИХ АДДИТИВНУЮ ФУНКЦИЮ НА МНОЖЕСТВАХ ТОЧЕК

А. А. Сидоров

asidorov566@gmail.com

Научный руководитель:

Б. А. Новиков

д.ф.-м.н., проф.; borisnov@acm.org

Аннотация: В данной работе предлагается метод, который позволяет улучшать произвольные структуры данных, обладающие операциями *build* от множества объектов и *get* от этого множества и некоторого параметра, так, что бы эти структуры могли обзавестись новой операцией *add*. В качестве подобных структур могут выступать, например, 2d-дерево отрезков, диаграмма Вороного или же kd дерево. Если операция *get* обратима, то новая структура будет поддерживать операцию *DEL*. Так же будет поддерживаться операция *MERGE*.

1. Введение

Существует множество структур, которые обладают операцией вычисления значения некоторой функции на множестве заданных объектов. Это может быть, например, число точек или же минимум по значениям точек в прямоугольнике, ближайшая точка в множестве. Зачастую, подобные структуры не поддерживают эффективной операции *add*, которая добавляла бы новые объекты в структуру. Для добавления новых объектов приходится полностью ее перестраивать, либо же добавление новых объектов напрямую значительно увеличивает время работы. Например, такими структурами являются 2d дерево отрезков и диаграмма Вороного. Эти структуры обладают двумя типами операций. Первый тип, *build*, создаёт структуру из множества заданных объектов. Второй тип, *get*, отвечает на запросы. В случае 2d дерева отрезков это может быть запрос суммы на прямоугольнике. В случае диаграммы Вороного это может быть поиск ближайшего соседа на плоскости. Но в этих структурах не поддерживается эффективная операция *add* — добавление новых элементов в исходное множество. Для добавления новой точки в диаграмму Вороного или дерево отрезков пришлось бы их перестроить. В данной работе предлагается метод, позволяющий создавать новые структуры на основе уже имеющихся, которые будут поддерживать эффективное добавление новых точек в исходное множество. При этом, внутренняя реализация исходных структур нам может быть не известна, для нас будет важен только интерфейс операций *build* и *get*. Размерность пространства для нас так же не будет иметь значения.

2. Постановка задачи

Пусть у нас имеется структура данных с двумя операциями *build* и *get*. *build* выполняет построение структуры на множестве заданных элементов. *get* вычисляет значение функции $f(X, p)$, где X — текущее множество элементов, а p — дополнительный параметр. $p \in U$ — множество допустимых параметров. При этом должен существовать такой оператор $\oplus$, что будет верно следующее:

$$\forall A, B, p : A \cap B = \varnothing,\ p \in U,$$
$$f(A, p) \oplus_p f(B, p) = f(A \cup B, p).$$

То есть существует способ по значениям функции $f(p)$ на двух непересекающихся множествах узнать ее значение на их объединении.

Время работы операции *build* — $O(build(N))$, где N — число элементов. Время работы операции *get* — $O(get(N))$. Время работы оператора $\oplus$ — $O(time_{\oplus})$.

Необходимо построить структуру, поддерживающую следующие операции: *BUILD*, *GET* и *ADD*. Операция $BUILD(A)$ строит структуру по множеству точек A. Операция $GET(A, p)$ вычисляет значение функции $f(A, p)$ $(p \in U)$. Операция *ADD* добавляет новую точку в множество точек A.

3. Построение новой структуры [1]

3.1. *Общий вид новой структуры*

Пусть у нас есть произвольная структура данных, которая обладает операцией *build* от множества точек и операцией *get* на множестве допустимых параметров U. *build* представляет из себя некоторый предварительный подсчёт, а *get* — ответ на некоторый запрос. Например, в качестве такой структуры можно рассмотреть диаграмму Вороного, отвечающую на запросы «найти точку из исходного множества, ближайшую к данной». Существует реализация, в которой *build* работает за $O(N \cdot log(N))$, а *get* за $O(log(N))$ [2, 3]. Построим новую структуру данных на основе этой. В каждый момент времени будем поддерживать следующий инвариант: все точки разбиты на несколько непересекающихся подмножеств попарно различных размеров, причём размер каждого — некоторая степень двойки. Таким образом, если A — множество точек в данный момент, а N — их количество, то выполняется следующее:

$$A = A_1 \cup A_2 \cup \ldots \cup A_k,$$
$$|A_i| = 2^{a_i},$$
$$a_i \neq a_j \quad (i \neq j),$$
$$A_i \cap A_j = \varnothing \quad (i \neq j).$$

Числа a_i соответствует номерам единичных бит двоичной записи числа N (младший бит — нулевой).

Для каждого такого подмножества будет создана своя структура данных исходного типа. Очевидно, что при подобном разбиении число различных структур не превосходит $log(N)$, т. к. размеры подмножеств соответствуют представлению N в двоичной системе счисления. Условимся, что заглавными буквами будут обозначаться запросы к новой структуре данных (*ADD*, *GET*, *MERGE*), а строчными — к исходной (*get*, *build*).

3.2. *Операция BUILD*

Пусть нам поступил запрос на выполнение операции *BUILD* на множестве элементов A из N элементов. Пусть $\{a_i\}_{i=1}^{k}$ — номера единичных бит в двоичной записи числа N. Тогда $N = 2^{a_1} + 2^{a_2} + ... + 2^{a_k}$. Разобьём произвольным образом наше множество A на непересекающиеся подмножества $A_1, A_2, ..., A_k$, такие что $|A_i| = 2^{a_i}$. Для каждого из них построим структуру исходного типа. В результате мы достигли состояния, удовлетворяющего условию нашего инварианта.

3.3. *Операция ADD*

Пусть нам поступил запрос на операцию *ADD*. Создадим новую структуру данных исходного типа размера 1, состоящую только из одного добавляемого элемента. Затем, пока существуют две структуры одинакового размера, будем объединять эти две структуры в одну с размером в два раза больше, полностью выполняя *build* для всего объединения множеств их элементов. После выполнения этих действий наш инвариант сохранился — размеры всех структур являются различными степенями двойки.

3.4. *Запрос GET*

Пусть нам поступил запрос $GET(A, p)$ $(p \in U)$ A — текущее множество элементов. Тогда вычислим значение *get* для всех A_i, а затем применим наш оператор $\oplus$ ко всем результатам по разным структурам. Более формально, ответом на наш запрос будет значение следующего выражения:

$$get(A_1, p) \oplus_p get(A_2, p) \oplus_p ... \oplus_p get(A_k, p).$$

3.6. *Оценка времени работы операции ADD*

Не нарушая общности, будем считать, что изначально наша новая структура была пуста, а затем в неё было добавлено N точек по одной. Тогда не сложно вычислить, что время добавления всех N точек оценивается следующим образом:

$$time_{ADD_{all}}(N) = O\left(\sum_{k=0}^{mlog} build(2^k) \cdot N / 2^k\right). \quad (1)$$

Тогда среднее время добавления одной точки оценивается так:

$$time_{ADD}(N) = O\left(\sum_{k=0}^{mlog} \frac{build(2^k)}{2^k}\right). \quad (2)$$

Необходимо помнить, что это амортизированная оценка. В дальнейшем мы покажем, как с помощью откладывания выполнения операций избавиться от амортизации в оценке.

3.6. *Оценка времени работы операции GET*

По данному ранее определению, время работы операции *get* на множестве из N точек мы обозначаем как $get(N)$. Нетрудно вывести, что время работы одной операции *GET* оценивается следующим образом:

$$time_{GET}(N) = O\left(\sum_{k=0}^{mlog} get(2^k) + mlog \cdot time_{\oplus_p}\right). \quad (3)$$

Отсюда видно, что время работы одной операции *get* возросло не более чем в $log(N)$ раз.

3.7. *Другие операции*

Данная структура так же может поддерживать операции *MERGE* и *DEL*. *MERGE* будет объединять подмножества каждой из структур, а затем выполнять объединении одинаковых по размеру подмножеств, начиная с самых маленьких, до тех пор, пока не выполнится условие инварианта. Операция удаления *DEL* возможная в случае, если запрос *GET* вычисляет обратимую функцию. Тогда можно добавлять элементы с обратным значением. Детальный разбор этих операций выходит за рамки этой статьи.

4. Избавление от амортизации в оценке

4.1. *Мотивация*

Вернёмся к оценке времени выполнения операции *ADD*. Среднее время добавления одного элемента по формуле (2) оценивалось как

$$time_{ADD}(N) = O\left(\sum\nolimits_{k=0}^{mlog} \frac{build(2^k)}{2^k}\right).$$

Посмотрим, каким может быть время добавления одного элемента в худшем случае. Пусть у нас перед добавлением нового элемента размер множества был равен $2^k - 1$, тогда при добавлении нового элемента по нашему алгоритму потребуется выполнить *build* от 2^k элементов. Очевидно, что это и есть худший случай — число элементов, от которых запускается *build*, максимально. В некоторых случаях хочется иметь гарантию на быстрое выполнение каждой операции и усреднённая оценка не подходит.

4.2. *Описание метода*

Рассмотрим метод, который позволяет достичь оценки (2) в худшем случае. Изменим наш инвариант следующим образом. Теперь количество подмножеств размера 2^k будет в каждый момент времени не превосходить двух. Операция *ADD* изменяется следующим образом:

1. Для каждого уровня, на котором есть два подмножества размера 2^k, выполнить $\frac{1}{2^k}$ часть от операции объединения этих двух подмножеств.
2. Если для каких-то двух подмножеств одного размера операция их объединения завершена, то заменяем их на результат их объединения. Результат записывается на уровень с номером на 1 больше.
3. Добавляем подмножество размера 1, состоящее из одного нового элемента.

Для доказательства корректности работы преложенного метода осталось проверить следующие утверждения:

- При добавлении новых элементов не нарушается новый инвариант.
- Асимптотика времени работы операций со структурой не изменилась.

И то и другое не сложно проверяется, детальная проверка обоих утверждений выходит за рамки данной статьи.

5. Пример применения: дерево отрезков 2d

5.1. *Постановка задачи*

Пусть у нас имеются точки на плоскости, каждой точке присвоены некоторые значения. Пусть A — множество точек в данный момент. Нам необходимо выполнять следующие запросы:

- $GET(R)$ — найти среди значений точек множества A, лежащих в прямоугольнике R, минимальное. Стороны R параллельным осям координат.
- $ADD(p, x)$ — добавить новую точку p в A и присвоить ей значение x.

5.2. *Статический вариант задачи*

Рассмотрим случай, когда множество точек заранее известно и нам поступают только запросы *GET*. Тогда построим на этом множестве точек двумерное дерево отрезков с операцией минимума [4]. В этой структуре сначала строится дерево отрезков по первой координате, а потом в каждую вершину этого дерева добавляется ссылка на дерево отрезков по второй координате для точек данной вершины. Построение такого дерева отрезков можно реализовать асимптотикой $O(N \cdot log^2(N))$, а время запроса будет равно $O(log^2(N))$. При этом используется $O(N \cdot log(N))$ памяти. Таким образом, мы получаем решение для статического варианта нашей задачи.

5.3. *Динамический вариант задачи*

Применим метод разбиения множества на $log(N)$ подмножеств. Для каждого подмножества будет строиться описанная выше структура — двумерное дерево отрезков. Для объединения результатов для разных деревьев нам будет достаточно найти среди них точку, значение в которой минимально, это делается просто перебором всех полученных результатов.

5.4. *Асимптотика работы*

Операция *build*, как было сказано выше, имеет асимптотику $O(N \cdot log^2(N))$, а операция *get* — $O(log^2(N))$. Тогда по формулам (2) и (3) имеем:

$$time_{GET}(N) = O\left(\sum_{k=0}^{mlog} get(2^k)\right) = O\left(\sum_{k=0}^{mlog} log^2(2^k)\right) = O(log^3 N),$$

$$time_{ADD}(N) = O\left(\sum_{k=0}^{mlog} \frac{build(2^k)}{2^k}\right) = O\left(\sum_{k=0}^{mlog} \frac{2^k \cdot k^2}{2^k}\right) = O(log^3 N).$$

5.5. *Выводы*

Мы получили способ добавления новых элементов в двумерное дерево отрезков, при использовании которого асимптотики времён выполнения операций *GET* и *ADD* будут такие, как описано в разделе 5.4. Добавление точек в такую структуру можно было бы реализовать и напрямую. Для этого достаточно найти $O(log(N))$ вершин первого дерева, которым принадлежит новая точка, и для каждой из них найти $O(log(N))$ вершин ее второго дерева и пересчитать в них функцию. Но такой подход будет работать только на случайных точках. Иначе последовательное добавление упорядоченных точек испортит балансировку дерева и его высота станет $O(N)$. Это приведёт к тому, что операции *ADD* и *GET* так же будут выполняться за линейное от числа точек время.

6. Другие работы

Подобный метод разбиения множества элементов на части размера 2^k был описан в статье [1]. Главное отличие состоит в том, что в этой статье не предлагается способа избавления от амортизации в оценке, что может быть очень важно в некоторых случаях. Так же в этой статье приводится анализ того, почему разбиение на части именно такого размера является наилучшим.

7. Заключение

Описанный метод разбиения множества элементов на части размера 2^k позволяет добавить динамическое поведение в очень широкий класс структур, которые до этого не являлись динамическими. При этом данный метод прост в реализации и обладает низкой внутренней константой. Важным является то, что полученная оценка на время добавления нового элемента в структуру является амортизированной и для того, что бы избавиться от неё и гарантировать быстрое добавление, придётся воспользоваться методом частичного выполнения операций объединения. Так же особенность такого метода заключается в том, что время добавления новых элементов в структуру будет практически не зависеть от текущего размера структуры, если большинство ее элементов принадлежат подмножествам старших уровней. Как было отмечено ранее, асимптотика времени работы операции *get* в результате применения этого метода возрастает не более чем в $log(N)$ раз.

Литература

1. *J. L. Bentley* and *J. B. Saxe*. Decomposable searching problems, 1980.
2. *Steven Fortune*. A sweepline algorithm for Voronoi diagrams. Proceedings of the second annual symposium on Computational geometry. Yorktown Heights, New York, United States. Pp. 313–322. 1986.
3. *Snoeyink, Jack* (2004). «Chapter 34: «Point Location». In Goodman, Jacob E.; O»Rourke, Joseph. Handbook of Discrete and Computational Geometry (2nd ed.). Chapman & Hall/CRC.
4. *D. E. Willard.* New Data Structures for Orthogonal Range Queries, SIAM Journal on Computing, 14(1):232–253. 1985.
5. *Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн.* Алгоритмы. Построение и анализ. Introduction to Algorithms. 2-е изд. «Вильямс», 2005.

Кибернетика и робототехника

Фрадков
Александр Львович

д.т.н.
профессор кафедры теоретической кибернетики СПбГУ
заведующий лабораторией
«Управление сложными системами» ИПМаш РАН

Лучин
Роман Михайлович

ст.преподаватель кафедры теоретической кибернетики СПбГУ
научный сотрудник лаборатории теоретической кибернетики

DCU – SDK ДЛЯ БЫСТРОГО РАЗВЕРТЫВАНИЯ СИСТЕМ ТЕЛЕМЕХАНИКИ

А. С. Крупенькин

студент кафедры систем управления и информатики;
alexandr.krupenkin@gmail.com

А. А. Хасанов

студент кафедры систем управления и информатики;
alisher.khassanov@gmail.com

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Аннотация: Предлагается описание работы по построению SDK для разработки систем телемеханики с интегрированным веб-интерфейсом. Даётся обзор проблем, возникающих при построении АСУ с использованием глобальных и локальных компьютерных сетей в качестве основных каналов передачи информации, и подходы к решению некоторых из них.

Введение

На сегодняшний день высокая развитость сетевых компьютерных технологий, Глобальной сети Интернет и беспроводных коммуникационных технологий делает компьютерные сети привлекательным инструментом для построения систем телемеханики.

Проникновение сети Интернет за последние годы достигло 25,6 % в мире, 40–90 % в Европе, 32 % в округах РФ, 58 % в Москве и Санкт-Петербурге[1], что делает её интересной для построения систем удалённого управления продукцией коммерческих организаций в рекламных целях.

Попытки использования вычислительных машин, функционирующих под управлением операционных систем общего назначения, и их сетей в системах управления порождают большой спектр проблем, связанных с наличием запаздывания и потери информации при передаче.

Имеется необходимость создания единой системы, позволяющей быстро и просто реализовывать системы телемеханики разработчиками различного уровня компетенции, будь то студент высшего учебного заведения, разработчик коммерческой компании или другое заинтересованное лицо, не задумываясь о решении стандартных проблем возникающих при построении таких систем.

Попытка решения этой задачи реализуется в проекте DCU.

DCU (аббревиатура от рекурсивного названия: DCU Control Unit) разрабатывается как набор ПО, позволяющий быстро развёртывать системы телемеханики с предоставлением веб-интерфейса к управляемому робототехническому комплексу или другому устройству.

[1] По данным International Communication Union и компании «Яндекс»: «Развитие интернета в регионах России» 2011 год.

В качестве базовых принципов при разработке SDK DCU были взяты следующие:

- Низкие требования к компетенции разработчика, работающего с SDK:

  — это продиктовано ориентацией SDK на студентов, обучающихся в области IT и управления, где для работы над проектом нет времени для освоения сложных и новых для разработчика технологий.

- Глубокая интеграция стандартных наукоемких решений. Предоставление удобной среды для решения текущих задач:

  — этот принцип позволит использовать DCU в коммерческих проектах, для работы над которыми не потребуется привлекать специалистов в области управления, а достаточно лишь разработчика, компетентного в используемых технологиях.

- Модульность и простота архитектуры с легко отчуждаемыми компонентами, использование популярных технологий:

  — модульность архитектуры позволяет системе быть достаточно гибкой для использования в решении различных задач, для управления самыми разнообразными объектами и системами; простота архитектуры и построение системы на базе популярных технологий, в свою очередь, даст возможность работающему с SDK разработчику быстро находить решение своей задачи, не отвлекаясь на изучение специализированных технологий.

Архитектура DCU, решаемые задачи

Архитектура DCU построена следующим образом:

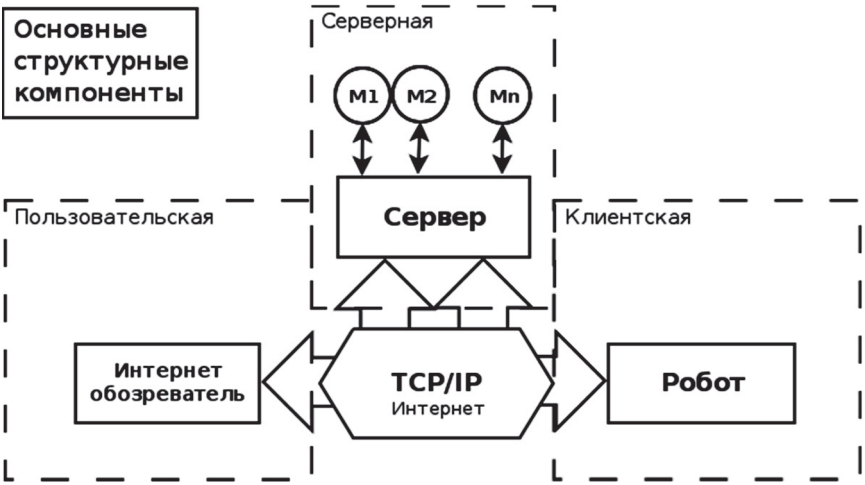


Рис. 1. Архитектура DCU

В рамках требования к модульности и простоте архитектуры было принято решение придерживаться концепции MVC (Model-view-controller, «Модель-представление-поведение»). Использование такой архитектуры позволяет, имея универсальное ядро системы, дополнять его модулями (M_1, ..., M_n на рис. 1) с описанием работы клиентского устройства (Робот на рис. 1) и его пользовательским интерфейсом. Это так же даёт необходимую гибкость и универсальность системе.

Базовые принципы построения DCU выявили ряд первичных задач:

1. Формирование SDK на основе стандартных решений и популярных технологий:
   1.1. настройка и интеграция фронт-энда, обрабатывающего статические запросы к веб-серверу;
   1.2. настройка и интеграция бэк-энда, обрабатывающего динамические запросы к веб-серверу;
   1.3. формирование фреймворка для веб-приложений на популярном языке программирования;
   1.4. стандартизация внутренних протоколов взаимодействия компонент и интерфейсов программного уровня;
   1.5. интеграция наукоемких решений стандартных задач в виде библиотечных функций;
   1.6. подготовка пакета документации.
2. Решение ряда научно-исследовательских задач, возникающих при использовании доступных и популярных технологий.

Текущие проблемы, работы и результаты

В качестве основы DCU были взяты пакеты свободного программного обеспечения и для решения поставленных задач использованы следующие:

1. Фронт-энд nginx;
2. Бэк-энд uWSGI;
3. Фреймворк django, позволяющий:
   3.1. описывать модуль DCU на популярном интерпретируемом языке программирования Python с использованием доступных веб-технологий для реализации пользовательского интерфейса;
   3.2. создавать библиотеки стандартных наукоёмких решений в виде модулей django;
   3.3. использовать единый протокол взаимодействия компонент и интерфейсов программного уровня;

не требуя при этом значительной модификации.

Одна из текущих задач — это предоставление возможности функционирования системы телемеханики в условиях задержек и потерь данных.

Из работ, проведённых в этой области [1–2], были рассмотрены следующие подходы:

1. Формирование траектории объекта управления с использованием его 3D модели на стороне оператора и последующая отработка траектории объектом;
2. Работа в синхронном режиме с фиксированной задержкой, максимальной для данной сети;
3. Минимизация запаздывания и восстановление потерянных данных;

Подходы (1) и (2) не являются актуальными для коммерческого сектора, потому был подробнее рассмотрен последний. В работе [2] предлагается использовать авторегрессионную модель для управления в условиях запаздываний в контуре системы. Рассмотрим подробнее этот подход.

Авторегрессионная (АР) модель для предикции показаний измерительных устройств

Пусть функциональная схема системы выглядит следующим образом:

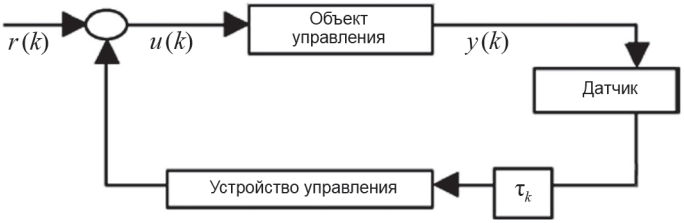


Рис. 2. функциональная схема для системы с АР-моделью

Здесь τ_k — запаздывание на k-том шаге.

Тогда восстановление данных, запаздывание которых превысило допустимое значение можно произвести с использованием следующей модели:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k), \\ y(k) &= Cx(k), \\ u(k) &= r(k) - Ky(k - h(\tau_k)), \end{aligned} \tag{1}$$

где K — матрица коэффициентов усиления устройства управления.

Здесь конечное неотрицательное целое $h(\tau_k)$ отражает количество считающихся потерянными на данный момент измерений и может быть получено следующим образом:

$$h(\tau_k) = \left\lfloor \frac{T_s + \tau_k - \tau_0}{T_s} \right\rfloor, \tag{2}$$

где T_s — период квантования дискретной модели, τ_0 — время порогового запаздывания, отражающее то время, после которого измерения считаются потерянными, $\llcorner\lrcorner$ — функция взятия антье аргумента, функция пол.

Для данного $h(\tau k)$ вектор состояния из (1) можно получить следующим образом:

$$x(k+1) = A^{h(\tau_k)+1}x(k-h(\tau_k)) + \sum_{j=0}^{h(\tau_k)} A^j Bu(k-j), \tag{3}$$

Далее, из (1) и (3) имеем:

$$x(k+1) = A^{h(\tau_k)+1}x(k-h(\tau_k)) + \\ + \sum_{j=0}^{h(\tau_k)} A^j B[r(k-j) - k\hat{y}(k-h(\tau_k)-j)]. \tag{4}$$

Здесь $\hat{y}$ — оценка показаний измерительного устройства.

Эта оценка может быть использована при $\tau_k > \tau_0$.

С помощью такого алгоритма оценки потерянных измерений можно иметь достаточно точные и актуальные данные о выходе объекта управления при наличии запаздываний, значительно превышающих период квантования.

Благодаря своей простоте, алгоритм может быть реализован в виде библиотечной функции DCU, над чем сейчас и ведется работа.

Эксперименты, проведённые авторами [2] показали, что алгоритм позволяет иметь актуальные данные при использовании AP-модели пятого порядка для задержек до 1,42 секунд, что вполне приемлемо как для работы в глобальной сети, так и локальной Ethernet сети, типичные запаздывания пакетов в которой показаны на рисунке 3.

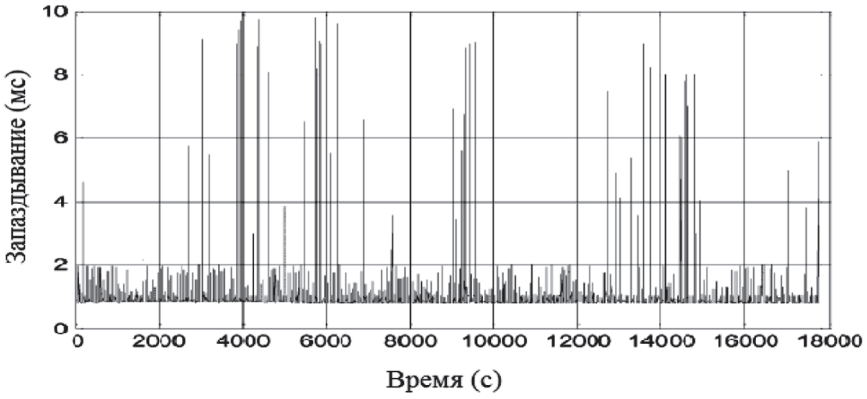


Рис. 3. Запаздывание данных при использовании пакетной технологии

Текущие результаты

1. Клиент-серверная часть:
   1.1. стартовая страница и набор стандартных визуальных средств: окна, фоны, кнопки и прочее;
   1.2. система авторизации и разделения привилегий пользователей;
   1.3. база данных PostgreSQL с кэшированием.
2. Модуль тестового робота Lynxmotion Johnny 5:
   2.1. Модель в виде python-класса, представление в виде html страницы с использованием CSS3 и JavaScript;
   2.2. Интерфейсы управления: переход между состояниями ДКА по кнопке, веб-джойстику, СТЗ на основе OpenCV.

Пробы стабильных версий DCU были выполнены на открытии фестиваля «HackSpace Weekend» в октябре 2011 года, на выставках с участием кафедры СУиИ СПбНИУ ИТМО и днях открытых дверей университета в весенний период 2012.

Актуальной на данный момент является работа над тестовой библиотекой предикции потерянных данных и разработка пакета документации для передачи DCU в виде SDK в работу над каким-либо другим проектом, где проявят себя недостатки текущей версии и будут выявлены новые актуальные задачи.

Заключение

В документе была представлена основная информация о SDK DCU, решаемых и перспективных задачах проекта, основных используемых технологиях, проблемах, возникающих при построении систем управления с использованием Глобальной сети в качестве канала коммуникаций и подходы к решению некоторых из них.

Литература

1. ИПМ им. М. В. Келдыша РАН. И. Р. Белоусов. Алгоритмы управления роботом-манипулятором через интернет // Математическое моделирование». Т. 14. № 8. 2002. С. 10–15.
2. *Won-jon Kim, Kun Ji, Abhinav Srivasta*. Network-based control with real-time prediction of delayed/lost sensor data // IEEE Transactions on control system technology. Vol. 14. No. 1. January 2006.

ДИФФЕРЕНЦИАЛЬНЫЙ АЛГОРИТМ ОЦЕНИВАНИЯ ЧАСТОТ МУЛЬТИГАРМОНИЧЕСКОГО СИГНАЛА

А. А. Лосенков

студент кафедры систем управления и информатики;
alosenkov@yandex.ru

Л. В. Никифорова

студент кафедры систем управления и информатики;
liliya.nikiforova@gmail.com

Научный руководитель:

А. А. Пыркин

к.т.н., доцент кафедры систем управления и информатики;
a.pyrkin@gmail.com

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Аннотация: Рассматривается новый метод идентификации параметров гармонического и мультигармонического сигналов. В основе данного метода лежит алгоритм оценки частоты, упрощающий схему идентификации и снижающий количество измерений в процессе оценки параметрически не определенного мультигармонического сигнала.

Введение

Задача идентификации параметров неизвестных гармонических сигналов является важной задачей теории управления. Во многих инженерных задачах, например, [1], встречаются возмущающие воздействия, имеющие подобную структуру. Определив параметры этих гармоник, можно построить алгоритм их компенсации. Актуальность подобных исследований подтверждается большим числом работ, посвященных идентификации неизвестной частоты синусоидальной функции [1–8]. Во многих работах (например, [1–6]) исследуются сигналы, состоящие из одной гармоники. Так же активно исследуются мультигармонические сигналы (например, [7, 8]).

В данной исследовательской работе представляются методы построения адаптивного наблюдателя и идентификации параметров гармонического и мультигармонического сигнала, включая частоту, амплитуду и фазу каждой гармоники, разработанные в [2]. Также предлагается алгоритм, позволяющий проводить идентификацию частот большого числа гармоник без заметного усложнения алгоритма.

Постановка задачи

Рассмотрим сигнал вида:

$$y(t) = \sum_{i=1}^{k} \mu_i \sin(\omega_i t + \phi_i), \tag{1}$$

представляющий собой сумму k гармоник с частотами ω_i, амплитудами μ_i и начальными фазами ϕ_i, которые являются неизвестными; $i = \overline{1, k}$ — номер гармоники.

Ставится задача определения частот ω_i сигнала (1).

Синтез алгоритма

Вводится в рассмотрение фильтр вида

$$\xi(s) = \frac{ks}{T^2 s^2 + 2\nu T s + 1} y(s), \tag{2}$$

где k — коэффициент усиления, T — постоянная времени, $\nu \in (0; 1)$ — коэффициент затухания.

Фильтр (2) имеет определённую полосу пропускания, которая может быть охарактеризована логарифмической амплитудной характеристикой. Пример такой характеристики представлен на рисунке 1. Как видно из рисунка, логарифмическая амплитудная характеристика фильтра (2) имеет явный максимум, соответствующий точке на оси частот, определяемой выражением $\omega_p = \frac{1}{T}$.

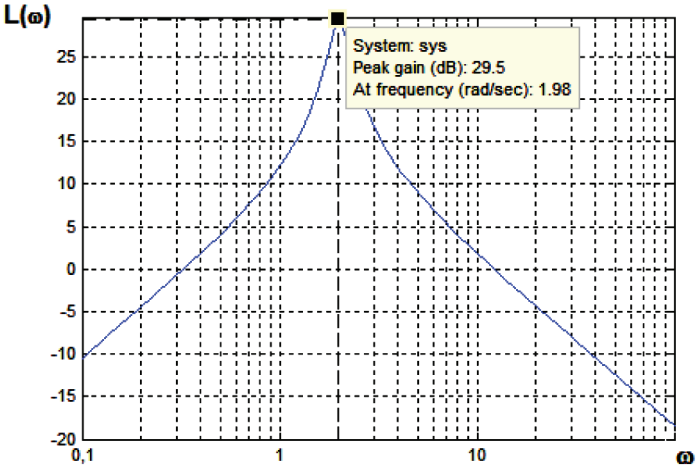


Рис. 1. Логарифмическая амплитудная характеристика фильтра при $T = 0.5$ с, $k = 3$, $\nu = 0.1$

Таким образом, амплитуды гармоник с частотами ω_i, близкими к ω_p, будут усиливаться, в то время как амплитуды гармоник с частотами ω_i, лежащими далеко от резонансной частоты, будут подавляться.

Это означает, что, задавая значение постоянной времени такой, что выделяется соответствующая этой частоте гармоника сигнала (1), её амплитуда вследствие резонанса заметно вырастает, в то время, как амплитуды других гармоник, подавленные фильтром, оказывают незначительное влияние на сигнал и могут трактоваться как шум.

На рисунке 2 четко видно, что сигнал выхода фильтра поочередно достигает максимумов своей амплитуды при значениях постоянной времени, обратных значениям частот поданного на фильтр мультигармонического сигнала. Данный график доказывает действие исследованного алгоритма идентификации частот мультигармонического сигнала. Полученные результаты справедливы для полисигналов с количеством гармоник больше двух.

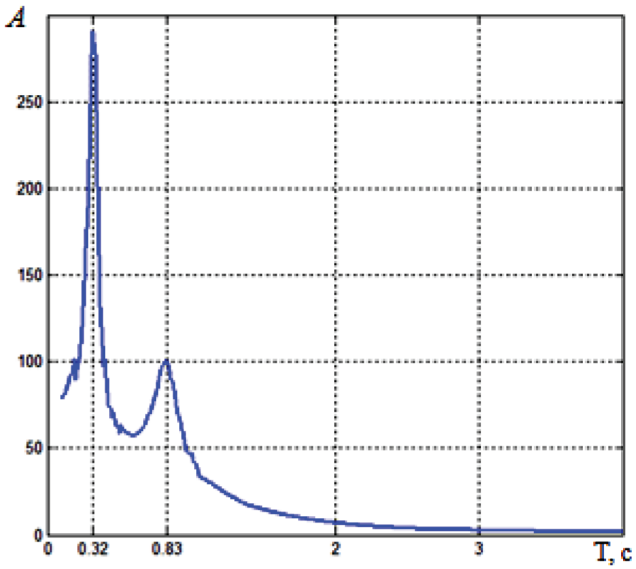


Рис. 2. Зависимость максимальной амплитуды выходного сигнала фильтра (2) от постоянной времени T при двух гармониках входного сигнала

$$y(t) = 5\sin(1.2t + 2) + 6\sin(3.1t + 4)$$

Заключение

В данной работе предложен дифференциальный алгоритм оценивания частот мультигармонического сигнала, представляющего собой сумму гармоник с неизвестными параметрами. Установлено, что алгоритм позволяет выделять из выходного спектра гармоники и определять их частоту.

Литература

1. Никифоров В. О., Гутнер *И. Е., Сергачев И. В.* Система активной виброзащиты: разработка, результаты испытаний и перспективы развития // Мехатроника, автоматизация и управление. 2004. № 2.
2. Пыркин *А. А.* Методы адаптивного и робастного управления в условиях запаздывания и возмущающих воздействий : дис. канд. техн. наук : 05:13:01 : защищена 19.10.10. Санкт-Петербург, 2010. 151 с.: ил. РГБ ОД, 61 10–5/3222.
3. Бобцов *А. А.,* Кремлев *А. С.* Адаптивная идентификация частоты смещенного синусоидального сигнала // Известия вузов. Приборостроение. 2005. № 4. С. 22–26.
4. *Marino R.* and *Tomei R.* Global Estimation of Unknown Frequencies // IEEE Transactions on Automatic Control. 2002. V. 47. P. 1324–1328.
5. *Xia X.* Global Frequency Estimation Using Adaptive Identifiers // IEEE Transactions on Automatic Control. 2002. V. 47. P. 1188–1193.
6. Hou M. Amplitude and frequency estimator of a sinusoid // IEEE Transactions on Automatic Control. 2005. V. 50. P. 855–858.
7. *Bobtsov A., Lyamin A., Romasheva D.* Algorithm of parameter»s identification of polyharmonic function // 15th IFAC World Congress on Automatic Control. Barcelona, Spain. 2002.
8. Бобцов *А. А.,* Колюбин *С. А.,* Пыркин *А. А.* Компенсация неизвестного мультигармонического возмущения для нелинейного объекта с запаздыванием по управлению // АиТ. 2010. № 11. С. 136–148.

ТРАЕКТОРНОЕ ДВИЖЕНИЕ ГРУППЫ РОБОТОВ

С. А. Вражевский

студент кафедры систем управления и информатики;
Vrazhevskij.S@gmail.com

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Аннотация: В данной работе предлагается к рассмотрению реализованная на мобильных роботах комплексная система поиска и оптимизации траектории в неструктурированной среде.

Введение

Мобильные роботы представляют собой класс устройств, наиболее подходящих для решения задач, требующих присутствия исполнительного устройства в труднодоступных или недоступных вовсе для человека местах. Причиной возникновения таких проблем могут быть техногенные аварии и катастрофы, стихийные бедствия, необходимость исследования поверхностей небесных тел, разведка аварийных зданий и прочее.

Таким образом, большую практическую значимость приобретают задачи траекторного движения роботов и поиска траектории в неструктурированной среде. Комплексное же решение таких задач позволяет повысить быстродействие системы и дает возможность выявить наиболее оптимальную для складывающейся ситуации траекторию.

Цель работы

Целью данной работы является моделирование и последующая апробация на реальных роботах системы управления движением группы агентов по неизвестной траектории.

Предлагается метод траекторного управления при движении в неструктурированной среде, основанный на распределении функций между агентами, последовательная реализация которых обеспечит выполнение поставленной задачи с требуемой точностью. Планирование траекторий для обхода препятствий осуществляется на основе данных, полученных непосредственно самими роботами в процессе работы.

Базовые положения

Специфика рабочей среды

Важным ограничением для функционирования разрабатываемой системы является относительная гладкость рабочей поверхности, влияющая

как на ходовые характеристики и проходимость используемых роботов, так и на точность обработки данных при определении препятствий.

Также в процессе синтеза и отладки системы требуется постепенно повышать плотность препятствий среды и расширять их класс, тем самым позволив сначала опробовать алгоритмы оптимизации траектории для простых случаев, а затем усложнять и адаптировать их вместе с алгоритмами ориентации в пространстве, добиваясь максимальной универсальности системы.

Аппаратная часть

В практической части используются мобильные роботы Boe-bot компании Parallax inc., обладающие удобными для апробации системы массо-габаритными показателями, ходовыми характеристиками и достаточной проходимостью.

Конструкция роботов:

- Конструктивно используемые роботы представляют собой трехколесные платформы с двумя ведущими и одним флюгерным колесом.
- Ведущие колеса приводятся в движение при помощи серводвигателей, на которые подаются напряжение питания и пульсирующий сигнал управления. Угол поворота колеса относительно своей оси определяется длительностью импульса.
- На платформе закреплена плата с микроконтроллером, регулирующим работу робота посредством генерации сигналов управления. Плата предусматривает возможность сборки дополнительных схем и включения необходимых электронных компонентов.
- В рамках адаптации к выполнению поставленной задачи роботы были оснащены модулями радиосвязи, а ведущий — дополнительно двумя парами ИК-датчиков.

Математический аппарат

Оптимизация траектории движения отдельного агента предполагает, во-первых, движение в обход препятствия по сглаженной траектории, и, во-вторых, такое движение, когда агент имеет возможность начать маневр уклонения задолго до непосредственной встречи с препятствием. Иными словами, чем раньше знания об окружающей среде будут доступны конкретному агенту, тем качественней можно будет определить оптимальную для движения траекторию. С другой стороны, агент, не имеющей доступа к информации об окружающих объектах до тех пор, пока те не станут видимы его локальным датчикам, не сможет в полной мере скоординировать свои действия таким образом, чтобы двигаться по кратчайшему пути.

Таким образом, группа из двух агентов, работающих в неизвестной среде, способна определить оптимальную траекторию движения, если функции между агентами будут распределены следующим образом:

1. Первый агент в процессе движения проводит мониторинг рабочей среды и отвечает за то, чтобы вся группа была своевременно обеспечена информацией об изменении характера его движения. В связи с тем, что подобная модель движения не предполагает возможности уклониться от столкновения с препятствием по гладкой траектории, допускается работа агента с ограниченной маневренностью (только большие углы поворота).
2. Второй агент, опираясь на получаемые данные о передвижении ведущего агента, имеет возможность анализировать массивы его координат и выстраивать собственную траекторию движения, находясь при этом на достаточном расстоянии от обнаруженного препятствия. В этом случае описанная модель не предполагает обязательного наличия у ведомого агента возможности детектировать препятствия самостоятельно, однако он должен быть обеспечен способностью поворачиваться на любые углы с допустимой точностью.

Основываясь на таком способе распределения функций, алгоритм оптимизации траектории будет заключаться в том, чтобы поставить траектории движения первого агента в соответствие новую траекторию, более короткую и гладкую. Или, иными словами, каждой паре координат ведущего агента поставить в соответствие другую пару, придерживаясь некоторых критериев оптимизации:

$$p \to p',$$
$$(x_i, y_i) \to (x_i', y_i').$$

Критерии оптимизации при этом можно обозначить следующим образом:

новый массив координат содержит координаты, каждая из которых отстоит от своих соседних координат на меньшее расстояние, чем соответствующая ей в исходном массиве;

$$(x_{i+1}' - x_i') \to \min,$$
$$(y_{i+1}' - y_i') \to \min;$$

новый массив координат содержит координаты, каждая из которых отстоит от соответствующей ей координаты из исходного массива на минимальное допустимое расстояние;

$$(x_i' - x_i) \to \min,$$
$$(y_i' - y_i) \to \min.$$

Т. о., задача оптимизации сводится к нахождению минимума двух функций. В таком случае, удобным механизмом для составления оптимальной траектории будет использование метода градиентного спуска, в основе

которого лежит идея того, что при некоторых начальных условиях максимум (минимум) функции быстрее всего будет найден при движении в направлении градиента (антиградиента). Алгоритм градиентного спуска для данной задачи будет выглядеть следующим образом:

$x'_{i+1} = x'_i + \alpha(x_i - x'_i)$ — первый критерий оптимизации;

$x'_i = x'_i + \beta(x'_{i+1} + x'_{i-1} - 2x'_i)$ — второй критерий оптимизации;

$1 > \alpha > \beta > 0$ — коэффициенты оптимизации, выбранные согласно с приоритетом обозначенных в алгоритме функций.

Заключение

Основным результатом данной работы является решение задачи траекторного движения группы роботов в неструктурированной внешней среде с возможностью отдельных агентов преодолевать препятствия по гладким траекториям. Эта способность основана на разработанной математической модели с учетом функциональных возможностей используемых при апробации системы роботов. Решение данной задачи позволяет проводить более глубокие исследования в вопросах мониторинга, разведки и исследования областей, недоступных для человека.

Литература

1. Robotics with the Boe-Bot. Student Guide. version 2.2/ © 2003–2004 by Parallax Inc.
2. Getting Started with XBee RF Modules. A Tutorial for BASIC Stamp and Propeller Microcontrollers. version 1.0/ Hebel M., Bricker G., Harris D. © 2010 by Parallax Inc.

Мультиагентные и нейросетевые технологии в информатике

Тимофеев
Адиль Васильевич

д.т.н.
профессор
заведующий лабораторией информационных технологий
в управлении и робототехнике СПИИРАН

НЕЙРОСЕТЕВОЙ И ИММУНОКЛЕТОЧНЫЙ ПОДХОДЫ К РАСПОЗНАВАНИЮ СЕТЕВЫХ АТАК

А. А. Браницкий

студент кафедры информатики;
alexander.branitskiy@gmail.com

А. В. Тимофеев

профессор кафедры информатики;
tav@iias.spb.su

Санкт-Петербургский государственный университет

Аннотация: В данной работе описаны 2 сравнительно новых подхода в области обнаружения сетевых атак — система из статически обученных нейронных сетей и система из динамически обучающихся карт Кохонена (иммунных клеток). Представленные математические модели особенно хорошо зарекомендовали себя при решении задач классификации данных. Разработанное на их основе программно-аппаратное средство предназначено для анализа сетевого трафика на наличие аномальных соединений, а также для их классификации при обмене данными по компьютерным сетям, использующим стек протоколов TCP/IP. Одним из наиболее важных критериев при тестировании подобного программного обеспечения является его способность обнаружения неизвестных типов атак. Благодаря тому, что нейросети представляют собой мощный инструмент аппроксимации рассматриваемой многомерной области, системы обнаружения вторжений, построенные на основных принципах их работы, имеют достаточно высокую степень эффективности в распознавании ранее неизвестных типов злоупотреблений. Это доказывают и приведенные в работе результаты проверки качества обнаружения сетевых атак нейронными сетями и картами Кохонена.

Введение

Под нейронными сетями подразумеваются вычислительные структуры, которые моделируют простые биологические процессы, обычно ассоциируемые с процессами человеческого мозга. Они представляют собой распределенные и параллельные системы, способные к адаптивному обучению путем анализа положительных и отрицательных воздействий. Нервная система и мозг человека состоят из нейронов, соединенных между собой нервными волокнами. Нервные волокна способны передавать электрические импульсы между нейронами. Все процессы передачи раздражений от кожи, ушей и глаз к мозгу, процессы мышления и управления действиями — все это реализовано в живом организме как передача электрических импульсов между нейронами [1].

Естественная иммунная система представляет собой сложную адаптивную систему, состоящую из нескольких функционально различных

компонентов и эффективно использующую различные механизмы защиты от внешних патогенов. Основная роль иммунной системы заключается в распознавании всех клеток организма и классификации их как «своих» и «чужих». Чужеродные клетки подвергаются дальнейшей классификации с целью стимуляции защитного механизма соответствующего типа. В процессе эволюции иммунная система обучается различать внешние антигены (например бактерии и вирусы) и собственные клетки или молекулы организма [2].

Построение нейросетевой системы для обнаружения сетевых атак

В качестве обучающего множества выступает база данных KDD Cup 99 [3]. Эта база содержит около 5 млн. записей о соединениях. Каждая запись представляет собой образ сетевого соединения, включает 41 параметр сетевого трафика и промаркирована как «атака» или «не атака». Например, первый параметр определяет длительность соединения, второй — используемый протокол, третий — целевую службу [4].

Для обнаружения и классификации сетевых атак используются трехслойные нейронные сети, обученные по правилу обратного распространения ошибки [5]. В процессе обучения на вход каждого нейросетевого детектора подаются сформированные наборы параметров сетевых соединений, и осуществляется корректировка весовых коэффициентов для распознавания одного типа атаки и нормального соединения. Для обучения нейронных сетей используются равные по объему выборки положительного и отрицательного трафиков.

Структура нейронной сети на языке программирования С следующая:

```
struct {
    struct fann *ann;
    enum type_attack attack;
    pthread_t thread;
    int index;
};
enum type_attack {
    back, neptune, pod, smurf, teardrop, ipsweep,nmap,
    portsweep, satan, normal};
```

Она состоит из полей:

- указателя `ann` на структуру `struct fann`, которая определена в библиотеке FANN (Fast Artificial Neural Network) [6];
- переменной `attack` типа перечисления `enum type_attack`, задающей тип атаки, которую обучена распознавать данная нейросеть;
- переменной `thread` типа `pthread_t`, которая служит в роли идентификатора потока, в котором исполняется данная нейронная сеть;

- переменной `index` типа `int`, которая задает положение (индекс) данной переменной типа `struct neural_network` в статическом массиве.

Построение иммуноклеточной системы для обнаружения сетевых атак

Типовая схема иммуноклеточной системы и основные принципы ее работы описаны в работе [2]. Иммунные клетки представляют собой самоорганизующиеся двумерные карты Кохонена, обученные по конкурентному алгоритму. Существенное отличие данного подхода от нейронных сетей заключается в постоянной модификации весовых коэффициентов, а также взаимодействие иммунных клеток в целях повышения эффективности обнаружения неизвестных типов атак.

За счет генерации случайной обучающей выборки создаются иммунные детекторы, разнообразные по своей структуре и способные реагировать на любую аномальную сетевую активность. В процессе обучения иммунные клетки приобретают способность правильно распознавать чужеродные антигены — сетевые атаки. После настройки весовых параметров карты Кохонена тестируются на корректность распознавания нормальных соединений, для того чтобы уменьшить число ложных срабатываний и проверить качество распознавания атак.

Каждая иммунная клетка наделена конечным сроком жизни, в течение которого она находится в стадии обнаружения сетевых атак. Если на протяжении данного ей времени жизни она не обнаруживает аномалий, то на ее место приходит другая структурно отличная иммунная клетка. В противном случае, если иммунный детектор обнаружил подозрительную сетевую активность, то его срок жизни увеличивается, происходит сигнализирование об обнаруженной атаке, а также осуществляется запись параметров данного соединения в базу для обучения текущего и последующего поколений иммунных детекторов.

Структура иммунной клетки на языке программирования С выглядит следующим образом:

```
struct kohonen_card {
    struct {
          double weight[DIM_IN];
    } out_neurons[DIM_OUT_Y][DIM_OUT_X];
    enum type_attack attack;
    struct {
          int x, y;
    } coordinate_winner;
    pthread_t thread;
    double life_time;
    enum type_status status;
```

```
        bool is_valid;
        int index;
        u_long quantity_attacks;
        bool repetition;
    };
    enum type_status {
        creating_status, initializing_status,
        training_status, testing_status, running_status,
        joining_status, loading_status, saving_status,
        destroying_status};
```

Она состоит из полей:

- двумерного массива `out_neurons[DIM_OUT_Y][DIM_OUT_X]`, играющего роль нейронов выходного слоя карты Кохонена с весами, представленными как элементы массива `weight[DIM_IN]` типа `double`;
- переменной `attack` типа перечисления `enum type_attack`, задающей тип атаки, которую обучена распознавать данная иммунная клетка;
- переменной `coordinate_winner`, содержащей два целочисленных поля `x` и `y`, задающих координаты нейрона-победителя;
- переменной `thread` типа `pthread_t`, которая служит в роли идентификатора потока, в котором исполняется данная иммунная клетка;
- переменной `life_time` типа `double`, задающей текущее оставшееся время жизни данной иммунной клетки;
- переменной `status` типа enum `type_status`, задающей текущее состояние, в котором находится данная иммунная клетка;
- переменной `is_valid` типа `bool`, которая характеризует, удачно ли завершила стадию тестирования данная иммунная клетка и готова ли она к распознаванию атак;
- переменной `index` типа `int`, которая задает положение (индекс) данной переменной типа `struct kohonen_card` в статическом массиве;
- переменной `quantity_attacks` типа `u_long`, задающей число обнаруженных и распознанных данной иммунной клеткой атак;
- переменной `repetition` типа `bool`, значение которой показывает, обучается ли данная иммунная клетка первый раз или повторно.

Процесс обнаружения сетевых атак

Процесс обнаружения сетевых атак следующий. Пакеты, поступающие на сетевую карту, перехватываются и обрабатываются анализатором пакетов. Из поля данных и заголовка каждого пакета выделяются и вычисляются необходимые для классификации соединений параметры. Собираются некоторые статистические сведения об активных в данный момент соединениях. После сбора всех необходимых показаний о каждом запущенном соединении набор вычисленных параметров «подвергается» текстовой обработке: каждая

отдельная составляющая набора параметров отображается в соответствующий ей числовой эквивалент и нормализуется. Опционально (необязательно) полученный набор параметров сжимается по методу главных компонент [7].

Преобразованные векторы подаются на вход обученных нейронных сетей, которые в свою очередь формируют линейную комбинацию каждого вектора со своими весовыми параметрами, которая выступает в роли аргумента функции активации нейронной сети. На основании полученного значения принимается решение о принадлежности конкретного соединения тому или иному классу атак.

В случае иммунных клеток полученные векторы сравниваются с весовым вектором нейрона-победителя каждой карты Кохонена. В зависимости от значения нормы разности этих векторов данное соединение попадает в определенный класс атак.

Результаты проведения экспериментов

При проведении этапа эмулирования сетевых атак были получены следующие результаты, представленные в табл. 1, 2 (левый столбец — тип эмулируемого соединения, верхняя строка — тип нейронной сети или иммунной клетки, их пересечение — процентное количество правильно распознанных атак соответствующего типа соответствующими нейронными сетями или иммунными клетками). В частности последняя строка в обоих таблицах показывает процентное число ложных срабатываний (когда типично нормальное соединение принимается за атаку). В общей сложности показатели эффективности были проверены на 4 680 621 образце сетевых соединений.

Таблица 1

Показатели эффективности обнаружения атак нейронными сетями

	back	neptune	pod	smurf	teardrop	ipsweep	nmap	portsweep	satan
back	99.7%	0%	0%	0%	0%	0%	0%	0%	0.2%
neptune	7.9%	100%	0%	0%	0%	0%	80.8%	100%	19.1%
pod	0.4%	0.4%	99.6%	0%	32.1%	1.5%	1.9%	0.4%	1.5%
smurf	0%	0%	0%	99.8%	0%	0%	99.9%	0%	100%
teardrop	0%	0%	77.3%	0.1%	99.9%	0%	0.1%	0%	0.1%
ipsweep	0%	0%	0%	0%	0%	99.8%	92.2%	1.5%	1.7%
nmap	0%	44.6%	0%	0%	0%	44.1%	99%	44.6%	3%
portsweep	0.4%	9.9%	0%	0%	0%	0%	0.1%	99.6%	90.2%
satan	0%	88.6%	0%	0%	0.1%	0.1%	3.6%	88.7%	99.5%
normal	0.4%	0%	0%	0%	0%	0.5%	1.1%	0.1%	1.3%

Таблица 2

Показатели эффективности обнаружения атак иммунными клетками

	back	neptune	pod	smurf	teardrop	ipsweep	nmap	portsweep	satan
back	99.6%	0%	0.5%	0%	55%	0%	0.3%	0%	0%
neptune	0%	80.9%	0%	0%	0%	0%	0.1%	19.1%	9.5%
pod	10.2%	0.4%	98.9%	0%	33.7%	61%	67.8%	0%	0%
smurf	0.1%	0%	0%	100%	0.1%	0%	0%	0%	0%
teardrop	15.1%	0%	89.7%	0.1%	95.9%	1.7%	0%	0%	0%
ipsweep	4.6%	0%	92.6%	0%	0%	93%	92.5%	0%	0%
nmap	12%	0%	55.2%	0%	9.5%	49.6%	97.1%	0%	0%
portsweep	0.1%	1.9%	0.1%	0%	0%	0.3%	0.1%	91.7%	22.2%
satan	1.9%	0%	9.3%	1%	5.5%	2.5%	0%	73.1%	88.1%
normal	87.4%	0%	27.5%	0.5%	46.1%	7.4%	5.8%	0%	0%

Заключение

Результаты проведенных экспериментов показывают, что применение рассмотренных подходов позволяет добиться высоких показателей эффективности при обнаружении сетевых атак. Использование нейросетевого и иммуноклеточного методов расширяет функциональность систем, позволяя им обнаруживать неизвестные атаки.

Литература

1. *В. В. Круглов, В. В. Борисов.* Искусственные нейронные сети: Теория и практика. 2-е изд. М.: Горячая линия-Телеком, 2002. 382 с.
2. *Д. Дасгупта.* Искусственные иммунные системы и их применение. М.: Физматлит, 2006. 344 с.
3. KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
4. Технологии обнаружения сетевых атак. http://www.bstu.by/~opo/ru/uni/bstu/science/ids
5. *С. Хайкин.* Нейронные сети: Полный курс. 2-е изд. М.: Издательский дом «Вильямс», 2006. 1104 с.
6. Fast Artificial Neural Network. http://leenissen.dk/fann/wp/
7. *С. А. Айвазян, В. С.* Мхитарян Прикладная статистика. Основы эконометрики. 2-е изд., испр. Т. 1. М.: Юнити-Дана, 2001. 656 с.

РАСПОЗНАВАНИЕ ПРОГРАММНЫХ ПОТОКОВ

А. Р. Ханов

awengar@gmail.com

Руководитель: ***М. В. Баклановский***

Санкт-Петербургский государственный университет

Аннотация: В данной работе исследуется возможность идентификации процессов и потоков по информации о системных вызовах, которые они совершают. Были выработаны специальные признаки, позволяющие выделить особенности, характерные для потоков. Показаны результаты экспериментов, доказывающих состоятельность алгоритма идентификации процессов на основе этих признаков.

Задача идентификации программ является важнейшей практической проблемой, являющейся предметом множества исследований. В [1] применен метод составления признаков по определенным наборам API-вызовов и распознавание с помощью дискриминантной функции для выделения вредоносных программ. При этом достигается достаточно высокая точность.

Наша задача состоит в том, чтобы научиться распознавать уже запущенные программы по их поведению в системе. Сначала мы получаем некоторую информацию о работающей программе, на ее основе мы учимся различать эту программу среди других. Затем, получив информацию о тестируемой программе, мы находим среди уже изученных программ ту, которая больше всего похожа на тестируемую.

В качестве информации, «следов», о работающей программе в системе были взяты списки системных вызовов, которые она производит. Мы не ставим целью доказательство функционального соответствия двух разных программ. Программы могут производить одни и те же действия в системе и давать различные последовательности системных вызовов. Мы также не знаем заранее, насколько эта информация характерна для процессов и можно ли их различить, основываясь лишь на ней.

За одну секунду программа может производить до нескольких тысяч системных вызовов. Для записи логов этих вызовов был использован метод, описанный в [2]. Из этих последовательностей нам необходимо сформировать признаки, которые позволят отличать одну программу от другой.

Мы записываем последовательность системных вызовов для процесса и затем выделяем из них вызовы каждого потока. Эти последовательности обладают высокой энтропией, что говорит о том, что информация, содержащаяся в них избыточна. Действительно, в самих цепочках присутствуют повторения одних и тех же последовательностей. Задача усложняется тем,

что программа существует в системе и последовательность ее вызовов сильно зависит от работы самой системы. Оценка контекстной энтропии по алгоритму PPM показала, что даже в состоянии, когда программа не должна проявлять какую-либо активность, возникают неожиданные вызовы с высокой частной энтропией, хотя сами вызовы являются достаточно предсказуемыми по своему контексту.

Чтобы преодолеть эти сложности, было решено не строить модель всего потока в целом, а искать специальные цепочки вызовов, называемые нами термы, которые повторяются в программе несколько раз. Эти цепочки должны описывать особенности работы программы. У каждой такой цепочки вызовов можно выделить следующие характеристики: длина, число встречаемости в данной цепочке, минимальный, средний и максимальный период встречаемости. Так как таких цепочек в программе очень много, были выработаны эвристические правила для них:

1) чем длиннее терм, тем больше он содержит информации о своем потоке;
2) чем большее число раз он встречается, тем вероятнее он встретится в тестовом потоке, который будет эквивалентен данному.

Эти правила в некоторой степени друг другу противоречат: длинные термы встречаются реже коротких. Здесь нужно искать баланс и брать достаточно длинные и частые термы. Среди всех термов, встречающихся не реже фиксированного числа раз, мы выберем непересекающееся подмножество, начиная с самых длинных. Мы также можем брать лишь необходимое число термов или фиксировать их минимальную длину.

Период встречаемости позволяет нам оценивать, сколько вызовов нужно взять из тестируемого потока, чтобы мы могли сказать, похож ли он на изученный поток.

Для того, чтобы оценить степень схожести тестируемого потока с одним из изученных, мы должны искать термы в качестве подпоследовательностей. Количество встреченных термов и их длины и будут характеризовать степень схожести.

Проведенные тесты показали, что с помощью построенных признаков, множества термов, удается различать процессы. Были составлены множества термов по самым длинным потокам 12 типичных программ ОС Windows 7 приблизительно за 10 минут их работы, за которые они совершили порядка миллиона вызовов. В дальнейшем на новых экземплярах этих программ были записаны тестовые последовательности вызовов. Новые экземпляры процессов удалось распознать по количеству встретившихся термов в самых длинных потоках уже по 60 000 вызовам. Это показывает, что построенные признаки хорошо описывают особенности работы процессов.

Тесты, проведенные по пяти наибольшим потокам, показали иной результат. Оказалось, что более короткие потоки одних процессов путаются с потоками других. Это говорит не столько о слабости метода, сколько о ра-

боте самих этих процессов, которые, по всей видимости, исполняют один и тот же код в этих потоках.

Таким образом, был построен признак, позволяющий выявлять схожесть у потоков. Было показано, что процессы могут быть идентифицированы лишь по их главному потоку. В дальнейшем будет продолжено изучение термов и будет строиться более четкая теория по выявлению особенностей работы потоков с их помощью.

Литература

1. *M. Ahmadi, A. Sami, H. Rahimi, B. Yadegari.* Iterative System Call Patterns Blow the Malware Cover// Security for The Next Generation 2011.
2. *Одеров Р. С., Тенсин Е. Д.* Способы размещения своего кода в ядре ОС Microsoft Windows Server 2008 // Сборник трудов межвузовской научно-практической конференции «Актуальные проблемы организации и технологии защиты информации». СПб., 2011. С.100–102.

УПРАВЛЕНИЕ ИНФОРМАЦИОННЫМИ ПОТОКАМИ И ПАКЕТАМИ ДАННЫХ В ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМАХ [1]

А. М. Бакурадзе

ведущий электроник лаборатории информационных технологий в управлении и робототехнике;
klauzert@yandex.ru

Санкт-Петербургский институт информатики и автоматизации РАН

Аннотация: Автором рассматриваются структура, функции локальных и глобальных телекоммуникационных систем. Рассматриваются основные характеристики и стратегии пакетной обработки данных в телекоммуникационных системах. Описываются проблемы управления потоками данных в телекоммуникационных системах.

Введение

В настоящее время в области управления потоками информации в глобальных телекоммуникационных системах (ТКС) имеется много нерешенных проблем: недостаточно многопротокольных и масштабируемых систем управления, большинство средств контроля осуществляют только наблюдение (мониторинг) за работой ТКС, многие системы только локально управляют отдельными элементами и не анализируют способность ТКС в целом к управляемой передаче информации на большие расстояния.

1. Структура, функции и классификация локальных и глобальных ТКС

Глобальная ТКС служат для объединения удалённых компьютеров в единую информационную сеть с помощью каналов и средств связи и управления потоками данных между компьютерами как узлами ТКС в изменяющейся информационной среде. При этом каждый компьютер — узел глобальной сети целесообразно рассматривать как локальную информационную управляемую систему, состоящую из следующих основных (базовых) компонент:

1. локальная информационная система (ЛИС);
2. локальная система управления (ЛСУ);
3. локальная коммуникационная система (ЛКС).

[1] Работа выполнена при поддержке грантов РФФИ № 09-08-00767-а, РФФИ – ГФЕН Китая № 10-08-91159-а, РФФИ № 12-08-01167-а, Программы № 14 Президиума РАН и издательского гранта РФФИ № 12-08-07022-д.

ЛИС представляет собой информационный объект управления и источник информационной обратной связи для ЛСУ. Она состоит из локальных БД, БЗ и прикладных (управляемых) программ их обработки.

ЛСУ представляет собой операционную систему (ОС) компьютера как узла ТКС и локальную систему управления БД и БЗ, хранящихся в ЛИС.

ЛКС включает в себя порты и шины данных для обмена информацией с другими компьютерами-узлами сети, а также терминалы для связи компьютера с человеком-пользователем и окружающей средой.

Таким образом, глобальная ТКС представляет собой распределённую сеть большого количества удалённых компьютеров или локальных ТКС, играющих роль узлов глобальной ТКС, которые имеют собственные (локальные) информационные ресурсы, средства связи и управления. Все эти компьютеры или локальные ТКС, т. е. узлы глобальной сети, имеют физические (электрические) каналы связи между собой, по которым передаются потоки данных и сигналы управления ими.

Структурная схема глобальной ТКС с управляемым доступом к информационным ресурсам, представленная на рис. 1, аналогична структуре узловых локальных ТКС. Она включает в себя следующие взаимосвязанные компоненты:

- глобальная информационная система (ГИС);
- глобальная система управления (ГСУ);
- глобальная коммуникационная система (ГКС).

ГИС представляет собой распределённые и удалённые на значительные расстояния узловые ЛИС, образующие информационные ресурсы глобальной ТКС.

ГСУ управляет потоками данных и распределёнными компонентами глобальной ТКС.

ГКС состоит из каналов связи и другого коммуникационного оборудования, связывающего узловые компьютеры или локальные ТКС в глобальную сеть, охватывающую значительные пространства.

Диаметром ТКС будем называть расстояние между наиболее удалёнными компьютерами (узлами) сети. Если этот диаметр достаточно мал (например, не превышает сотен метров), то ТКС называется локальной.

ТКС с большим диаметром, объединяющие компьютеры или локальные ТКС как узлы сети, расположенные в различных городах и странах, называются глобальными ТКС. Примером простейшей глобальной ТКС может служить ТКС, состоящая из двух удалённых на большое расстояние локальных ТКС, связанных между собой с помощью маршрутизаторов. Другим примером глобальной ТКС с очень большим диаметром является глобальная сеть Internet.

ТКС, узлами которой являются однотипные компьютеры, будем называть однородными (гомогенными). Если же узловые компьютеры ТКС разнотипны, то ТКС является разнородной (гетерогенной).

Локальные ТКС часто бывают однородными. Различные локальные ТКС могут играть роль узлов глобальных ТКС. Глобальные ТКС, как правило, разнородны.

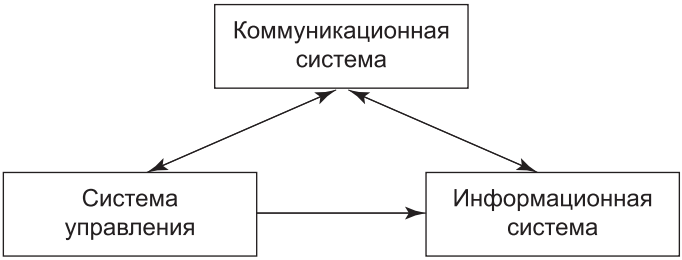


Рис. 1. Общая структура локальной и глобальной ТКС с управляемым доступом к информационным ресурсам

2. Функции и способы коммутации каналов связи в глобальных ТКС

В узлах ТКС обычно имеются коммутационные процессоры, соединяемые через модемы с каналами связи. Эти процессоры могут выполнять функцию коммутации каналов между абонентами (пользователями и узлами) ТКС, организуя канал постоянной прямой и обратной связи на время обмена информацией.

Один из способов организации связи, называемый коммутацией каналов, приводит к неэффективному использованию глобальной ТКС. Это связано с тем, что во время сеанса связи между потоками данных, которыми обмениваются абоненты ТКС, могут возникать значительные паузы. При этом канал связи оказывается занятым. Однако он фактически может быть недогруженным даже при непрерывной передаче информации, если скорость передачи данных мала (например, при передаче запросов с терминала).

Для более интенсивной загрузки каналов в ТКС можно использовать другой способ связи, называемый коммутацией сообщений. В этом случае сообщения, которыми обмениваются абоненты ТКС, запоминаются в узлах (коммутационных процессорах) ТКС и с большой скоростью передаются между соответствующими узлами в виде потоков данных. Недостатком такого способа является то, что требуемая память обычно выбирается из расчёта передачи самых длинных сообщений. Поэтому необходимая память фактически избыточна.

Для устранения этого недостатка в современных ТКС используется коммутация пакетов. Это — самый быстрый способ связи и передачи данных. Он основан на декомпозиции сообщений на пакеты данных стандартной (одинаковой или различной) длины, которые могут передаваться независимо друг от друга и даже по различным каналам связи ТКС. Благодаря такому

параллелизму увеличивается общая скорость передачи данных в глобальных ТКС.

Реализация этого способа в так называемых концентраторах пакетов предъявляет дополнительные требования к системе управления ТКС. Среди них важнейшими требованиями являются:

- необходимость автоматической декомпозиции сообщений на пакеты (одинаковой или различной длины),
- агрегирование (сборка) из пакетов передаваемых сообщений,
- оптимизация маршрутов передачи пакетов в зависимости от загрузки каналов и узлов ТКС,
- адаптация управления ТКС к непредсказуемо изменяющемуся трафику.

При проектировании и создании ТКС приходится решать две основные задачи обработки информации и управления потоками данных.

Первая задача заключается в организации потоков данных для обмена информационными ресурсами между узлами (компьютерами) ТКС, а также между ними и пользователями ТКС.

Вторая задача — это организация и реализация управления потоками данных в ТКС.

3. Организация и управляемая передача потоков данных в ТКС

Существует несколько способов организации обмена информационными ресурсами между узлами ТКС.

Для организации информационных связей между любыми двумя узлами ТКС можно использовать общие информационные шины данных, к которым присоединяются все узлы (компьютеры) сети. Реализация этого способа наталкивается на значительные трудности (радикальные изменения локальных ОС и т. п.). По существу она сравнима по сложности с созданием новой мультипроцессорной компьютерной системы. Поэтому такой подход практически неприемлем для глобальных разнородных (гетерогенных) ТКС.

Более простой способ организации информационных обменов в ТКС заключается в объединении узлов (компьютеров) через специально подключённое к ним оперативное запоминающее устройство (ОЗУ) — память адресов. Это ОЗУ играет роль «почтового ящика», куда каждый узел ТКС может послать сообщение, адресованное любому другому узлу, или получить сообщение, адресованное ему самому. В этом случае необходим специальный интерфейс для прямой и обратной связи с «почтовым ящиком» ТКС, т. е. с её ОЗУ.

Ещё более простым способом обмена информацией является установление прямых связей между сетевыми компьютерами (узлами ТКС) с помощью парных адаптеров и групповых коммутаторов типа «канал — канал».

Этот способ позволяет реализовывать любую (например, кольцевую) топологию связей между узлами ТКС. Поэтому такой способ часто применяется на практике.

В современных ТКС для обмена информацией между компьютерами широко используются средства и стандарты телефонной и телеграфной связи, а также мультимедийные технологии. Поэтому возникла необходимость в создании многоуровневой системы протоколов для представления и передачи потоков данных в ТКС.

Для управления потоками данных в ТКС обычно используются сетевые принципы централизованного и децентрализованного информационного управления.

При централизованном управлении ТКС обычно выделяется некоторый управляющий центр — центральный компьютер, который координирует (например, с помощью сетевого администратора) работу остальных компьютеров ТКС и через них управляет различными потоками данных в ТКС. Главным недостатком централизованных систем управления потоками данных является их полная зависимость от «центра» и связанная с этим низкая надёжность, так как выход из строя центрального компьютера нарушает работоспособность всей ТКС.

При децентрализованном управлении ТКС каждый узловой компьютер локально управляет с помощью ЛСУ потоками данных на основе обмена информацией с другими узловыми компьютерами ТКС. Поэтому децентрализованные системы управления, распределённые по всей ТКС, более надёжны и независимы.

4. Основные характеристики и стратегии пакетной обработки данных в ТКС

Данные, передаваемые по ТКС, представляют собой закодированные сообщения. Они измеряются числом битов передаваемой информации, определяющим длину данных.

Передаваемые данные часто разбиваются на блоки постоянной или переменной длины, называемые пакетами. Такая декомпозиция данных используется в ТКС с коммутацией пакетов. В этом случае пакеты передаются от узла-источника информации к узлу-получателю данных по некоторому маршруту, разделяя между собой доступные каналы и средства связи ТКС.

В случае ТКС с коммутацией каналов между источником информации и её получателем устанавливается специальный (выделенный) маршрут передачи данных от одного узла к другому без разделения коммуникационных ресурсов.

Пакеты, поступающие на вход узла ТКС, в процессе движения к узлу-получателю, накапливаются (запоминаются) и обрабатываются с целью выбора

доступного ими наилучшего канала связи. Затем они считываются в этот канал, как только наступит время их передачи.

Время задержки, т.е. время ожидания пакетом своей передачи, является одним из важнейших показателей работы ТКС. Это время зависит от времени обработки пакета в узле, длины пакета, пропускной способности канала связи, интенсивность поступления пакетов и дисциплины (стратегии) их обслуживания в ТКС.

Пропускная способность канала связи — это число, которое ограничивает общее количество пакетов определенной длины, передаваемых по каналу за заданное время. Для определения этого числа обычно используется следующая формула

$$\mu = \frac{n_d}{T}, \tag{1}$$

где n_d — максимальное число передаваемых пакетов определенной длины d, T — время передачи. Величина m измеряется числом пакетов, передаваемых в секунду (пакет/с).

Интенсивность поступления пакетов l определяется скоростью в единицах пакет/с.

Канал связи характеризуется скоростью s передачи данных, измеряемой обычно в бит/с. Если, например, канал связи, передающий пакеты длиной $d = 10^3$ бит, работает со скоростью $s = 2 \cdot 10^3$ бит/с, то он сможет передавать пакеты, поступающие с интенсивностью $\lambda = 2$ пакетов/с.

Обычно средняя длина пакета равна $\frac{1}{\mu'}$ и измеряется в битах. Тогда пропускная способность канала связи определяется по формуле:

$$\mu = \mu' s.$$

и определяется в единицах бит/пакет.

В общем случае, если $\lambda \geq m$, входные пакеты будут накапливаться и возникнет очередь.

Для количественной оценки таких ситуаций используется показатель

$$\rho = \frac{\lambda}{\mu}, \tag{2}$$

называемой коэффициентом использования канала связи или интенсивностью сетевой нагрузки. При $\rho \geq 1$ возникают перегрузки («скученность» пакетов) и возрастает время задержки. В этих случаях поступающие пакеты начинают блокироваться.

Для обслуживания очереди из интенсивно (с перегрузкой) поступающих пакетов обычно используются следующие стратегии (дисциплины обслуживания) [1–3]:

- обслуживание в порядке поступления;
- обслуживание в обратном порядке, когда первым обслуживается пакет, который поступил последним;
- обслуживание с приоритетом.

Эти стратегии основываются на вероятностном подходе, связанном с анализом статистики входящих пакетов и распределением их длин в терминах теории массового обслуживания (queueing theory) [2,3].

5. Динамическая модель получения и передачи пакетов данных в ТКС

Пакеты поступают на входы узлов ТКС или отправляются с выходов узлов, образуя изменяющиеся с течением времени информационные потоки.

Потоки данных в ТКС можно рассматривать как дискретные целочисленные потоки, состоящие из поступающих на вход и передаваемых по каналам связи пакетов. Эти потоки могут быть детерминированными, случайными или непредсказуемыми.

Независимо от природы возникновения потоков данных, представляющие их пакеты последовательно поступают на входы узлов ТКС. Они запоминаются и ждут своего обслуживания (передачи) в накопителе соответствующего узла ТКС с ограниченной памятью. Затем эти потоки в некоторые дискретные моменты времени (вообще говоря, управляемые) передаются по свободным каналам связи в другие узлы ТКС.

Обозначим через $z(t)$ общее число входных пакетов, поступивших в накопитель (буфер) узла ТКС к моменту времени t. Аналогично введём функцию $y(t)$, определяющую общее число выходных пакетов, отправленных к моменту времени t в канал связи после ожидания в накопителе. Тогда функция

$$l(t) = z(t) - y(t),\ t \in [t_0,\ t), \tag{3}$$

определит текущее число пакетов, ожидающих своего обслуживания (передачи) к моменту времени t.

Пакеты данных последовательно поступают на вход узла ТКС в дискретные моменты времени $t_1,\ t_2,\ \ldots$. В эти моменты времени значения целочисленной функции $z(t)$ возрастает на 1, т. е.

$$z(t_{k+1}) = z(t_k) + 1,\ k = 0,\ 1,\ 2,\ \ldots\ . \tag{4}$$

Аналогичным образом в некоторые дискретные моменты времени $t'_1, t'_2, \ldots$ рекуррентно изменяется функция $y(t)$, т. е.

$$y(t'_{k+1}) = y(t'_k) + 1,\ k = 0, 1, 2, \ldots . \tag{5}$$

Моменты отправления пакетов данных в ТКС образуют монотонно возрастающую последовательность вида

$$t_0 \le t'_1 < t'_2 < t'_3 < \ldots . \tag{6}$$

Моменты отсылки пакетов t'_k могут совпадать с моментами их поступления t_k в тех случаях, когда канал связи свободен.

Ради простоты предположим, что входные пакеты обслуживаются в порядке их поступления в узел ТКС. Однако следует отметить, что рассматриваемые модели и результаты справедливы и при других стратегиях (дисциплинах) обслуживания входных пакетов .

Примеры возможного изменения функции получения пакетов данных $z(t)$ и передачи пакетов данных $y(t)$ по мере последовательного поступления входных пакетов в узел ТКС представлены на рис. 2. Здесь функция $z(t)$ изображена сплошной линией, а функция $y(t)$ — пунктирной линией.

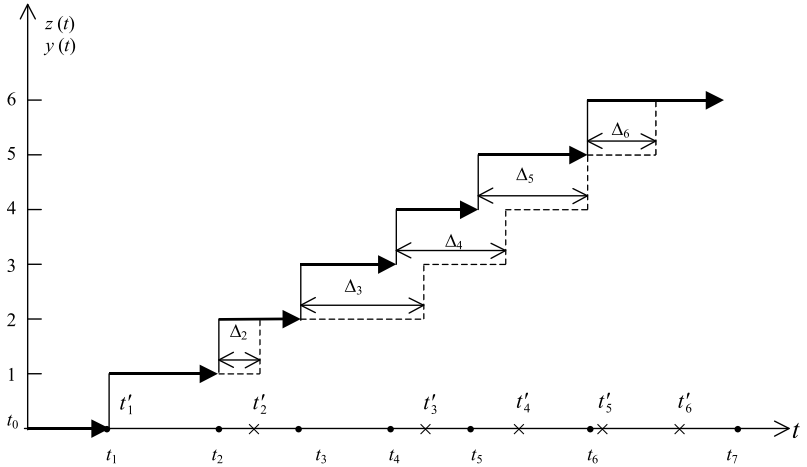


Рис. 2. Динамика получения и передачи пакетов данных в ТКС

На рис. 2 показаны также времена ожидания (задержки) пакетов данных $\Delta_1, \Delta_2, \ldots, D_6$. Эти величины определяют период времени, в течение которого соответствующий пакет находится в накопителе узла ТКС с момента его поступления t_k до момента его отправки t'_k, причём $t'_k \ge t_k$.

Наряду с начальным моментом t_0 рассмотрим некоторый конечный момент t_T. Тогда общее число поступлений входных пакетов в узел ТКС на интервале времени $T = t_T - t_0$ можно вычислить по формуле

$$n(T) = z(t_T) - z(t_0). \tag{7}$$

В примере на рис. 2 общее время поступления потоков данных $n(T) = 6$.

Аналогично определяется общее число отсылок (передачи) пакетов данных за время T

$$m(T) = y(t_T) - y(t_0). \tag{8}$$

Средние скорости поступления и передачи входных пакетов определяются соответственно по формулам

$$\lambda(T) = \frac{n(T)}{T}, \tag{9}$$

$$\beta(T) = \frac{m(T)}{T}. \tag{10}$$

Рассмотрим интегральный показатель процесса ожидания (задержки) входных пакетов в узле ТКС

$$J(T) = \int_{t_0}^{t_T} l(t)\,dt. \tag{11}$$

Геометрически этот показатель определяет площадь под кривой $l(t)$ на интервале $[t_0, t)$. Так как это площадь образована прямоугольниками с высотой 1 и длиной Δ_k, то справедливы следующие соотношения

$$J(T) = \sum_{k=1}^{n(T)} \Delta_k = \sum_{k=1}^{n(T)} (t'_k - t_k) = \sum_{k=1}^{n(T)} t'_k - \sum_{k=1}^{n(T)} t_k. \tag{12}$$

Таким образом, величина $J(T)$ зависит не от времён ожидания Δ_k, а от суммы моментов времени отправки пакетов по каналам связи ТКС.

Так как $l(t)$ — это общее число ожидающих отправки входных пакетов в момент времени передачи t, то среднее число входных пакетов в накопителе узла ТКС можно определить по формуле

$$l(T) = \frac{1}{T}\int_{t_0}^{t_T} l(t)\,dt. \tag{13}$$

Средним временем ожидания на интервале $[t_0, t)$ принято называть величину

$$\Delta(T) = \frac{1}{n(T)} \sum_{k=1}^{n(T)} \Delta_k. \tag{14}$$

Поскольку средняя скорость поступления пакетов в узел ТКС определяется формулой (9), то можно установить связь между средними величинами (13) и (14) в виде

$$l(T) = \lambda(T)\Delta(T). \tag{15}$$

Выражение (9) совпадает с известной формулой Литта [1, 3], полученной для стратегии (дисциплины) обслуживания пакетов в порядке их поступления на конечном интервале времени $[t_0, t)$.

6. Проблемы управления потоками данных в ТКС

Для адресной и своевременной передачи информации в ТКС необходимо разработать модели на алгоритмы управления коммуникационным оборудованием и дискретными потоками данных.

Одной из важнейших проблем управления ТКС является маршрутизация потоков данных, т. е. определение пути на графе ТКС, связывающего некоторый начальный узел ТКС с конечным узлом. Обычно эта проблема ставится как оптимизационная задача, в результате решения которой определяются оптимальные (например, по длине или по времени передачи) маршруты.

Другая проблема заключается в управлении потоками данных в ТКС по определённому маршруту «из конца в конец», т. е. от начального узла маршрута к его конечному узлу. При этом необходимо учитывать ограниченные сетевые ресурсы ТКС, связанные с ограничениями на пропускную способность каналов связи, объём памяти накопителей узлов и т.п. Вследствие этого приходится регулировать темп (скорость) передачи данных между всеми узлами маршрута передачи сообщений.

Важной проблемой управления ТКС является анализ влияния сетевого трафика (нагрузки), который может случайным или непредсказуемым образом изменяться в широких пределах, на качество работы ТКС и представляемые ей услуги.

Анализ трафика ТКС необходим для учёта в процессе управления потоками данных реально складывающегося сетевого трафика. Дело в том, что резкое возрастание сетевой нагрузки может временно исчерпать возможности сетевых ресурсов (каналы связи и накопители в узлах ТКС) и вызвать значительные сетевые перегрузки («скрученность» потоков).

Таким образом, возникает необходимость в синтезе алгоритмов робастного или адаптивного управления потоками данных, обеспечивающих работоспособность ТКС в условиях неопределённости стохастически или непредсказуемо изменяющегося трафика.

Значительное увеличение трафика ξ приводит к резкому возрастанию задержки (времени обслуживания) Δ. При этом производительность ТКС, измеряемая общим числом P пакетов, доставляемых по назначению (т.е. адресату ТКС) в единицу времени может сильно снижаться.

При увеличении сетевой нагрузки ξ сверх допустимого предела ξ^* происходит значительное увеличение задержки пакетов и резкое падение производительности ТКС. Это связано с блокировкой ограниченных сетевых ресурсов (ограничения пропускной способности каналов связи и объёма памяти накопителей узлов) и возможностью возникновения тупиковой ситуации (при $\xi = \xi^*$), когда все накопители узлов ТКС переполнены. В этом случае поступление пакетов данных прекращается, и производительность ТКС падает до нуля ($P = 0$), т. е. ТКС перестаёт функционировать.

ТКС как сложный распределённый объект управления можно разбить на множество управляемых коммуникационных объектов: каналы связи, модемы, сегменты локальных ТКС, мосты, концентраторы, коммутаторы, маршрутизаторы, мультиплексоры и т. п. Управление этими объектами и потоками информации в ТКС должно обеспечивать адаптацию к изменяющемуся гетерогенному трафику и высокую производительность и качество услуг, предоставляемых ТКС.

Заключение

Предлагаемые в данной работе принципы обработки пакетов данных в ТК представляют важное значение для оптимизации трафика в глобальных информационных и телекоммуникационных сетей, что имеет большое прикладное и социальное значение. В частности, эти результаты могут быть полезны для организации адаптивного мультиагентного (массового) обслуживания компьютерных и GRID-сетей различного масштаба и назначения или для создания мировой сети Internet нового поколения.

Литература

1. *Уолрендж Дж.* Телекоммуникационные и компьютерные сети: Вводный курс. М.: Постмаркс. 2001. 480 с.
2. *Дилип Н.* Стандарты и протоколы Internet. М.: Русская редакция, 1999.
3. *Уолрендж Дж.* Введение в теорию сетей массового обслуживания. М.: Мир, 1993.

МУЛЬТИАГЕНТНЫЕ И НЕЙРОСЕТЕВЫЕ ТЕХНОЛОГИИ ИНТЕЛЛЕКТУАЛЬНОГО УПРАВЛЕНИЯ ПОТОКАМИ ДАННЫХ В GRID-СЕТЯХ

А. В. Тимофеев

профессор математико-механического факультета СПбУ,
заведующий лабораторией информационных технологий
в управлении и робототехнике СПИИРАН;
tav@iias.spb.su

Санкт-Петербургский государственный университет,
Санкт-Петербургский институт информатики и автоматизации РАН

Аннотация: Рассматриваются задачи и методы интеллектуального сетевого и нейросетевого управления информационными потоками в мультиагентных телекоммуникационных системах и GRID-сетях. Обсуждаются принципы построения и оптимизации сетевых и нейросетевых агентов, обеспечивающих интеллектуальный анализ информационных потоков и адаптивное сетевое управление в условиях неопределённости в нечёткой среде.

Введение

Развитие информационных и телекоммуникационных технологий на современном этапе требует разработки теоретических основ проектирования и имитационного моделирования глобальных телекоммуникационных систем (ТКС) нового поколения и средств их интеграции с GRID-сетями, в которых аккумулируются распределенные информационные и вычислительные ресурсы. Такие интегрированные ТКС и GRID-сети предоставляют своим многочисленным пользователям как внешним агентам высококачественные услуги для их массового удаленного доступа и эффективного использования распределенных информационных и вычислительных ресурсов с помощью IP-протоколов и других средств управляемой связи и передачи информационных потоков.

1. Задачи сетевого управление и интеллектуального анализа информационных потоков

Совершенствование глобальных ТКС и GRID-сетей связано прежде всего с развитием методологии автоматизации, адаптации и интеллектуализации систем сетевого управления информационными потоками на базе динамических моделей ТКС как сложных объектов управления с переменной структурой и нечёткими или частично неопределёнными условиями эксплуатации. Важное значение имеет также разработка инновационных методов оптими-

зации процессов маршрутизации информационных потоков и принципов адаптивного и интеллектуального управления трафиком с использованием мульти-агентных технологий и протоколов нового поколения (IPv6 и др.). На этом новом пути возможен как учет реальной динамики ТКС, т. е. фактического состояния или изменения структуры (топологии) и параметров (весов каналов связи) ТКС в реальном времени в нечёткой среде, так и адаптация к различным факторам неопределенности на основе мониторинга и функциональной диагностики мультиагентных ТКС [1–2].

2. Архитектура мультиагентных сетей

Архитектуру глобальных ТКС или распределённых GRID-сетей можно представить как сложную систему, состоящую из четырёх основных (базисных) подсистем [2]:

1) Распределенная система связи (РСС).
2) Сетевая система управления (ССУ).
3) Распределенная информационная система (РИС).
4) Распределенная транспортная система (РТС).

Эти подсистемы взаимосвязаны и предназначены для управляемой передачи внешним агентам-пользователям ТКС по их запросам информационных и вычислительных ресурсов, распределенных в GRID-средах.

Центральную роль в эффективной организации и управляемой передаче информационных потоков играют ССУ. ССУ нового поколения должны быть адаптивными и интеллектуальными [2], т. е. обладать способностями к адаптации (автоматической самонастройке) по отношению к изменяющемуся количеству пользователей с учетом их запросов «по интересам» и персональных требований к качеству предоставляемых услуг, к изменяющимся структуре (топологии) ТКС и параметрам (весам) узлов и каналов связи в нечёткой или частично неопределённой среде.

Интеллектуальная ССУ основана на обучении и самообучении новым функциям и правилам функционирования ТКС и GRID-сетей, а также на самоорганизации структуры и функций ССУ в зависимости от возможных изменений в РТС и РИС и предотвращении отказов и сетевых конфликтов.

3. Методы маршрутизации и адаптации сетевого управления информационными потоками данных

Традиционная статическая постановка задачи планирования и оптимизации маршрутов передачи данных основывается на предположении, что структура (число узлов, топология связей) и параметры (стоимость каналов связи) ТКС известны и неизменны. При этом в роли внешнего агента-пользователя ТКС обычно выступает один клиент, формирующий запрос к одному из узловых компьютеров ТКС [1].

Динамическая постановка задачи маршрутизации исходит из того, что структура или параметры ТКС могут изменяться с течением времени, но при этом остаются известными. В этом случае сетевая информация о ТКС обновляется, что приводит к автоматическому изменению (пересчету) оптимальных маршрутов передачи потоков данных в системе сетевого управления.

Адаптивная постановка задачи предполагает, что маршрутизация и сетевое управление осуществляются в условиях неопределённости в нечёткой среде, когда топология и параметры каналов связи ТКС, а также трафик и число пользователей могут непредсказуемо изменяться в широких пределах. При этом доступная информация о ТКС обычно имеет локальный характер. Мониторинг и обновление сетевой информации по каналам обратной связи позволяют адаптивно скорректировать маршруты и алгоритмы сетевого управления потоками данных [1–7].

Первые системы управления в глобальных ТКС выполняли в автоматизированном режиме (т. е. с участием человека) важные, но сравнительно простые функции. К ним, прежде всего, относятся:

- управление телекоммуникационным оборудованием;
- контроль изменяющегося трафика ТКС.

Примером программного обеспечения такой системы управления может служить программный продукт Sun Net Manager, разработанный компанией Sun Soft в 1989 году [1].

Главной целью сетевого управления ТКС является быстрая доставка (транспортировка) необходимой информации пользователям ТКС с высоким качеством предоставляемых услуг. С этой точки зрения общую задачу управления ТКС можно разделить на три взаимосвязанные подзадачи:

- управление потоками данных для всех видов гетерогенного трафика;
- управление телекоммуникационным оборудованием;
- административное управление производительностью и конфигурацией ТКС.

В глобальных ТКС (например, в Internet) управление передачей и распределением (маршрутизацией) потоков данных должно осуществляться не по жесткой программе, а «в россыпь» по непредсказуемо изменяющимся запросам пользователей или узловых компонентов ТКС.

Традиционный подход к управлению ТКС зачастую не обеспечивает интерактивность в реальном времени (например, речевой диалог без задержек). Другими недостатками традиционного подхода являются неадаптивность (по отношению к изменяющемуся трафику) управления потоками информации, невозможность автоматического предотвращения сетевых конфликтов, распознавания неисправностей и реконфигурирования ТКС без участия человека (администратора сети) и т. п.

4. Особенности мультиагентного управления потоками данных в ТКС

При проектировании и эксплуатации современных телекоммуникационных сетей (ТКС) важную роль играют системы управления потоками передаваемых данных. Однако теория управления ТКС (в отличие, например, от теории автоматического управления движением самолетов, роботов и других подвижных объектов) развита слабо. Поэтому возникает необходимость в постановке, формализации и решении задач управления, обработки и передачи информации в ТКС с учётом их особенностей и неопределённости условий эксплуатации [2].

Специфика ТКС как объекта управления заключается в распределённом характере компонент ТКС и управляемых потоков разнородных (гетерогенных) данных, передаваемых через узлы ТКС по различным маршрутам и каналам связи. Вследствие этого система управления ТКС также должна быть распределённой и иметь многоуровневую иерархическую структуру [2–5].

На каждом уровне этой системы возникают специфические цели и задачи управления. Однако многие из этих целей не формализованы, а задачи не решены (в том смысле, что для этих задач отсутствуют теоретически обоснованные модели и алгоритмы управления). Поэтому в настоящее время значительное внимание уделяется постановке и решению задач управления потоками данных в условиях неопределённости и сетевых конфликтов.

Неопределённость условий эксплуатации ТКС проявляется в непредсказуемом характере изменения и гетерогенности сетевого трафика. Кроме того, неопределённым является и число пользователей ТКС, которое может значительно изменяться в течение суток. В ТКС могут возникнуть сбои или отказы отдельных компонент, а также разного рода сетевые конфликты. Поэтому необходимы адаптация к трафику, мониторинг и диагностика состояний ТКС и классификация и разрешение сетевых конфликтов.

Указанные особенности ТКС требуют исследования и разработки адаптивного подхода к решению задач управления потоками данных в ТКС на базе современных интеллектуальных и мультиагентных технологий. Предлагаемые методы адаптивного и интеллектуального управления обеспечивают адаптацию к непредсказуемо изменяющемуся трафику, адаптивную маршрутизацию потоков данных, мультиагентную обработку информации, функциональную диагностику ТКС, распознавание и разрешение сетевых конфликтов [2–7].

Широкий класс сложных распределённых ТКС может быть представлен как мультиагентная система (МАС). При этом в роли агентов ТКС как МАС выступают либо пользователи (внешние агенты)ТКС, либо узловые компьютеры или локальные ТКС (внутренние агенты). Характерными чертами этих внутренних агентов ТКС является наличие локальных баз данных и знаний и телекоммуникационных каналов связи для обмена информацией между

агентами в процессе совместного использования распределенных информационных ресурсов и обработки информации.

Главная особенность мультиагентной обработки информации и управления заключается в том, что сначала сложная задача декомпозируется (фрагментируется) на ряд локальных задач, решение которых распределяется (распараллеливается) между агентами, а затем результаты решения этих локальных задач агрегируются (интегрируются) и реализуются с помощью телекоммуникационных ресурсов.

Работа МАС обработки информации пи интеллектуального анализа поддерживается ТКС, реализующей сетевые технологии передачи данных между агентами. Коммуникация между удаленными агентами осуществляется на уровне их локальных баз данных и знаний путём управляемого обмена сообщениями в процессе решения локальных (индивидуальных) или общих (глобальных) задач.

Значительный теоретический и практический интерес представляют две стратегии мультиагентной обработки и передачи информации:

- с координатором (когда один из агентов отвечает за координацию поведения всех остальных агентов);
- и без координатора (когда все агенты «равноправны» и не подчиняются ведущему агенту — координатору).

При мультиагентном управлении потоками данных в ТКС возникает необходимость в разработке методов автоматического предотвращения или разрешения сетевых конфликтов, которые могут возникать между агентами ТКС. Важное значение имеют мультиагентные модели и алгоритмы обработки информации (репликация кода, фрагментация данных, адаптивная маршрутизация и т. п.).

При проектировании систем управления потоками данных в ТКС важную роль играет надёжность используемого оборудования. Надёжность глобальной ТКС тем ниже, чем больше узловых компьютеров входит в состав ТКС. Это объясняется тем, что с увеличением числа узлов ТКС возрастает вероятность выхода из строя одного или нескольких компьютеров. Поэтому возникает необходимость в адаптивном управлении и мультиагентной обработке информации в ТКС, гарантирующих решение задач при непредсказуемом изменении трафика, сбое или отказе одного или нескольких узловых компьютеров ТКС [4,7].

5. Многопотоковая маршрутизация в мультиагентных ТКС

Многопотоковая маршрутизация является задачей, близкой к много-адресной маршрутизации. Отличием является то, что в многопотоковой маршрутизации один узел-источник и один узел-получатель, а передаваемые потоки данных дублируются и передаются по разным маршрутам.

Многопотоковая динамическая маршрутизация полезна в тех случаях, когда планирование маршрутов производится с учетом возможности выхода из строя какого-то участка ТКС, через который пройдет поток данных. При этом случае каждый узел ТКС будет «знать» один или несколько запасных маршрутов, по которым можно будет передавать поток данных. Использование «запасных» маршрутов зависит от стратегии управления потоками данных в ТКС.

Сформулируем задачу K-потоковой маршрутизации при $K \geq 2$ [4, 7].

Рассмотрим графовую модель ТКС

$$G = G(A, R, W), \tag{1}$$

где A — множество вершин графа G, соответствующих узлам ТКС, R — множество однонаправленных дуг G, соответствующих каналам связи, а W — множество весов ребер (каналов связи), соответствующих некоторым числовым характеристикам (параметрам) каналов связи, определяющих их «стоимость». Стоимость маршрута определяется как суммарная стоимость составляющих его рёбер (каналов связи).

Будем говорить, что два маршрута пересекаются, если они имеют общие вершины, отличные от начальной и конечной.

Пусть выбраны два узла $s \in A$ — узел-источник, а $d \in A$ — узел-получатель. Необходимо проложить K непересекающихся маршрутов из s в d, таких, что их суммарная стоимость наименьшая из всех возможных вариантов.

Введем необходимые обозначения. Пусть все несамопересекающиеся маршруты из s в d проиндексированы. Обозначим i-й маршрут из s в d как i_{sd}, а его стоимость — как $w(i_{sd})$. Если i-й и j-й маршруты не пересекаются, то будем обозначать этот факт следующим образом: $i_{sd} \cap j_{sd}$.

Определим все возможные множества из K непересекающихся маршрутов из s в d в виде

$$I = \{i_{sd}\} : \begin{cases} \forall i_{sd}, j_{sd} \in I : i_{sd} \cap j_{sd}, \\ |I| = K. \end{cases} \tag{2}$$

Зададим над этими множествами функционал Q вида

$$Q(I) = \sum_{i_{sd} \in I} w(i_{sd}). \tag{3}$$

Тогда задача K-потоковой оптимальной маршрутизации сводится к оптимизации функционала (3), т. е.

$$Q(I) = \sum_{i_{sd} \in I} w(i_{sd}) \to \min \tag{4}$$

при ограничениях (2).

6. Классификация GRID-сетей и базовые фрактальные архитектуры

По своему назначению GRID-сети принято делить на вычислительные сети (computational GRID) и сети, ориентированные на хранение больших массивов информации (data GRID).

GRID-системы целесообразно использовать прежде всего для решения следующих научных и прикладных задач высокой сложности:

5. математическое и имитационное моделирование сложных систем и процессов;
6. совместная визуализация и динамическая анимация больших массивов и быстрых потоков научных данных;
7. распределенная обработка в целях системного анализа и кластеризации данных;
8. комплексирование научного инструментария с удаленными компьютерами и архивами данных и знаний;
9. распределенная обработка баз данных и знаний с использованием современных технологий «Data Mining» и «Knowledge Discovery».

Как показывает практика, GRID-системы наиболее эффективно используются в следующих областях:

1. распределенные высокопроизводительные вычисления;
2. решение сверхсложных задач, требующих максимальных процессорных ресурсов, памяти и быстродействия;
3. «многопоточная» обработка информации, позволяющая организовать эффективное использование основных информационно-вычислительных ресурсов с утилизацией временно простаивающих компьютеров;
4. проведение крупных разовых оптимизационных расчетов;
5. вычисления с привлечением больших объемов распределенных данных (например, в метеорологии, генетике, астрономии, физике высоких энергий);
6. распределенные коллективные вычисления, координирующие одновременное решение нескольких взаимодействующих вычислительных задач разных пользователей.

Специфика GRID-систем и обслуживающих их ТКС нового поколения определяются прежде всего архитектурой (структурой и функциями) самой вычислительной сетевой среды, на которой они строятся. В общем случае эта среда представляет собой глобальную динамическую сеть с изменяющейся (в течение некоторого периода времени) архитектурой, т. е. с изменяющимся количеством активных вычислительных узлов (узловых компьютеров) и каналов связи между ними [4, 7].

Изменения архитектуры GRID-среды или связанной с ней глобальной ТКС происходят как за счет нештатного отказа, планового выключения

компьютеров или подключения новых вычислительных мощностей. Кроме того, часто возникают измерения качественных характеристик и количественных параметров интегрированной сети (изменение вычислительной нагрузки узлов или пропускной способности каналов связи и т. п.) [5–7].

Архитектура интегрированной вычислительной, информационной и телекоммуникационной среды определяются характеристиками сетевых топологических структур, образующих саму эту среду в виде архитектуры (топологической структуры) узловых компьютеров и каналов связи между ними. Важно отметить, что именно топологические структуры (архитектуры) компонентов GRID-систем и ТКС имеют решающее значение как при системном анализе и моделировании существующих GRID-систем и ТКС, так и особенно при проектировании, имитационном моделировании и оптимизации архитектур вновь создаваемых GRID-систем и ТКС нового поколения.

Исходя из перечисленных выше требований, предъявляемых к компьютерным и телекоммуникационным сетям как основе для построения интегрированных GRID-систем и современных ТКС, наиболее существенными являются следующие показатели (критерии качества) [5,6,7]:

I. надёжность;
II. стоимость;
III. пропускная способность.

Эти показатели (критерии) отражают наиболее важные аспекты функционирования GRID-систем и ТКС любого масштаба. Они соответственно обеспечивают:

- Устойчивую работоспособность GRID-систем и ТКС в целом.
- Экономический фактор, актуальный для заказчика или проектировщика GRID-системы и ТКС с точки зрения стоимости.
- Качество и быстрота обслуживания внешних агентов-абонентов существующей или вновь создаваемой GRID-системы и ТКС.

Под базовыми топологиями (архитектурами) GRID-систем и ТКС будем понимать следующие самоподобные топологические структуры (фракталы):

1. Полная ячеистая топология.
2. Кольцевая топология.
3. Топология «Звезда».
4. Линейная топология, а также смешанная топологическая структура (мультифрактал).
5. Смешанная топология (включающая в себя в различных сочетаниях базовые топологии 1–4).

При имитационном моделировании и прогнозировании GRID-сетей и связанных с ними ТКС нового поколения важную роль играет оценка основных показателей I—III базовых топологических структур 1–5 (фрактальных и мультифрактальных архитектур), а также их многокритериальная оптимизация. Эта задача рассмотрена и решена в работах [6–7].

7. Сетевые и нейросетевые агенты в GRID-системах

Основные функции обработки информации, самоорганизации и управления информационными потоками по запросам внешних агентов-пользователей глобальной ТКС распределяются между внутренними агентами ТКС и GRID-сетей, роль которых выполняют сетевые или нейросетевые агенты. Архитектура этих внутренних сетевых агентов аналогична архитектуре ТКС. В этом проявляется фрактальность сетевых и нейросетевых агентов по отношению к ТКС и ее подсетям как автономным системам [6, 7].

Каждый внутренний сетевой или нейросетевой агент имеет собственную локальную БД и локальную БЗ, а также средства связи с другими агентами через РСС для обмена информацией в процессе совместного принятия решений, самоорганизации по «интересам» и автоматического формирования сетевого управления РТС, обеспечивающего адресную доставку информационных и вычислительных ресурсов по запросам внешних агентов-пользователей глобальной ТКС или GRID-сети [5–7].

Нейросетевые агенты предназначены прежде всего для параллельной передачи и обработки сложных мультимедийных сигналов и образов (2D- или 3D-изображения и т.п.). В результате обучения по множеству прецедентов из обучающей БД осуществляется настройка архитектуры (топологии сетевых нейронов) и параметров (синаптических весов) нейронных агентов на решаемую задачу [3–5]. В последнее время разработаны модели нейросетевых агентов для адаптивной маршрутизации (агенты-маршрутизаторы) и автоматической классификации WEB-сайтов на естественном (русском) языке (агенты-классификаторы) [5–7]. Программная реализация и имитационное моделирование этих агентов свидетельствует об их эффективности и преимуществах по отношению к традиционным информационным технологиям [7].

Заключение

В настоящей работе на основе системного анализа эволюции и особенностей динамики глобальных ТКС с изменяющейся структурой (сетевой топологией) и варьируемыми параметрами (весами каналов связи) обосновывается необходимость разработки теории адаптивного высококачественного мультиагентного обслуживания глобальных ТКС нового поколения и связанных с ними GRID-сетей как хранилища распределенных информационных и вычислительных ресурсов. Эта новая теория должна прийти на смену традиционной теории массового обслуживания, вероятностные предположения (гипотезы) которой, как правило, не выполняются на практике.

Отмечая достоинства и недостатки классических моделей и методов фиксированной (статической) маршрутизации потоков данных, авторы предлагают принципиально новые подходы, отличающиеся возможностью учета нечётких, нестационарных и неопределенных факторов. Эти факто-

ры фактически игнорируются (или, в лучшем случае, учитываются лишь «в среднем») традиционной теорией массового обслуживания ТКС, основанной на вероятностных предположениях (гипотезах). Однако их учет и адаптивная компенсация неопределенности при сетевом управлении потоками данных как в современных ТКС, так и в глобальных ТКС новых поколений, связанных с GRID-сетями, являются принципиально важными и абсолютно необходимыми.

Предлагаемые в данной работе модели и методы динамической, адаптивной, нейросетевой и мультиагентной (многоадресной и многопотоковой) маршрутизации информационных потоков для глобальных ТКС нового поколения представляются первым важным шагом в направлении создания теории адаптивного мультиагентного (массового) обслуживания глобальных информационных и телекоммуникационных сетей, имеющей большое прикладное и социальное значение. В частности, эти результаты могут быть полезны для организации адаптивного мультиагентного (массового) обслуживания GRID-инфраструктур различного масштаба и назначения или для создания мировой сети Internet нового поколения.

Работа выполнена при поддержке грантов РФФИ № 09-08-00767-а, РФФИ – ГФЕН Китая № 10-08-91159-а, Программы № 14 Президиума РАН и издательского гранта РФФИ № 12-08-07022-д.

Литература

1. *Столлингс В.* Современные компьютерные сети. СПб.: Питер, 2003. 783 с.
2. Тимофеев А. В. Проблемы и методы адаптивного управления потоками данных в телекоммуникационных системах // Информатизация и связь. 2003. № 1, 2. С. 68–73.
3. *Тимофеев А. В., Сырцев А. В.* Мультиагентная и нейросетевая маршрутизация потоков данных в телекоммуникационных сетях // Труды 10-й международной конференции «Knowledge — Dialogue — Solution» (16–26 июня, 2003, Варна). 2003. С. 187–190.
4. *Тимофеев А. В., Сырцев А. В.* Модели и методы маршрутизации потоков данных в телекоммуникационных системах с изменяющейся динамикой. М.: Новые технологии, 2005.
5. *Timofeev A. V.* Multi-Agent Information Processing and Adaptive Control in Global Telecommunication and Computer Networks // Information Theories and Their Applications. 2003. No. 10. P. 54–60.
6. *Тимофеев А. В.* Фрактальное моделирование и многокритериальная оптимизация компьютерных сетей // Information Science and Compiting. 2011. Vol. 5. No. 3, P. 237–244.
7. *Тимофеев А. В.* Адаптивное управление и интеллектуальный анализ информационных потоков в компьютерных сетях. М.: Наука, 2012.

СПЕКТРАЛЬНЫЕ, НЕЙРОСЕТЕВЫЕ И ДИОФАНТОВЫЕ РЕГУЛЯТОРЫ СИСТЕМ УПРАВЛЕНИЯ ПРОГРАММНЫМ ДВИЖЕНИЕМ

А. В. Тимофеев

профессор математико-механического факультета СПбУ,
заведующий лабораторией информационных технологий
в управлении и робототехнике СПИИРАН;
tav@iias.spb.su

Санкт-Петербургский государственный университет,
Санкт-Петербургский институт информатики и автоматизации РАН

Аннотация: Рассматриваются методы синтеза и анализа спектральных (стабилизирующих и модальных) регуляторов систем управления программным движением с обратимой (на подпространстве) динамикой. Значительное внимание уделяется исследованию и расчёту параметров декомпозирующих, диофантовых и нейросетевых регуляторов с помощью диаграмм Вышнеградского и их обобщений.

Введение

Современная теория автоматического управления нелинейными объектами тесно связана с информационными, нейросетевыми и когнитивными технологиями. Эта связь особенно важна в таких прикладных областях, как робототехника, мехатроника и аэрокосмические системы [1–7]. При этом информационные и когнитивные технологии полезны как на этапе автоматизированного проектирования (синтез, анализ, имитационное моделирование) систем автоматического управления, так и на этапе их программно-аппаратной реализации на современной микропроцессорной базе (цифровые сигнальные процессоры, многоядерные процессоры, нейропроцессоры и т. п.).

Эффективность управления сложными многомерными объектами (роботы, беспилотные летательные и подводные аппараты, планетоходы и т.д.) оценивается по многим показателям качества. К ним прежде всего относятся высокая точность и быстродействие, асимптотические и техническая устойчивость, робастность и адаптивность в нечётких и неопределённых условиях эксплуатации.

В настоящее время теория автоматического (в том числе оптимального) управления линейными динамическими объектами разработана почти исчерпывающим образом. В рамках этой теории синтез регуляторов по заданному расположению корней характеристического полинома замкнутой системы, т. е. по ее спектру, позволяет обеспечить не только асимптотическую устойчивость, но и заданные показатели качества переходных процессов (ПП).

Идея спектрального анализа и синтеза была предложена в 1876 г. И. А. Вышнеградским применительно к задаче о центробежном регуляторе паровой машины [1]. В дальнейшем эта идея была развита в теории модальных регуляторов многомерных линейных и линеаризованных систем.

В работах [2–6] были предложены спектральные методы синтеза, оптимизации и анализа нелинейных многомерных регуляторов обратимых динамических систем (ОДС), а также критерии глобальной управляемости, обратимости, декомпозируемости и стабилизируемости ОДС. В [2–6] были получены также аналитический вид и расчетные формулы для синтеза нелинейных спектральных регуляторов.

В настоящей работе рассматривается новый класс диофантовых и нейросетевых регуляторов и приводятся соотношения, связывающие их целочисленные параметры с целочисленными корнями характеристического полинома уравнения ПП. При этом существенно используются информационные технологиями и когнитивная графика, основанные на диаграммах Вышнеградского и их модификациях.

1. Архитектура систем управления программным движением

Во многих случаях глобальную задачу автоматического управления многомерными линейными и нелинейными объектами можно декомпозировать на две взаимосвязанные локальные задачи [2–7]:

1. задача построения и оптимизации программных движений (ПД);
2. задача синтеза, анализа и оптимизации регуляторов переходных процессов (ПП).

Методы решения первой локальной задачи позволяют построить в аналитическом виде ПД, обеспечивающее достижение цели движения с учётом конструктивных и динамических ограничений. Это ПД может быть оптимизировано при заданном показателе (функционале) качества. Некоторые методы и алгоритмы построения и оптимизации ПД описаны в работах [2–6]. В ряде случаев (например, в мобильной робототехнике) они связаны с методами навигации и маршрутизации [7–9].

Принципы решения второй локальной задачи направлены на синтез (в аналитическом виде), системный анализ и оптимизацию регуляторов ПП. Важную роль при этом играют математическое и имитационное моделирование поведения замкнутой системы автоматического управления. Эти принципы могут быть реализованы с помощью методов и алгоритмов, предложенных в работах [2–6], а также на основе диаграмм Вышнеградского и их модификаций.

Математическое и программное обеспечение решения первой и второй локальной задачи в сочетании с современными информационными технологиями и когнитивной графикой являются основой для автоматизированно-

го проектирования и мультимикропроцессорной реализации современных и перспективных систем управления программным движением.

Архитектура таких систем включается в себя две подсистемы, а именно [2–7]:

- программатор движений;
- регулятор переходных процессов.

Программатор движений представляет собой аппаратно-программный комплекс, обеспечивающий построение и оптимизацию ПД.

Регулятор ПП также реализуется в виде аппаратно-программного комплекса (например, на базе PLD, DSP или нейропроцессоров), обеспечивающего осуществление ПД с заданными показателями качества [2–7].

2. Динамика, обратимость и управляемость

Динамика широкого класса механических и мехатронных систем, роботов, космических и подводных аппаратов описывается векторным дифференциальным уравнением вида

$$A(y, y, ..., y^{(r-1)}, \xi)\, y^{(r)} + b(y, y', ..., y^{(r-1)}, \xi) + \pi = u, \;\; t \geq t_0, \tag{1}$$

где y — m-мерный управляемых координат, $y^{(r)}$ — его r-я производная по времени t, $\mathbf{u}$ — m-мерный вектор управлений, ξ — p-мерный вектор варьируемых параметров, $\mathbf{A}(\cdot)$, $\mathbf{b}(\cdot)$ — заданные матричная и векторная функции размерности $\mathbf{m} \times \mathbf{m}$ и $\mathbf{m}$, удовлетворяющие условиям существования и единственности решения уравнения (1) при заданных $\mathbf{u}$, ξ, начальных данных

$$y(t_0) = x_1(t_0), ..., y^{(r-1)}(t_0) = x_{r-1}(t_0) \tag{2}$$

и измеряемых или неконтролируемых внешних возмущениях $\pi = \pi(t)$.

Критерием обратимости систем вида (1) является невырожденность матрицы $\mathbf{A}(\cdot)$ при всех допустимых значениях аргументов. Этим свойством обладают многие объекты управления (в частности, механические и электромеханические системы, мехатронные системы и роботы) [2, 3]).

Введём вектор состояний канонического вида

$$x = |x_i|_{i=1}^{r}, \dot{x}_i = x_{i+1}, x_1 = y, n = rm. \tag{3}$$

Тогда систему (1), (2) можно записать в форме Коши

$$\dot{x} = F(x, u, \xi, \pi), \;\; x(t_0) = x, \;\; t \geq t_0. \tag{4}$$

Эта система имеет порядок $\mathbf{n} = rm$, причём

$$F(x, u, \xi, \pi) = |x_1, x_2, ..., w|^T, \tag{5}$$

где

$$w = A^{-1}(x, \xi)(u - b(x, \xi) - \pi).$$

В работах [2–4] показано, что система (4), (5) разрешима относительно **u** в виде (1), т. е. является ОДС на некотором подпространстве.

Представим систему (4), (5) в канонической форме

$$\dot{x} = Px + Qw,\ \ x(t_0) = x_0,\ \ t \geq t_0,$$
$$P = \begin{bmatrix} 0_m^{n-m} & I_{n-m} \\ 0_m^m & 0_{n-m}^m \end{bmatrix},\ \ Q = \begin{bmatrix} 0_m^{n-m} \\ I_m \end{bmatrix}. \qquad (6)$$

Здесь P и Q — блочные матрицы, $0_\alpha^\beta = I_\alpha$ – нулевая и единичная матрицы размерности $\alpha \times \beta$ и $\alpha \times \alpha$

Система (6) глобально управляема по w, так как для нее справедлив критерий Калмана

$$rank\,(Q, PQ, ..., P^{r-1}Q) = n. \qquad (7)$$

Следовательно, существует такой канонический регулятор w = *w(t)*, что движение замкнутой им системы (6) удовлетворяет граничным условиям

$$x(t_0) = x_0,\ \ x(t_T) = x_T,\ \ T = t_T - t_0 < \infty. \qquad (8)$$

Поскольку *w* связано с *u* взаимно однозначным преобразованием (5), то из (7) следует управляемость для исходной ОДС (1).

3. Синтез программных и стабилизирующих регуляторов

Обозначим через $x = x_p(t)$ допустимое решение (4), (5) при $\pi = 0$, удовлетворяющее граничным условиям (8), Назовём его программным движением (ПД).

Подставляя ПД в (1) при $\pi = 0$ получим в аналитическом виде программный регулятор

$$u(t) = A(x_p, \xi)\, x_{1,p}^{(r)} + b(x_p, \xi) \qquad (9)$$

и регулятор с обратной связью по вектору состояний

$$u(t, x) = A(x, \xi)\, x_{1,p}^{(r)} + b(x, \xi) \qquad (10)$$

Регуляторы (9) и (10) обеспечивают осуществление ПД в замкнутой системе в силу единственности решения (1) при начальном условии $x_p(t_0) = x_0$ и известном (или измеряемом) законе изменения параметров $\xi = \xi(t)$ и $\pi = 0$. Поэтому нестационарная нелинейная ОДС (1), замкнутая (10) является программно управляемой.

Программный регулятор (9) является решением обратной задачи динамики (1), впервые поставленной и решённой И. Ньютоном для случая $r = 2$ в задаче о динамике планет , а регулятор с обратной связью (10) синтезирован по принципу программного регулирования старшей производной (ускорени-

ем) уравнения движения (1) для случая $r = 2$, предложенному Г. С. Поспеловым в задаче о самонастраивающемся автопилоте.

Управляемость ОДС гарантирует существование стабилизирующих регуляторов, обеспечивающих асимптотическую устойчивость ПД. Для синтеза таких регуляторов воспользуемся принципом скоростного управления ПД, предложенным в работах [2–6]. Согласно этому принципу сначала сконструируем эталонное дифференциальное управление переходных процессов (ПП) вида

$$e^{(r)} = \phi(e, \dot{e}, ..., e^{(r-1)}, t), \quad e^{(i)}(t_0) = e_0^{(i)}, \tag{11}$$

где $e = y - y_p = x_1 - x_{1,p} = e_1$, $e^{(i)}(t)$ — i-я производная e по t, $i = 0, 1, \ldots, r-1$, $\varphi(\cdot)$ — m-вектор-функция, удовлетворяющая условиям асимптотической устойчивости тривиального решения $e(t) = 0$. Обозначив $E = x - x_p$, получим из (1.3), (2.1) эталонное уравнение ПП

$$\dot{E} = \Phi(E, t), \; E(t_0) = \left| e_0^{(i)} \right|_{i=0}^{r-1},$$

$$\Phi(E, t) = [e, e, ..., \phi]^T, \; E(t_0) = \left| e_0^{(i)} \right|_{i=0}^{r-1}, \tag{12}$$

причём $\Phi(0, t) = 0$ и существуют такие числа $c > 0$ и $\gamma > 0$, что для любого решения (2) справедлива экспоненциальная оценка

$$\|E(t)\| \le c \exp(-\gamma(t - t_0)) \|E(t_0)\|, \; t \ge t_0. \tag{13}$$

На втором этапе синтезируем нелинейный стабилизирующий регулятор из условия

$$\dot{x}_p + \Phi(E, t) = F(E + x_p, u, \xi). \tag{14}$$

Используя свойство обратимости, получим искомый нелинейный стабилизирующий регулятор в аналитическом виде

$$u = A(E + x_p, \xi)(x_{1,p}^{(r)} + \Phi(E, t)) + b(E + x_p, \xi). \tag{15}$$

Подставляя (15) в (1), убеждаемся, что ПП в замкнутой системе удовлетворяют эталонному уравнению (12) при $\pi = 0$. Следовательно, ПД будет асимптотически устойчивым.

На практике важен случай, когда

$$\phi = -\Gamma_{r-1} e^{(r-1)} - ... - \Gamma_1 \dot{e} - \Gamma_0 e, \tag{16}$$

где $\Gamma_i = \Gamma_i(t)$, $i = 0, 1, ..., r-1$ — варьируемые $m \times m$ матричные параметры, определяющие характер ПП. Тогда регулятор (15), (16) будет стабилизирующим, если блочная матрица Фробениуса размерности $p x p$ вида

$$\Gamma = \begin{vmatrix} 0 & I & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ \Gamma_0 & \Gamma_1 & \cdots & \Gamma_{r-1} \end{vmatrix}, \quad \operatorname{Re} \lambda_j(\Gamma) < 0, \quad j = 1, ..., n. \tag{17}$$

гурвицева. Здесь λj — собственные числа матрицы Г, определяющие спектр замкнутой ОДС. В этом случае ПП удовлетворяют при $\pi = 0$ экспоненциальной оценке (13), где $\gamma = \max\limits_{j} \lambda_j(\Gamma)$.

4. Декомпозирующие регуляторы и диаграммы Вышнеградского

Многомерным ОДС присущи нелинейные перекрестные связи, интенсивность которых зависит от текущего состояния. Будем называть регулятор декомпозирующим [2–5], если уравнения ПП в замкнутой ОДС (1) распадаются на систему независимых скалярных уравнений по всем управляемым координатам.

Критерием декомпозируемости регулятора (15) при $\pi = 0$ является диагональность матричных коэффициентов усиления, т. е.

$$\Gamma_i(t) = \operatorname{diag}(\gamma_i(t))_{i=1}^{m}, \ i = 0, 1, ..., r-1. \tag{18}$$

В этом случае уравнение ПП в замкнутой ОДС имеет вид

$$e_j^{(r)} = -\sum_{i=0}^{r-1} \gamma_{ij} e_j^{(i)}, \ j = 1, ..., m.$$

Следовательно, декомпозирующий регулятор (15) полностью компенсирует (развязывает) перекрестные связи и исключает их динамическое взаимовлияние в процессе стабилизации ПД. При этом значительно облегчается расчет параметров регулятора (15) по сданным показателям качества ПП.

Общий вид и показатели качества ПП в замкнутой ОДС (1), (15), (16) целиком определяются спектром матрицы Г.

Впервые на эту связь обратил внимание И. А. Вышнеградский [1]. Он предложил метод анализа качества регуляторов одномерных линейных систем порядка $n = 3$ в пространстве параметров $v_1 v_2$ нормированного полинома

$$p(q) = q^3 + v_2 q^2 + v_1 q + 1 = \prod_{i=1}^{3} (q - q_i), \tag{19}$$

построив для этого диаграмму, представляющую собой исчерпывающую картину неустойчивости, устойчивости и качества ПП.

Декомпозирующие регуляторы (15)–(17) обобщают этот метод на широкий класс многомерных линейных и нелинейных ОДС порядка $n = 3m$. Поэтому будем называть их обобщёнными регуляторами Вышнеградского.

Достоинствами диаграмм Вышнеградского являются наглядность, информативность и простота их использования для синтеза стабилизирующих и модальных результатов с наперёд заданными показателями качества и системного анализа ПП в замкнутых ОДС. Кроме того, они удобны для автома-

тизации проектирования линейных и нелинейных регуляторов, так как могут быть представлены средствами компьютерной «когнитивной графики» и дополнены «экспертными правилами» синтеза, оптимизации и анализа. Это позволило создать экспертную систему проектирования и реализации высококачественных регуляторов нелинейных ОДС порядка $n \geq 3$т [3, 6].

Расчет параметров стабилизирующего регулятора осуществляется наведением курсора в область над гиперболой $v_1 v_2 > 1$, $v_1 > 0$, $v_2 > 0$.

Задание апериодического, монотонного или колебательного ПП (по терминологии И. А. Вышнеградского) производится наведением курсора в областях D_a, $D_{\mathbf{m}}$ или D_k, которым соответствуют различные варианты расположения корней полинома (19). Для обеспечения наибольшей «степени устойчивости», связанной с робастностью ОДС, достаточно навести курсор в точку $v_1 = v_2 = 3$ на пересечении границ областей D_a и $D_{\mathbf{m}}$, что соответствует корням $q_j = -1$ или $i = 1, 2, 3$.

Априорный и сравнительный анализ обобщённых нелинейных регуляторов Вышнеградского сводится к подстановке их параметров в формулы

$$\gamma_1 = v_1 \sqrt[3]{\gamma_0}, \ \gamma_2 = v_2 \sqrt[3]{\gamma_0}, \ i = 1, ..., m, \tag{20}$$

и определению области диаграммы, куда попала точка с координатами v_j, v_2.

Этот метод применим также для синтеза и анализа нелинейного аналога m-мерного ПИД-регулятора вида [3].

$$u = A(x, \xi) \left[\ddot{x}_{1p} - \Gamma_1 \dot{e} - \Gamma_0 e - \Gamma_* \int_{t_0}^{t} e(s)\, ds) \right] + b(x, \xi), \ x = |y_i|_{i=1}^{2}. \tag{21}$$

Регуляторы, обеспечивающие заданное расположение корней замкнутой системы, принято называть модальными. К ним относятся линейные регуляторы Вышнеградского и нелинейные регуляторы общего вида (15–17).

Однако представляется более обоснованным называть их спектральными, поскольку параметры этих регуляторов однозначно определяются по спектру матрицы Г. Если этот спектр удовлетворяет условиям (17), (18), то регуляторы являются стабилизирующими и декомпозирующими, а тип (характер) ПП целиком определяется расположением корней $\lambda_i(\Gamma)$, $j = 1, ..., n$ слева от мнимой оси.

Для упрощения расчетов и удобства цифровой реализации спектральных регуляторов (15), (17) потребуем, чтобы их параметры и соответствующие корни характеристических полиномов уравнения ПП (16) были целыми числами. Такие регуляторы будем называть диофантовыми.

Для вычисления корней по параметрам регулятора в случае $r = 3$ и $r = 4$ можно воспользоваться формулами Кардано и Феррари. Расположение целочисленных корней определяется неравенствами

$$1 \leq \lambda_i < 1 + \omega, \ i = 1, ..., r,$$

$$\omega = \max(|\gamma_0|, ..., |\gamma_{r-1}|). \quad (22)$$

Параметры диофантовых регуляторов связаны неравенствами Гюа

$$q_{ij} \equiv \gamma_{ij}^2(\gamma_{i-1,j}, \gamma_{i+1,j})^{-1} > 1, \quad i = 1, ..., r-1. \quad (23)$$

Величины q_{ij} особенно удобны для задания характера и показателей качества ПП, так как они безразмерны и инвариантны по отношению к умножению полиномов на произвольное число.

Критерий апериодичности ПП при использовании диофантова регулятора (15) задаётся неравенствами

$$q_{ij} \geq 4, \quad i = 1, ..., r-1, \quad j-1, ..., m. \quad (24)$$

Необходимое и достаточное условие асимптотической устойчивости ПД для декомпозирующих диофантовых регуляторов общего вида определяется неравенствами

$$\gamma_{ij} \geq 1, \quad \sum_{k=1}^{2} s_{k+1,j} < 1, k = 1, ..., r-4,$$
$$s_{kj} = \gamma_{k-1,j}, \gamma_{k+2,j}(\gamma_{k,j}, \gamma_{k+1,j})^{-1}. \quad (25)$$

Заключение

Исследование робастности и синтез алгоритмов адаптации для синтезированных регуляторов с заданным спектром основываются на принципах адаптивного управления ПД. Методы синтеза и анализа робастных нейросетевых и адаптивных систем управления ПД рассмотрены в работах [2–7]. Развитие этих принципов и методов на мультиагентные и нейросетевые системы управления рассмотрены в работах [7–10].

Работа выполнена при частичной поддержке грантов РФФИ № 09-08-00767-а, РФФИ № 12-08-01167-а и издательского гранта РФФИ № 12-08-07022-д.

Литература

1. *Вышнеградский И. А., Максвелл Д. К., Стодола А.* Теория автоматического регулирования. М.: Изд-во АН СССР, 1949. 386 с.
2. *Тимофеев А. В.* Построение адаптивных систем управления программным движением. Л.: Энергия, 1980. 88 с.
3. *Тимофеев А. В.* Управляемость, робастность и инвариантность обратимых систем с нелинейной динамикой // Доклады АН. 1998. Т. 359. № 2. С. 171–174.
4. *Тимофеев А. В.* Управление роботами. Л.: Изд. ЛГУ, 1985. 217 с.
5. *Тимофеев А. В.* Мультиагентное и интеллектуальное управление сложными робототехническими системами // Юбилейный сборник «Теоретические основы и прикладные задачи интеллектуальных информационных технологий», по-

священный 275-летию РАН и 20-летию СПИИ РАН. СПб.: СПИИРАН, 1999. С. 71–81.

6. *Тимофеев А. В.* Методы высококачественного управления, интеллектуализации и функциональной диагностики автоматических систем // Мехатроника, автоматизация, управление. 2003. № 5.
7. *Тимофеев А. В., Юсупов Р. М.* Принципы построения интегрированных систем мультиагентной навигации и интеллектуального управления мехатронными роботами // Information Technologies & Knowledge. Vol. 5. No. 3. 2011. Pp. 237–244.
8. *Тимофеев А. В*. Мульти-агентное управление коллективом роботов // Проблемы информатизации. 2000. Вып. 1. С. 70–77.
9. *Тимофеев А. В., Кай З., Хе Х*. Навигация и управление движением роботов в неизвестной среде // Гироскопия и навигация. № 3. 2004. №2 (45). С. 13–24.
10. *Тимофеев А. В.* Мультиагентные робототехнические системы и нейросетевые технологии // Известия ЮФУ. Технические науки. Перспективные системы и задачи управления. 2010. № 3. С. 20–23.

АГЕНТООРИЕНТИРОВАННЫЙ ПОДХОД К МОДЕЛИРОВАНИЮ КОМПЛЕКСА «ИНФОРМАЦИОННАЯ СИСТЕМА — ПЕРСОНАЛ — ЗЛОУМЫШЛЕННИК» В ЗАДАЧАХ ОЦЕНКИ ЗАЩИЩЕННОСТИ ОТ СОЦИОИНЖЕНЕРНЫХ АТАК[1]

А. А. Азаров

аспирант, м.н.с.**; artur-azarov@yandex.ru*

Т. В. Тулупьева

доц., с.н.с.**; alexander.tulupyev@gmail.com*

А. Л. Тулупьев

проф., зав. лаб.**; alexander.tulupyev@gmail.com*

*** Кафедра информатики Санкт-Петербургского государственного университета**

**** Лаборатория теоретических и междисциплинарных проблем информатики СПИИРАН**

Аннотация: Анализ защищенности информационных систем от социо-инженерных атак, направленных на пользователей таких систем, является одним из перспективных направлений развития информационных технологий.

Для оценки защищенности персонала таких систем необходимо построить агентоориентированную многокомпонентную модель комплекса «информационная система — персонал — злоумышленник», состоящую из более низкоуровневых моделей персонала системы, злоумышленника, аппаратно-технической составляющей системы, а также набора документов с учетом атрибутов их критичности.

Введение

Комплексные корпоративные информационные системы в настоящее время получают все большее распространение в современном мире. Разработка, поддержка и защита подобных систем занимает значительное количество времени и ресурсов, кроме того только высококвалифицированные специалисты могут заниматься подобными системами. Информация, хранимая в таких информационных системах, имеет огромную ценность для компаний-вла-

[1] Поддержано: грант РФФИ, проект № 10-01-00640-а «Интеллектуальные модели и методы анализа защищенности информационных систем от социо-инженерных атак (деревья атак)»; грант СПбГУ, проект № 6.38.72.2011 «Моделирование комплексов „информационная система — персонал“ для агрегированной оценки их готовности к отражению социоинженерных атак».

дельцев систем, поэтому значительные усилия затрачиваются на построение системы защиты таких систем от различных угроз безопасности. Новейшие системы безопасности могут защитить информационные системы от большинства кибер-атак, а также серьезно осложнить жизнь злоумышленникам, пытающимся добраться до конфиденциальной информации, но в то же время большинство таких систем не защищено от внутренних угроз, исходящих от пользователей таких систем. Таким образом, злоумышленники, обладающие навыками социо-инженеров, с легкостью могут повлиять на пользователей информационных систем с целью получения именно той информации, которая им нужна.

Необходимо научиться защищать информацию от такого типа атак. Кроме того необходимо научиться оценивать уровень защищенности персонала таких информационных систем от социоинженерных атак. Исходя из того, что подобные атаки используют персонал информационных систем в качестве основного пути развития атаки, необходимо научиться прогнозировать уязвимости пользователей информационных систем. Подобные показатели могут служить индикаторами того, насколько пользователь подвержен тому или иному действию злоумышленника. Также можно предположить, какие действия будут совершены пользователем в ответ на действия злоумышленника и с какой вероятностью. Подобную информационную систему, в которой действуют пользователи и злоумышленники, целесообразно представить в виде мультиагентной системы (более того — осознанно или неосознанно такое представление в том или ином масштабе будет опираться на мультиагентный подход и использовать его особенности). Данное представление возможно из-за того, что между пользователями есть различные связи (дружественные, корпоративные, коммуникативные и другие), злоумышленник влияет на пользователей через определенные связи (вербально или не вербально), а также и пользователи и злоумышленник отличаются некоторой самостоятельностью, автономностью в реализации своих действий [1, 6, 7]. Целью данной работы является выявление особенностей пользователя для последующего построения его модели — пользователя-агента, — которая может стать составляющей частью соотвествующей мультиагентной системы.

Выявление элементарных уязвимостей

Для того чтобы непосредственно переходить к построению пользователя-агента необходимо выявить элементарные уязвимости пользователя, на которые злоумышленник будет влиять элементарными атакующими воздействиями. Из данных элементарных уязвимостей можно будет построить профиль уязвимостей пользователя, представляющий собой вектор, состоящий из конкретных значений каждой уязвимости. Теоретико-психологическое объяснение классификации уязвимостей не дает ответа на вопрос, как их выявлять,

измерять, оценивать и прогнозировать. Так как уязвимость — латентная переменная; без сбора эмпирических данных невозможно определить степень проявления уязвимости через степень выраженности психологических особенностей пользователя. Для этого было проведено исследование [2–5], с помощью которого был построен фрагмент профиля уязвимостей пользователя. В него вошло 5 уязвимостей (приводятся условные названия), оценка степени проявления которых в численном виде была сделан экспертами:

1. Техническая неосмотрительность.
2. Слабый пароль.
3. Техническая халатность и установка на получение личной выгоды.
4. Техническая неопытность.
5. Техническая безграмотность.

Оценка ущерба, наносимого социо-инженерными атаками на пользователей информационных систем

После успешного построения фрагмента профиля уязвимостей, было установлено, что для каждой уязвимости, из включенных в профиль уязвимостей, существует несколько вариантов воздействий злоумышленника, а также несколько вариантов ответных действий пользователя на них. Кроме того, было установлено, что некоторые действия злоумышленника могут вызвать активацию сразу нескольких уязвимостей. Так например, действие злоумышленника «Отправка письма с «полезным» для пользователя приложением» активирует уязвимости техническая неосмотрительность, техническая неопытность и техническая безграмотность. Степень проявления каждой из этих уязвимостей переводится в вероятность исходя из значения соответствующей уязвимости. Таким образом, если действие злоумышленника приводит к активации нескольких уязвимостей, которые ведут к одному и тому же ответному действию с вероятностями p_1, p_2, p_3, то можно найти агрегированную вероятность совершения такого действия. Она будет находиться по формуле $p = 1 - (1 - p_1)(1 - p_2)(1 - p_3)$.

Кроме того, очевидным образом из всех действий злоумышленника выделяется два итоговых состояния, в которые может прийти злоумышленник. Он может получить идентификационные данные пользователя, или же он получит доступ к компьютеру пользователя. Соответствующие общие вероятности также получаются с помощью формулы того же типа, что и формула, представленная выше.

Таким образом, на основе психологических характеристик пользователя построен психологический профиль пользователя, который позволяет построить вероятностную оценку защищенности информационной системы против действий злоумышленника.

Заключение

Таким образом, на основе психологических характеристик пользователя можно построить психологический профиль пользователя, который, в свою очередь, позволяет построить профиль уязвимостей, лежащей в основе имитации деятельности (реакций на социоинженерные атаки) пользователей-агентов — как элементов мультиагентной системы «информационная система — персонал — злоумышленник».

Литература

1. *Тулупьев А. Л., Азаров А. А., Тулупьева Т. В., Пащенко А. Е., Степашкин М. В.* Социально-психологические факторы, влияющие на степень уязвимости пользователей автоматизированных информационных систем с точки зрения социоинженерных атак // Труды СПИИРАН. 2010. Вып. 1 (12). С. 200–214.
2. *Тулупьев А. Л., Азаров А. А., Пащенко А. Е.* Информационная модель пользователя, находящегося под угрозой социоинженерной атаки // Труды СПИИРАН. 2010. Вып. 2 (13). С. 143–155.
3. *Тулупьев А. Л., Азаров А. А., Тулупьева Т. В., Пащенко А. Е.* Визуальный инструментарий для построения информационных моделей комплекса «информационная система — персонал», использующихся в имитации социоинженерных атак // Труды СПИИРАН. 2010. Вып. 4 (15). С. 231–245.
4. *Азаров А. А.* Анализ защищенности информационных систем от социоинженерных атак рекомпенсационного типа в отношении пользователей // VII Санкт-Петербургская межрегиональная конференция «Информационная безопасность регионов России (ИБРР-2011)» (Санкт-Петербург, 26–28 октября 2011 г.) Материалы конференции. СПб.: СПОИСУ, 2011. С. 160.
5. *Азаров А. А., Тулупьева Т. В., Пащенко А. Е., Тулупьев А. Л.* Развитие методов и моделей анализа защищенности информационных систем от социоинженерных атак на основе применения реляционно-алгебраических представлений и алгоритмов // VII Санкт-Петербургская межрегиональная конференция «Информационная безопасность регионов России (ИБРР-2011)» (Санкт-Петербург, 26–28 октября 2011 г.) Материалы конференции. СПб.: СПОИСУ, 2011. С. 160–161.
6. *Тулупьева Т. В., Тулупьев А. Л., Азаров А. А., Пащенко А. Е.* Психологическая защита как фактор уязвимости пользователя в контексте социоинженерных атак // Труды СПИИРАН. 2011. Вып. 18. С. 74–92.
7. *Ванюшичева О. Ю., Тулупьева Т. В., Пащенко А. Е., Тулупьев А. Л., Азаров А. А.* Количественные измерения поведенческих проявлений уязвимостей пользователя, ассоциированных с социоинженерными атаками. // Труды СПИИРАН. 2011. Вып. 19. С. 34–47.

ВЫЧИСЛЕНИЕ МОЩНОСТИ МНОЖЕСТВА МИНИМАЛЬНЫХ ГРАФОВ СМЕЖНОСТИ[1]

А. А. Фильченков

аспирант, м.н.с.**; aaafil@mail.ru*

А. Л. Тулупьев

д.ф.-м. н., доц., профессор зав. лаб.**; alt@iias.spbu.su*

*** Кафедра информатики Санкт-Петербургского государственного университета**

**** Лаборатория теоретических и междисциплинарных проблем информатики СПИИРАН**

Аннотация: Алгебраические байесовские сети являются вероятностной графической моделью, их глобальная структура представляется в виде графа смежности. Строение графа смежности (наличие или отсутствие в нем циклов) оказывает существенное влияние на возможность логико-вероятностного вывода в соответствующей алгебраической байесовской сети, а также на сложность алгоритмов такого вывода. В работе выражена мощность множества минимальных по числу ребер графов смежности через численные характеристики особых подграфов максимального графа смежности.

Введение

Алгебраические байесовские сети (АБС) относятся к классу вероятностных графических моделей [1, 2], к которым также относятся байесовские сети доверия и скрытые марковские модели [7, 8, 10]; в отличие от двух последних АБС позволяют моделировать сеть с вероятностной информацией в узлах, которая может содержать как скалярные, так и интервальные оценки вероятности истинности.

Помимо того, что АБС позволяются оперировать интервальными оценками вероятности, еще одним их отличием является определенная свобода для построения графа на заданных вершинах. К такому графу предъявляется ряд требований, и граф, который отвечает им всем, называется графом смежности [2, 4]. Однако обычно число таких графов достаточно велико, поэтому

[1] Работа выполнена при финансовой поддержке РФФИ, проект № 09-01-00861-а «Методология построения интеллектуальных систем поддержки принятия решений на основе баз фрагментов знаний с вероятностной неопределенностью», проект № 12-01-00945-а «Развитие теории алгебраических байесовских сетей и родственных им логико-вероятностных графических моделей систем знаний с неопределенностью».

одной из задач глобального обучения АБС является построение такого графа смежности, который был бы лучшим по определенным характеристикам. Минимальные по числу ребер графы смежности представляют в этом смысле особый интерес, поскольку обладают рядом важных свойств, существенных для работы АБС [3]. Однако даже число таких графов может быть достаточно велико. С точки зрения задачи глобального обучения, желательно оценить мощность такого множества до его построения (или построения отдельных его элементов) для того, чтобы спрогнозировать время построения этого множества или рассчитать вероятности при построении случайного минимального графа смежности.

В докладе рассмотрено конструктивное описание элементов такого множества, а также вычислена его мощность.

Основные определения

Алфавитом будем называть множество атомарных пропозициональных формул $A = \{x_1, \ldots, x_n\}$. *Словом* тогда будет называться некое подмножество алфавита.

Первичная структура или множество главных конъюнктов максимальных ФЗ (МФЗ), вошедших в АБС, — такое множество слов $V^* = \{V_i \subseteq A\}_{i=1}^{i=m}$, что никакое слово полностью не содержит никакое другое слово: $\forall i \neq j (V_i \not\subset V_j)$ и $(V_j \not\subset V_i)$.

Граф МФЗ — ненаправленный граф, вершины которого соответствуют элементам множества главных конъюнктов МФЗ, вошедших в АБС, а ребра возможны только между теми вершинами, что пересечение соответствующих МФЗ непусто.

Вес $W(V_i)$ вершины $V_i \in V(G)$ — множество атомов алфавита, вошедших в V_i: $W(V) = \{x_i | x_i \in V\}$. *Вес* $W(\{V_i, V_j\})$ ребра $\{V_i, V_j\} \in E(G)$, графа G определяется как пересечение весов тех вершин, которые соединены этим ребром[1]: $W(\{V_i, V_j\}) = W(V_i) \cap W(V_j)$.

Вес $W(H)$, *подграфа* $H \subseteq G$ — наибольшее по включению слово, которое входит в веса всех его вершин: $W(H) = \bigcap_{V \in H} W(V)$.

Под *магистральным путем* $B: V_b \dashrightarrow V_e$ от вершины V_b до вершины V_e, пересечение весов которых непусто, будем понимать такой путь от вершины V_b до вершины V_e, что вес любой принадлежащей ему вершины содержит пересечение весов начальной и конечной вершин. $B: V_b \dashrightarrow V_e = P: V_b \dashrightarrow\!\!\!\!\!\rightarrow V_e|$ такой, что $\forall V_i \in BW(V_b) \bigcap W(V_e) \subset W(V_i)$.

---

1 Под V и E будем понимать функции, которые возвращают множество вершин и множество ребер графа соответственно.

Граф *магистрально связен*, если между каждой парой несовпадающих вершин, веса которых содержат общие элементы, существует магистральный путь.

Благодаря введенным понятиям *граф смежности* определяется как магистрально связный граф МФЗ. При этом магистрально связный граф не обязательно связен (например, граф из двух вершин *ab* и *cd*, в котором нет ребер, тем не менее, является магистрально связным).

Минимальный граф смежности — граф смежности, число ребер в котором минимально. Множество минимальных графов будем обозначать как **MJG**.

Максимальный граф смежности $G_{\max}$ — наибольший по числу ребер граф смежности. Так как в графе МФЗ возможны не все ребра, а только те, которые соединяют вершины, пересечение весов которых непусто, то максимальный граф смежности вовсе необязательно совпадает с полным подграфом. В [4] было доказано, что для заданного множества вершин существует и при этом единственный максимальный граф смежности.

Сужение $G{\downarrow}U$ ненаправленного графа G на слово U — это ненаправленный граф, в который входят только те вершины и ребра исходного графа G, веса которых содержат или равны U:

$$G{\downarrow}U = \{V_i \mid V_i \in V(G),\ U \subseteq W(V_i)\},\ \{E_i \mid E_i \in E(G),\ U \subseteq W(E_i)\}.$$

Значимое слово графа G — слово, являющееся весом какого-либо ребра графа *G*. *Значимое сужение* $G{\downarrow}U$ — сужение графа G на значимое для графа G слово U. *Значимая клика веса U* — значимое сужение $G_{\max}{\downarrow}U$. Доказано [4], что значимая клика является кликой (т.е. полным подграфом). Будем обозначать множество всех клик графа $G_{\max}$ через Clique**.**

Родительский граф над множеством значимых клик — направленный граф, вершинами которого являются значимые клики из множества Clique. Ребро из вершины P в вершину Q проведено, если значимая клика P содержит значимую клику Q, и не существует значимой клики R, такой, что P содержит R и R содержит Q.

$$G_{\text{Clique}} = \text{Clique}, E_{\text{Clique}},$$

где

$$P, Q \in \text{Clique},\ (P,Q) \in E \Leftrightarrow Q \subset P \text{ и } \nexists R \in \text{Clique} : Q \subset R \subset P.$$

Будем называть значимые клики, в которые входят ребра из значимой клики C в родительском графе *сыновьями C*, а C — их *родителем*.

Представление минимальных графов смежности

Сильное сужение $G \twoheaddownarrow U$ — значимое сужение $G \downarrow U$, из которого удалили все ребра веса U:

$$G \twoheaddownarrow U = \{V_i \mid V_i \in V(G),\ U \subseteq W(V_i)\},\ \{E_i \mid E_i \in E(G),\ U \subset W(E_i)\}.$$

Сильное сужение графа $G_{\max} \twoheaddownarrow U$ представляет собой компоненты связности, на которые разбивается сужение $G_{\max} \downarrow U$ удалением ребер веса U. Такие компоненты будем называть *владениями*.

Сжатие σ_U компоненты связности $P_U^i \subseteq G \twoheaddownarrow U$ в вершину f_i — отображение на множестве подмножеств вершин, сопоставляющее множеству вершин P_U^i вершину f_i.

Сжатие σ_U множества ребер

$$E_{i,j}\,(G \downarrow U) \backslash (G \twoheaddownarrow U),$$

соединяющих владения P_U^i и P_U^j в ребро $e_{i,j}$ — отображение на множестве подмножеств ребер, сопоставляющее множеству ребер $E_{i,j}$ ребро $e_{i,j}$, соединяющее вершины $f_i = \sigma_U(P_U^i)$ и $f_j = \sigma_U(P_U^j)$ и имеющее кратность, равную $|E_{i,j}|$.

Сжатие σ_U графа смежности G в граф K_U — отображение на множестве графов, сопоставляющий графу G, являющему графом смежности, граф K_U, вершинами которого являются владения сильного сужения $G \twoheaddownarrow U$, а ребро между двумя вершинами f_1 и f_2 графа K_U существует, если существует ребро в графе G между вершинами, принадлежащими соответствующим f_1 и f_2 владениям P_U^1 и P_U^2. Кратность такого ребра (f_1, f_2) равна числу всех ребер, соединяющих вершины из P_U^1 и P_U^2.

Феод — f_i из определения сжатия — вершина, получившаяся сжатием какого-то владения.

Курия веса U — граф K_U из определения сжатия — ненаправленный граф с кратными ребрами, полученный сжатием значимого сужения $G \downarrow U$.

Оммаж H_U — курия K_U, являющаяся деревом, все ребра которой имеют кратность, равную единице. Известно [6], что любой минимальный граф смежности сжимается до оммаж.

Жила S_U — множество ребер графа смежности G, такое, что $E_u = \{\sigma_U(e) \mid e \in S_U\}$ является множеством ребер оммажа сжатия σ_U. В [6] было доказано, что в жилу S_U входят те и только те ребра минимального графа смежности M, вес которых равен U.

Пучок — граф, построенный на исходном наборе вершин, множество ребер которого равно объединению жил, выбранных по одной для каждого

значимого слова. В [6] было доказано, что каждый пучок задается однозначно набором жил, выбранных по одной для каждой клики.

Теорема 1 (о множестве минимальных графов смежности) [4, 6]. Множество минимальных графов смежности совпадает с множеством пучков.

Следствие 1 [4]. Множество минимальных графов смежности совпадает с множеством пучков, которое равно декартовому произведению множеств жил каждой значимой клики.

Следствие 2 [4]. Мощность множества графов смежности равна произведению мощностей множеств жил каждой значимой клики.

Стереоклики — значимые клики, сжимающиеся до более чем одного феода.

Следствие 3. Мощность множества минимальных графов смежности равна произведению мощностей множеств жил каждой стереоклики.

Мощность множества минимальных графов смежности

Объем феода F — $v(F)$ — равен числу вершин, входящих во владение, соответствующее феоду.

Объем ребра оммажа H — $v((F,G))$ — равен числу возможных ребер, проведенных между вершинами соответствующих концам-феодам этого ребра. Доказано [5], что $v\left((F,G)\right)=v(F)\cdot v(G)$.

Утверждение 1 [5]. Каждому оммажу H соответствует число жил, равное произведению объемов всех ребер оммажа:

$$\prod_{\substack{F,G\in V(H)\\(G,G)\in E(H)}} v(F,G).$$

Степень феода F — $d(F)$ — равна числу ребер, из него выходящих.

Утверждение 2 [5]. Каждому оммажу H соответствует число жил, равное произведению объемов всех феодов в степени, равной степени феода:

$$\prod_{F\in V(H)} \left(v(F)\right)^{d(F)}.$$

Коэффициент раздробленности значимой клики C — $\mathfrak{D}(C)$ — произведение объемов всех феодов значимой клики, помноженных на число ее вершин, возведенное в степень числа ее владений, уменьшенному на 2:

$$\mathfrak{D}(C)=\left(\prod_{i=1}^{n} v(F_i)\right)(|C|)^{n-2},$$

где C сжимается до n феодов $F_1,\ldots,F_n$.

Теорема 2 (о мощности множества минимальных графов смежности) [5]. Мощность множества минимальных графов смежности равна произведению коэффициентов раздробленности всех значимых клик:

$$|\mathbf{MJG}| = \prod_{C \in \mathrm{Clique}} \mathfrak{D}(C).$$

Схема доказательства. Следствие 2 (из теоремы 1) говорит о том, что мощность множества графов смежности равна произведению мощностей множеств жил каждой клики. Поэтому доказываем, что коэффициент раздробленности клики равен мощности множества ее жил.

За счет того, что число вершин в значимой клике равно сумме объемов всех ее феодов, коэффициент раздробленности переписывается так:

$$\mathfrak{D}(C) = \left(\prod_{i=1}^{n} v(F_i)\right)\left(\sum_{i=1}^{n} v(F_i)\right)^{n-2},$$

где F_i — феоды значимой клики C.

При раскрытии скобки получаются n^{n-2} слагаемых вида

$$\left(\prod_{i=1}^{n} v(F_i)\right) \cdot v\left(F_{a_1}\right) \cdot \ldots \cdot v\left(F_{a_{n-2}}\right),$$

где индексы $a_1, \ldots, a_{n-2}$ пробегают все значения от 1 до n. Каждое такое слагаемое мы можем сопоставить коду Прюфера [9], из чего можно вывести, что каждое такое слагаемое соответствует какому-либо оммажу, причем оно в точности равно числу жил, этому оммажу соответствующих. Отсюда следует, что сумма всех таких слагаемых равна числу всех жил для данной значимой клики, что доказывает. что коэффициент раздробленности равен мощности множества жил, что в свою очередь значит, что мощность множества минимальных графов смежности равна произведению коэффициентов раздробленности для каждой значимой клики.

Следствие 4. Мощность множества жил для каждой значимой клики равна коэффициенту раздробленности значимой клики.

Следствие 5. Мощность множества минимальных графов смежности равна произведению коэффициентов раздробленности стереоклик.

Заключение

Благодаря введенной системе терминов и привлечению кодов Прюфера, удалось явно выразить мощность множества минимальных графов смежности через число вершин значимых клик и число вершин их владений. Для этого использовалась теорема о множестве минимальных графов смеж-

ности, которая представляет указанное множество в виде декартового произведения жил всех значимых клик.

Это позволяет оценить мощность указанного множества еще до построения каких-либо графов смежности.

Литература

1. *Тулупьев А. Л., Николенко С. И., Сироткин А. В.* Байесовские сети: логико-вероятностный подход. СПб.: Наука, 2006. 607 с.
2. *Тулупьев А. Л., Сироткин А. В., Николенко С. И.* Байесовские сети доверия: логико-вероятностный вывод в ациклических направленных графах. СПб.: Изд-во С.-Петерб. ун-та, 2009, 400 с.
3. *Тулупьев А. Л., Фильченков А. А., Вальтман Н. А.* Алгебраические байесовские сети: задачи автоматического обучения // Информационно-измерительные и управляющие системы. 2011. №11. Т. 9. С. 57–61.
4. *Фильченков А. А., Тулупьев А. Л.* Структурный анализ систем минимальных графов смежности Труды СПИИРАН. 2009. Вып. 11. С. 104–127.
5. *Фильченков А. А., Тулупьев А. Л., Сироткин А. В.* Мощность множества минимальных графов смежности // Труды СПИИРАН. 2010. Вып. 4(15). С. 136–161.
6. *Фильченков А. А., Тулупьев А. Л., Сироткин А. В.* Особенности анализа вторичной структуры алгебраической байесовской сети // Труды СПИИРАН. 2010. Вып. 1(12). С. 97–118.
7. *Jensen F.* Bayesian Networks and Decision Graphs. NY.: Springer. 2001. 268 p.
8. *Pearl J.* Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, 1988. 552 p.
9. *Prüfer H.* Neuer Beweis eines Satzes über Permutationen // Arch. Math. Phys. 27, 742–744, 1918.
10. *Stengel M.* Introduction to Graphical Models, Hidden Markov Models and Bayesian Networks. Department of Information and Computer Sciences Toyohashi University of Technology Toyohashi, 441-8580. Japan, 2003. 46 p.

ОЦЕНКА ИНТЕНСИВНОСТИ ПОВЕДЕНИЯ ДЛЯ МОДЕЛИРОВАНИЯ ДЕЯТЕЛЬНОСТИ ИНДИВИДОВ В СОЦИАЛЬНЫХ СЕТЯХ

А. В. Суворова

м.н.с., асп.**; suvalv@mail.ru*

А. Е. Пащенко

н.с., м.н.с.**; aep@iias.spb.su*

А. Л. Тулупьев

зав. лаб., проф.[2]; alexander.tulupyev@gmail.com*

Т. В. Тулупьева

с.н.с., доц.**; tvt100a@mail.ru*

А. В. Лавренов

*студент**, vedrfiolnir@gmail.com*

*** Лаборатория теоретических и междисциплинарных проблем информатики СПИИРАН**

**** Кафедра информатики Санкт-Петербургского государственного университета**

Аннотация: Одной из важнейших среди задач моделирования деятельности в социальных сетях является моделирование рискованного поведения, путей распространения информации, а также передачи какого-то признака от одного агента к другому. Одним из таких факторов, удобным и интересным для изучения, а также построения связанных с ним математических моделей, является вероятность передачи особо опасных инфекций между индивидами.

Целесообразно построить мультиагентную систему, которая будет моделировать поведение индивидов, используя наборы характеристик, присущих индивидам в социальной сети.

Введение

Одной из важнейших среди задач моделирования деятельности в социальных сетях является моделирование рискованного поведения, путей распространения информации, а также передачи какого-то признака от одного агента к другому. Одним из таких факторов, удобным и интересным для изучения, а также построения связанных с ним математических моделей, является вероятность передачи особо опасных инфекций между индивидами.

При известных параметрах рискованного поведения агента мы можем приступить к моделированию целого ряда процессов среди агентов и получить, например, сведения о скорости распространения инфекции, време-

ни, через которое произойдет передача какого-то признака от одного агента к другому, возможно, по сложному маршруту.

Кроме того, станет доступным ряд других выводов и оценок, крайне важных в медицине, и которые позволят лучше понимать и контролировать процесс распространения инфекций в социальных сетях.

Возможным источников получения информации о рискованном поведении агентов является подход, который развивается в СПИИРАН, и основывается на анализе данных в условиях дефицита информации [1–5]. Цель доклада — описание некоторых особенностей оценки величины параметра λ, характеризующего интенсивность участия респондента в поведении определенного вида; такие оценки используются в мультиагентных системах, моделирующих социальные сети, по которым могут распространяться инфекции. Оценка также используется в расчетах производных характеристик рискованного поведения: комулятивного риска, ожидаемого ущерба, эффективности превентивных мероприятий.

Анализ данных в условиях дефицита информации

Рассмотрим используемый подход к анализу данных в условиях дефицита информации [6]. Более подробно он описан в [1–5].

В результате интервью становятся известными сведения о нескольких (в рассматриваемом случае ¾ трех) последних эпизодах поведения. Серия эпизодов рассматривается как пуассоновский случайный процесс с основным уравнением [7]

$$P\left[N\left([t_0, t_0+T]\right)=k\right]=\frac{e^{-\lambda t}(\lambda T)^k}{k!},$$

где t_0 — начальный момент времени, k — число последовательных событий, которые вспомнил респондент, а T — тот период времени, за который эти эпизоды произошли, λ — интенсивность.

Применив метод максимального правдоподобия [8] к основному уравнению пуассоновского процесса при вышеуказанных данных, получим оценку интенсивности $\hat{\lambda}=\frac{k}{T}$.

При оценивании по методу максимального правдоподобия мы неявно предполагаем, что в день интервью также произошёл эпизод поведения (поскольку мы относимся к интервалу как к интервалу между эпизодами). Таким образом, полученная выше оценка является оценкой сверху для интенсивности. Обозначим в связи с этим

$$\lambda_{\max}=\frac{3}{T}. \qquad (1)$$

Коррекция подхода к учёту последнего интервала

Как было показано в предыдущем разделе, при использовании существующего подхода к анализу данных, полученных в результате интервью, делается необоснованное предположение о том, что в момент интервью также происходит эпизод поведения.

Введём дополнительные обозначения. Если x — случайная величина с оговоренным распределением, то через $p(x)$ обозначим значение функции плотности x. Через $\pi(x)$ обозначим частный случай — экспоненциальное распределение $\lambda e^{-\lambda x}$. Тогда для случайного интервала T, подчиняющегося экспоненциальному распределению с интенсивностью λ, $p(T) = \pi(T)$. Пусть τ — случайная величина, отвечающая прерыванию интервала T моментом интервью — имеет функцию плотности $p_T(\tau)$. Предположим (и это предположение оправдано, поскольку «жизненные планы» интервьюера и респондента независимы), что эти случайные величины являются независимыми. Тогда для случайной величины τ_0, соответствующей интервалу между последним эпизодом поведения и моментом интервью, функция плотности имеет вид

$$p(\tau_0) = \int_0^\infty p_T(\tau_0)\, p(T)\, dT.$$

Предполагая, что интервью может случиться в любой момент интервала $[0, T]$ с равной вероятностью, получим

$$p(\tau_0) = \int_{\tau_0}^\infty \frac{\lambda e^{-\lambda T}}{T} dT.$$

Применив метод максимального правдоподобия, получим следующее уравнение:

$$3 - \lambda(\tau_1 + \tau_2) - \frac{e^{-\lambda\tau_0}}{\left(\int_{\tau_0}^\infty \frac{e^{-\lambda u}}{u} du\right)} = 0. \qquad (2)$$

Далее будем ссылаться на уравнение (2) как на уравнение интенсивности. Следующий раздел будет посвящён исследованию этого уравнения.

Исследование уравнения интенсивности

Рассмотрим функцию

$$F(\lambda) = 3 - \lambda(\tau_1 + \tau_2) - \frac{e^{-\lambda\tau_0}}{\int_{\tau_0}^\infty \frac{e^{-\lambda u}}{u} du}. \qquad (3)$$

Таким образом, уравнение $F(\lambda) = 0$ имеет решения.

Теперь исследуем $F(\lambda)$ на монотонность. Для этого рассмотрим уравнение

$$\frac{dF}{d\lambda} = -(\tau_1 + \tau_2) + \frac{\tau_0 e^{-\lambda\tau_0}}{\int\limits_{\tau_0}^{\infty} \frac{e^{-\lambda u}}{u} du} - \frac{e^{-2\lambda\tau_0}}{\lambda\left(\int\limits_{\tau_0}^{\infty} \frac{e^{-\lambda u}}{u} du\right)^2} = 0.$$

И если это уравнение имеет решения, то можно написать, что

$$\frac{e^{-\lambda\tau_0}}{\int\limits_{\tau_0}^{\infty} \frac{e^{-\lambda u}}{u} du} = \frac{1}{2}\left(\lambda\tau_0 \pm \sqrt{\lambda^2\tau_0^2 - 4\lambda(\tau_1 + \tau_2)}\right).$$

Но правая часть равенства меньше, чем $\lambda\tau_0$, а левая — строго больше, что невозможно. Таким образом, производная функции $F(\lambda)$ не обращается в ноль, то есть функция F монотонна. Из этого можно заключить, что уравнение $F(\lambda) = 0$ имеет единственное решение.

Так как при близких к нулю значениях λ, значение $F(\lambda) > 0$, а $F(\lambda_{max}) < 0$, то из монотонности $F(\lambda)$ сразу следует, что корень уравнения $F(\lambda) = 0$ меньше λ_{max}, то есть, что искомое значение интенсивности действительно меньше максимального.

Теперь, пользуясь тем, что

$$\int\limits_{\tau_0}^{\infty} \frac{e^{-\lambda u}}{u} du = E_1(\lambda\tau_0) = -\gamma - \ln(\lambda\tau_0) + \sum_{k=1}^{\infty} \frac{(-1)^{k+1}(\lambda\tau_0)^k}{k \cdot k!} \qquad (5)$$

(где γ — это постоянная Эйлера, $\gamma \approx 0{,}5772156649$) и заменяя сумму ряда на частичную сумму из достаточно большого числа слагаемых, искомое значение интенсивности можно найти с помощью компьютерных приближений с любой точностью.

Для этой цели была разработана программа, которая по значениям $\tau_1 + \tau_2$ и τ_0 вычисляет значение интенсивности λ (с точностью более чем 10^{-10}), так же для сравнения приводит величину λ_{max} (см. (1)).

Таблица 1

Сравнение оценок интенсивности

$\tau_1 + \tau_2$	τ_0	λ	λ_{max}
100.0	20.0	0.02036110827008067	**0.02**5
80.0	1.0	0.03327070392353336	**0.03**7037037037037035
60.0	1.0	0.04384859066390613	**0.04**918032786885246
40.0	1.0	0.06447826030761894	**0.0**7317073170731707
20.0	3.0	0.10752191882608336	**0.1**3043478260869565
10.0	2.0	0.2036110827008067	**0.2**5
2.0	15.0	0.13142530495996352	**0.1**7647058823529413

В таблице 1 приведены значения λ и λ_{max} на ряде примеров. Полужирным шрифтом выделены совпадающие знаки после запятой для первоначальной и уточненной оценок. Таким образом, в приложениях, не требующих большой точности, можно использовать простую формулу вида (1). В общем случае исправленная оценка более точна.

Неточные естественно-языковые формулировки оценок длины временных интервалов

Как было отмечено ранее, ответы на вопросы об эпизодах поведения поступают на естественном языке, т.е. являются в значительной степени нечеткими и неполными. Отметим, что респонденты используют в своих высказываниях разные единицы измерения: часы, дни, недели, месяцы, полугода, года. Причем использованная единица измерения несет в себе информацию о точности измерения. Поясним это на примере двух высказываний: «семь дней назад» и «неделю назад». Когда респондент использует формулировку «семь дней назад», это свидетельствует о высокой «надежности» припоминания и его уверенности в том, что событие произошло ровно семь дней назад. Обобщая сказанное выше, можно сделать предположение, что если событие произошло за период более длинный, чем количество единиц измерения (например, семь дней), которых достаточно для построения высказывания, используя другую «следующую» языковую конструкцию (одну неделю назад), то можно говорить о повышенной степени точности, а также, вероятно, об экстраординарности произошедшего события, которая позволила запомнить его гораздо лучше и выделить из ряда других повседневных событий. Когда респондент использует формулировку «неделя назад», он априорно снижает точность высказывания. Неделя назад — это и шесть дней назад и восемь. При этом если речь идет о высказывании, например, «семь недель», мы опять имеем повышенную точность. Это опять может свидетельствовать об экстраординарности события, обычно в таких случаях употребляется конструкция «два месяца назад».

Можно ввести определение, что под «нормальной точностью» понимается точность единиц измерения, а также кратные ей единицы, если они не могут быть заменены такой языковой конструкцией, как например «семь дней» → «неделя», «четыре недели» → «месяц» и так далее. Под «повышенной точностью» будем понимать, высказывания, которые являются временными промежутками, которые могли быть заменены языковой конструкцией следующего порядка, например «двадцать три дня назад», «14 месяцев назад».

То есть для человеческого восприятия «непрерывного времени» уже выработаны внутренние механизмы перехода от одной точности высказывания (более высокой) к другой (более грубой) в зависимости от отдаленности события в прошлом. При этом ряд высказываний могут обладать более высокой точностью.

Для учета указанной неточности каждый ответ рассматривается не как точка на временной оси, а как интервал, длина которого зависит от единицы измерения. Значение каждого ответа рассматривается, таким образом, не как константа, а как случайная величина с заранее заданным распределением [1]. Введенная случайная величина за счет рандомизации [6] неопределенности ответа, обусловленной нечеткостью его формулировки, позволяет рассмотреть интенсивность как случайную величину и вычислить характеристики последней. Для каждого значения соответствующий интервал разбивается на m частей. Рассматриваются все возможные сочетания точек из интервалов, вычисляются их веса и рассчитываются значения интенсивности по описанному выше методу. Затем строится обобщенная взвешенная оценка [6].

Заключение

В данной работе обсуждается один из аспектов модели, описывающей социально-значимое поведение респондента по данным о его последних эпизодах. Описанный метод позволяет избежать неявного предположения о том, что в момент интервью происходит следующий эпизод. Другой подход, учитывающий особенности интервала между последним эпизодом и моментом интервью, рассматривает введение распределения особого вида (принадлежащего к классу простых бета-распределений) [9].

Рассмотренный в докладе метод, а также ряд его свойств позволит при моделировании социальных сетей при помощи мультиагентного подхода опираться на реальные обоснованные данные, полученные в результате исследований, а также понимать, как полученный данные могут меняться, от чего они зависят и в каких пределах их значения измеряются.

Литература

1. *Суворова А. В., Тулупьев А. Л., Пащенко А. Е., Тулупьева Т. В., Красносельских Т. В.* Анализ гранулярных данных и знаний в задачах исследования социально значимых видов поведения // Компьютерные инструменты в образовании. № 4. 2010. С. 30–38.
2. *Тулупьева Т. В., Пащенко А. Е., Тулупьев А. Л., Красносельских Т. В., Казакова О. С.* Модели ВИЧ-рискованного поведения в контексте психологической защиты и других адаптивных стилей. СПб.: Наука, 2008. 140 с.
3. *Пащенко А. Е., Тулупьев А. Л., Николенко С. И.* Моделирование заражения ВИЧ-инфекцией на основе данных о последних эпизодах рискованного поведения // Известия высших учебных заведений: Приборостроение. 2006. № 8. 33–34 с.
4. *Тулупьева Т. В., Тулупьев А. Л., Пащенко А. Е.* Оценка интенсивности поведения респондента в условиях информационного дефицита // Труды СПИИРАН. Вып. 7. СПб.: Наука, 2008. С. 239–254.
5. *Тулупьева Т. В., Пащенко А. Е., Тулупьев А. Л., Голянич В. М.* Модели ВИЧ-рискованного поведения в контексте психологической защиты и адаптации // Вестник СПбГУ. 2010. Серия 12. Вып. 1. С. 95–104.

6. *Хованов Н. В.* Анализ и синтез показателей при информационном дефиците. СПб.: Изд-во СПбГУ, 1996. 196 с.
7. *Розанов Ю. А.* Случайные процессы (краткий курс). М.: Главная редакция физико-математической литературы издательства «Наука», 1971. 288 с.
8. *Крамер Г.* Математические методы статистики. М.: Мир, 1975. 648 с.
9. *Зельтерман Д., Тулупьев А. Л., Суворова А. В., Пащенко А. Е., Мусина В. Ф., Тулупьева Т. В., Красносельских Т. В., Гро Л., Хаймер Р.* Обработка систематической ошибки, связанной с длиной временных интервалов между интервью и последним эпизодом в гамма-пауссоновской модели поведения // Труды СПИИРАН. 2011. Вып. 16. С. 160–185.

ИСПОЛЬЗОВАНИЕ НЕЙРОННЫХ СЕТЕЙ ДЛЯ ПРОГНОЗИРОВАНИЯ ФИНАНСОВЫХ ВРЕМЕННЫХ РЯДОВ

А. А. Красноперов

студент Математико-Механического факультета;
Alexey.Krasnoperov@gmail.com

Санкт-Петербургский государственный университет

Аннотация: Данная статья раскрывает возможности использования нейронных сетей для прогнозирования финансовых временных рядов, таких как цены на акции и производные финансовые инструменты на фондовом рынке.

Введение

Нейронные сети с самого момента возникновения были интересны в тех областях, где точное прогнозирование будущего может быть вознаграждено и где, одновременно, это прогнозирование возможно. Такой областью применения является фондовый рынок, действующий по определенным законам, а следовательно, потенциально прогнозируемый с допустимой точностью.

На момент формирования рынка ценных бумаг, самыми популярными и надежными были облигации, т. к. инвесторы часто не представляли в каком направлении движется рынок и, следовательно, не могли предсказать вырастет цена на акции в следующий момент времени или же наоборот опустится. Первым индикатором, сдвинувшим торговлю в сторону акций, явился индекс Доу Джонс, вобравший в себя цены на акции одиннадцати крупнейших американских компаний. С возникновением этого индекса для инвесторов открылся новый спекулятивный рынок, где от точности предсказания движения цены в будущем зависел доход инвестора.

С развитием рынка увеличивались и скорости торговли, а с переходом на электронные торги и с вовлечением в торговлю не только инстуциональных, но и частных инвесторов, скорость выросла в разы. Прогнозирование «вручную» динамично развивающегося рынка в краткосрочном периоде стало невозможным, требовались другие решения. Одним из них стали нейронные сети.

В настоящей статье рассматривается использование нейронных сетей при прогнозировании цен на акции и производные финансовые инструменты для высокочастотной торговли.

Проектирование нейронной сети

Для успешного прогнозирования цен на акции и производные финансовые инструменты для высокочастотной торговли, нейронная сеть должна удовлетворять следующим критериям:

- подстраиваться под динамично изменяющийся рынок;
- обладать достаточной скоростью предсказания и переобучения при использовании для высокочастотной торговли;
- иметь минимальный процент ошибок при угадывании тренда.

Исходя из требований, было принято решение использовать сеть с обратным распространением ошибки, а в качестве активационной функции использовать сигмоидальную функцию. Для наилучшего прогнозирования необходимо правильно подобрать следующие параметры: количество входных нейронов, а также количество скрытых слоев и нейронов в них.

Количество входных нейронов является редко изменяющимся параметром, поэтому в определении их наилучшего количества были использованы эвристические алгоритмы, выполняющиеся с низкой частотой. Количество скрытых слоев и нейронов в них, наоборот, могут изменяться с высокой частотой, в зависимости от состояния рынка в момент времени. Поэтому было принято решение производить динамическое создание новых нейронных сетей, наиболее подходящих под изменившиеся условия, и переключение прогнозирования на них в режиме реального времени. Для такой задачи не подходят эвристические алгоритмы ввиду их медленного выполнения, необходим быстрый алгоритм перестройки нейронной сети.

Таким алгоритмом явился каскадный метод. Применение алгоритма автоматического синтеза каскадных сетей с разнотипными нейронами позволяет получить более точные приближения к оцениваемому временному финансовому ряду в условиях неопределённости параметров и нелинейности функциональных зависимостей. При этом характеристические параметры предсказываемой цены находятся в пределах ограничений, заданных целевыми требованиями (с точки зрения уровня риска, ликвидности и ограничений по суммам). Определение управляющих параметров, удовлетворяющих целевым требованиям на основе синтеза каскадных сетей, по сути, является «эквивалентом» решения задачи системы уравнений с заданными начальными условиями, когда явный вид функций, входящих в систему уравнений, не определён. Таким образом, реализация алгоритма автоматического синтеза каскадных нейросетей с разнотипными нейронами обеспечивает инвесторов полезным инструментом предсказания цен на акции и производные финансовые инструменты в условиях неопределенности.

Создание новых нейронных сетей в разработанном инструменте происходит в режиме реального времени. Переключение на новую нейронную сеть происходит в том случае, если на проверочных данных она показывает себя лучше, чем используемая в настоящий момент нейронная сеть.

Ниже приведен пример предсказания цены на акции ОАО «Аэрофлот», с помощью разработанной нейронной сети на тиковых данных.

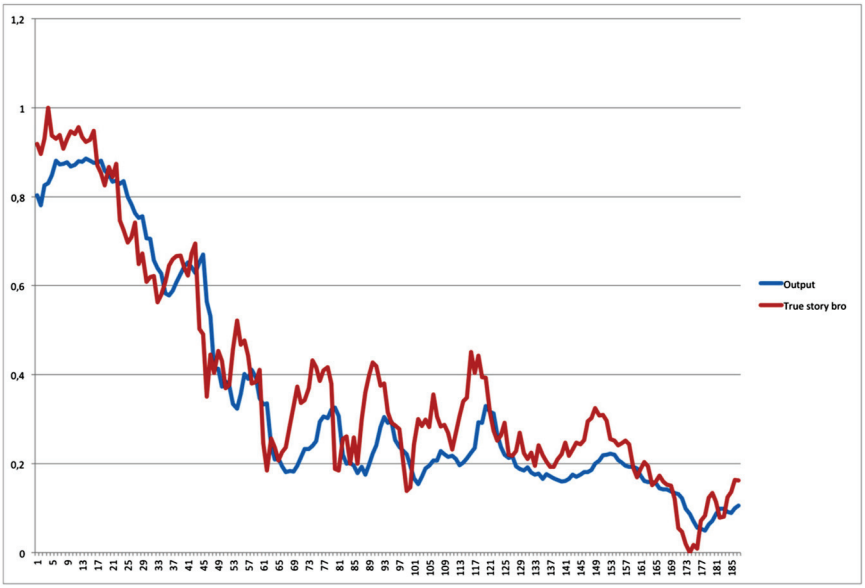


Рис. 1. Предсказание цены на акции с помощью разработанной нейронной сети на тиковых данных

Решения в итоговой, разработанной автором системе, принимаются не только исходя из показателей данной нейронной сети, а выводятся на основе многих индикаторов. Такая система показала стабильный положительный результат при применении к финансовым временным рядам различных интервалов, в том числе и краткосрочных, используемых при высокочастотной торговле.

Заключение

Нейронные сети могут быть успешно использованы для предсказания динамики цен на акции и производные финансовые инструменты на фондовом рынке в сочетании с другими индикаторами. Система, разработанная автором, показала стабильный положительный результат, а доходность при моделировании, учитывающем различные факторы, такие как задержки связи, объем торгов и прочие, колеблется в районе 16%.

Литература

1. *Маркин М. И., Смелянский Р. Л.* Синтез архитектуры нейросетевого аппроксиматора под заданное приложение // Искусственный интеллект. Вып. 2. Донецк, 2000.
2. *Fahlman S. E.* and *Lebiere C.* The cascade-correlation learning architecture // Advances in Neural Information Processing Systems, 2 / D. S. Touretzky, editor, Morgan Kaufmann. Los Altos CA, 1990.

Автоматное управление, эволюционные алгоритмы, верификация на моделях

Шалыто
Анатолий Абрамович

д.т.н.
профессор,
заведующий кафедрой технологии программирования
СПбГУ ИТМО

ВЫБОР ФУНКЦИИ ПРИСПОСОБЛЕННОСТИ ОСОБЕЙ ЭВОЛЮЦИОННОГО АЛГОРИТМА С ПОМОЩЬЮ ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ

А. С. Афанасьева

студентка кафедры компьютерных технологий;
afanasyevarina@gmail.com

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Аннотация: В данной работе предлагается метод, позволяющий динамически выбирать вспомогательную функцию приспособленности, наиболее выгодную для использования в эволюционном алгоритме. Метод основан на применении обучения с подкреплением. Приведены экспериментально полученные результаты его использования для решения модельной задачи, а также результаты, позволяющие сравнить предлагаемый метод с алгоритмами многокритериальной оптимизации.

Введение

В теории оптимизации известны задачи скалярной и многокритериальной оптимизации [1]. На практике возникает возможность решать модифицированную скалярную задачу оптимизации, в которой присутствуют дополнительные критерии [2]. Целью ее решения является максимизация единственной целевой функции, а дополнительные критерии имеют лишь вспомогательное значение. Целевая функция может некоторым образом зависеть от дополнительных критериев, поэтому в некоторых случаях вместо максимизации целевой функции оказывается выгодным оптимизировать дополнительные критерии.

Важным классом алгоритмов оптимизации являются эволюционные алгоритмы (ЭА), где в качестве критерия выступает целевая функция приспособленности (ФП). При наличии нескольких вспомогательных функций приспособленности выбор наиболее выгодной из них приходится производить вручную [2]. Подобный подход не вполне эффективен, так как предполагает многократный перезапуск ЭА.

Целью описываемых исследований является разработка метода, позволяющего автоматически выбирать из заранее подготовленного набора такие вспомогательные ФП, применение которых способствует ускорению «выращивания» особей с высокими значениями целевой ФП. В литературе встречаются разработки по автоматической настройке значений параметров ГА, таких как вероятность применения генетических операторов или число особей в поколении, а также по настройке некоторой фиксированной ФП [5].

Предлагаемый метод отличается от существующих подходов к настройке ГА тем, что предполагает выбор между качественно разными ФП.

Выбор ФП предлагается осуществлять с помощью обучения с подкреплением [3, 4]. Отметим, что применение обучения с подкреплением к оптимизации работы эволюционных алгоритмов мало исследовано на данный момент [5]. В предлагаемой работе подобный подход впервые применяется для контроля ФП.

Существуют также подходы к улучшению производительности алгоритмов однокритериальной оптимизации, основанные на введении многокритериальности [6]. Их применение предполагает разработку дополнительных ФП, таких что получающаяся задача многокритериальной оптимизации решается проще, чем исходная задача. Отличие предлагаемого метода от многокритериального подхода состоит в том, что вспомогательные ФП задаются заранее и могут не коррелировать с целевой функцией. Ниже будут описаны результаты эксперимента, позволяющего сравнить производительность предлагаемого метода и алгоритмов многокритериальной оптимизации.

Описание предлагаемого подхода

Для выбора вспомогательной ФП используется обучение с подкреплением [3, 4]. В алгоритмах этого типа обучение происходит одновременно с применением накопленного опыта, что позволяет выбрать и применить оптимальные ФП в течение одного запуска ЭА.

В алгоритмах обучения с подкреплением агент применяет действия к среде, которая отвечает на каждое действие наградой. В разработанном методе действие агента состоит в выборе ФП для каждого вновь сформированного поколения ЭА. Вознаграждение тем выше, чем существеннее рост значений целевой ФП. На рис. 1 представлена схема предлагаемого метода.

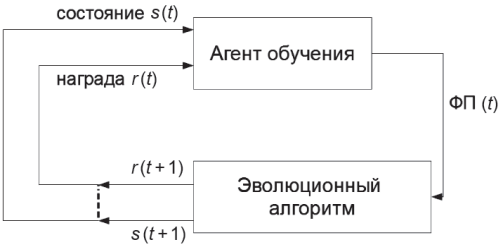


Рис. 1. Схема предлагаемого метода,
t — номер поколения ЭА

Алгоритмы обучения с подкреплением нацелены на формирование оптимальной стратегии поведения агента, следование которой приводит к максимизации суммарного вознаграждения, а следовательно, к ускорению роста целевой ФП.

Результаты экспериментов

Эффективность предлагаемого метода была проверена экспериментальным способом. Было реализовано несколько различных эволюционных алгоритмов и контролирующих их алгоритмов обучения с подкреплением, а именно Q-learning [3], Delayed Q-learning [7], R-learning [8] и Dyna [3]. В ходе первого эксперимента предлагаемый метод использовался для решения модельной задачи, в которой на различных этапах оптимизации выгодны различные вспомогательные ФП. В ходе второго эксперимента было проведено сравнение с методами многокритериальной оптимизации на примере задачи H-IFF [6], применяющейся для тестирования генетических алгоритмов.

Результаты решения модельной задачи

Рассмотрим постановку следующей модельной задачи. Особи представлены битовыми строками. Пусть x бит равны единице. Определим функции приспособленности. Целевая ФП задается формулой $g(x)=[x/k]$. Вспомогательные ФП имеют следующий вид:

$$h_1(x)=\begin{cases} x, x<p \\ p, x\geq p \end{cases}, \quad h_2(x)=\begin{cases} p, x<p \\ x, x\geq p \end{cases}.$$

Будем называть p точкой переключения.

Можно видеть, что для особей с числом единиц меньшим значения точки переключения выгоднее использовать функцию h_1 в качестве текущей ФП. Для особей, число единиц в представлении которых превышает значение точки переключения, выгодно использовать h_2.

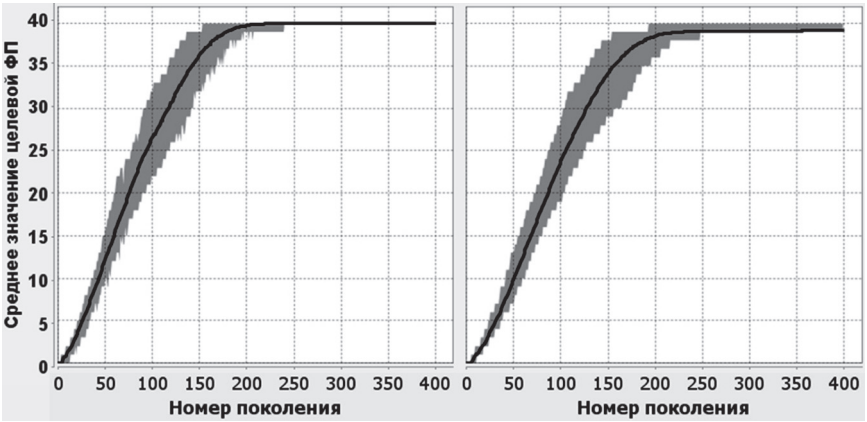


Рис. 2. Сравнительные графики роста ФП лучшей особи во время работы алгоритма, использующего Delayed Q-learning [7] (слева) и обычного ГА (справа)

Поставлен эксперимент, заключающийся в многократных запусках ГА с обучением и обычного ГА. На рис. 2 представлены графики зависимости усредненной целевой ФП лучшей выращенной в текущем поколении особи от номера поколения. Результаты эксперимента подтверждают, что применение обучения ускоряет получение особей с более высокими значениями целевой ФП.

Результаты оптимизации функции H-IFF

Определим задачу скалярной оптимизации функции H-IFF (Hierarchical-if-and-only-if function). Пространство поиска состоит из битовых строк $B = b_1, b_2, \ldots, b_l$ фиксированной длины l. Требуется максимизировать функцию H-IFF [6]:

$$f(B) = \begin{cases} 1, |B| = 1, \\ |B| + f(B_L) + f(B_R), \ |B| > 1 \wedge (\forall i \{b_i = 0\} \vee \forall i \{b_i = 1\}), \\ f(B_L) + f(B_R), \text{ иначе.} \end{cases}$$

Функция задана таким образом, что существует два оптимальных решения: строка, полностью состоящая из единиц, и строка, полностью состоящая из нулей. Особенностью задачи является то, что поиск оптимального решения с помощью эволюционных алгоритмов часто останавливается в локальном оптимуме. Существует подход к решению этой проблемы, при котором скалярная задача оптимизации H-IFF заменяется многокритериальной задачей оптимизации функции MH-IFF [6]. Вместо исходной функции f вводятся функции f_0 и f_1:

$$f_n(B) = \begin{cases} 0, |B| = 1 \wedge b_1 \neq n, \\ 1, |B| = 1 \wedge b_1 = n, \\ |B| + f_n(B_L) + f_n(B_R), |B| > 1 \wedge \forall i \{b_i = n\}, \\ f_n(B_L) + f_n(B_R), \text{ иначе.} \end{cases}$$

Затем проводится максимизация предложенных функций. Этот подход позволяет найти решения с более высокими значениями исходной функции, чем позволяет подход, основанный на скалярной оптимизации.

В ходе эксперимента было реализовано новое решение задачи оптимизации H-IFF с использованием предлагаемого метода. В качестве целевой ФП выступала функция f. В качестве вспомогательных ФП были взяты функции f_0 и f_1, применяемые при оптимизации MH-IFF. Использовалось два различных эволюционных алгоритма: генетический алгоритм (ГА) и $(1+m)$-эволюционная стратегия (ЭС). В ГА с вероятностью 70 % применялся оператор одноточечного кроссовера и оператор мутации, инвертирующий каждый бит каждой особи с вероятностью $2/l$. В ЭС оператор

мутации инвертировал один бит каждой особи, выбранный случайным образом.

Параметры эксперимента соответствовали параметрам, примененным в статье [6], что позволяет сравнить новые результаты с результатами, полученными ее авторами. Длина особи составляла 64 бита. Соответствующее максимально возможное значение H-IFF равно 448. В табл. 1 представлены результаты оптимизации (M)H-IFF с помощью различных алгоритмов. Результаты отсортированы по среднему значению целевой ФП лучших особей, полученных в результате 30 запусков соответствующих алгоритмов. Вычисления запускались на фиксированное число поколений. Алгоритмы 1, 2, 4, 5, 7 реализованы с помощью предлагаемого метода с использованием различных алгоритмов обучения. Результаты 3, 6, 9, 11 получены авторами статьи, причем алгоритмы 3 и 6 (PESA и PAES) являются алгоритмами многокритериальной оптимизации. Можно видеть, что предлагаемый метод в случае использования алгоритма обучения R-learning [8] позволяет преодолеть проблему остановки в локальном оптимуме столь же эффективно, как и метод PESA, и более эффективно, чем метод PAES.

Таблица 1

Результаты оптимизации (M)H-IFF.
Алгоритмы 1, 2, 4, 5, 7 реализованы с применением предлагаемого метода

№	Алгоритм	Лучшее значение	Среднее значение	σ	% одного оптимума	% двух оптимумов
1	(1+10)-ЭС + R-learning	448	448,00	0,00	100	40
2	ГА + R-learning	448	448,00	0,00	100	10
3	PESA	448	448,00	0,00	100	100
4	ГА + Q-learning	448	435,61	32,94	87	3
5	ГА + Dyna	448	433,07	38,07	80	0
6	PAES	448	418,13	50,68	74	43
7	ГА + Delayed QL	448	397,18	49,16	53	0
8	ГА + Random	384	354,67	29,24	0	0
9	DCGA	448	323,93	26,54	3	0
10	ГА	384	304,53	27,55	0	0
11	SHC	336	267,47	29,46	0	0
12	(1+10)-ЭС	228	189,87	17,21	0	0

В табл. 2 отдельно рассмотрена оптимизация H-IFF с применением ЭС. Применяемая ЭС устроена таким образом, что решает задачу весьма неэффективно. Однако применение предлагаемого метода позволяет увеличить среднее значение целевой ФП примерно в два раза.

Таблица 2

Результаты оптимизации H-IFF с помощью эволюционных стратегий. Алгоритмы 1, 3, 5 реализованы с применением предлагаемого метода

№	Алгоритм	Лучшее значение	Среднее значение	σ	% одного оптимума	% двух оптимумов
1	(1+10)-ЭС+R-learning	448	448,00	0,00	100	40
2	(1+10)-ЭС	228	189,87	17,21	0	0
3	(1+5)-ЭС+R-learning	448	448,00	0,00	100	37
4	(1+5)-ЭС	216	179,07	16,99	0	0
5	(1+1)-ЭС+R-learning	448	403,49	59,48	73	10
6	(1+1)-ЭС	188	167,07	11,98	0	0

Заметим, что целью предлагаемого метода является ускорение получения особей с высокими значениями целевой ФП. Приведенные результаты свидетельствуют о том, что метод успешно справляется с этой задачей. Невысокий процент нахождения обоих возможных оптимумов объясняется тем, что предлагаемый метод не проводит многокритериальную оптимизацию. Ожидается, что в случаях, когда среди вспомогательных ФП есть ФП, не коррелирующие с целевой, предлагаемый метод будет эффективнее методов многокритериальной оптимизации.

Заключение

Предложен метод, повышающий эффективность скалярной оптимизации со вспомогательными критериями. Метод основан на выборе функции приспособленности эволюционного алгоритма с помощью обучения с подкреплением. Работа вносит вклад в исследование применимости обучения с подкреплением для контроля работы эволюционных алгоритмов, настройка выбора функций приспособленности с помощью обучения проведена впервые. В ходе экспериментов подтверждена эффективность метода: метод позволяет динамически выбирать наиболее выгодную функцию приспособленности и избегать остановки в локальном оптимуме. Метод, примененный к $(1+m)$-эволюционным стратегиям, позволяет в два раза увеличивать значение целевой функции приспособленности особей, выращиваемых за фиксированное число поколений.

Литература

1. *Лотов А. В., Поспелова И. И.* Многокритериальные задачи принятия решений: Учебное пособие. М.: МАКС Пресс. 2008.
2. *Буздалов М. В.* Генерация тестов для олимпиадных задач по теории графов с использованием эволюционных алгоритмов. Магистерская диссертация. СПбГУ ИТМО. http://is.ifmo.ru/papers/2011-master-buzdalov/ [дата просмотра: 10.05.2012]

3. *Sutton R. S., Barto A. G.* Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, 1998.
4. *Gosavi A.* Reinforcement Learning: A Tutorial Survey and Recent Advances // INFORMS Journal. Vol. 21. No. 2. 2009. P. 178–192.
5. *Eiben A. E., Horvath M., Kowalczyk W., Schut M. C.* Reinforcement learning for online control of evolutionary algorithms // Proceedings of the 4th international conference on Engineering self-organising systems (ESOA»06). Springer-Verlag, Berlin, Heidelberg, 2006. P. 151–160.
6. *Knowles J. D., Watson R. A., Corne D.* Reducing Local Optima in Single-Objective Problems by Multi-objectivization // Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization EMO «01. London, UK: Springer-Verlag, 2001. P. 269–283.
7. *Strehl A. L., Li L., Wiewora E., Langford J., Littman M. L.* PAC model-free reinforcement learning // Proceedings of the 23rd international conference on Machine learning. ICML'06. 2006. P. 881–888.
8. *Mahadevan S.* Average reward reinforcement learning: foundations, algorithms, and empirical results. Machine Learning 22, 1–3. 1996. P. 159–195.

ПРИМЕНЕНИЕ ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ ДЛЯ ПОКРЫТИЯ КОДА ТЕСТАМИ

М. В. Буздалов

ассистент кафедры компьютерных технологий;
mbuzdalov@gmail.com

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Аннотация: В данной работе описываются основные положения применения эволюционных алгоритмов для покрытия кода тестами. Описывается новый вид функции приспособленности теста относительно еще не покрытого фрагмента кода и метод использования такой функции в эволюционных алгоритмах с большим числом особей в поколении.

Введение

Тестирование программного обеспечения (ПО) занимает до 50 % времени и более 50 % стоимости производства ПО [1]. По этой причине, автоматизация тестирования ПО, за счет уменьшения затрачиваемого времени и снижения влияния человеческого фактора, приводит к сокращению периода выхода программного продукта на рынок, снижению его себестоимости и уменьшению числа программных ошибок в нем. Такие методологии производства ПО, как экстремальное программирование [2], уделяют значительное внимание написанию так называемых модульных тестов, а также автоматизации их регулярного запуска, однако, написание модульных тестов выполняется программистами, а следовательно, этот процесс нуждается в усовершенствовании. В частности, актуальной задачей является автоматическая генерация тестов.

Одним из показателей качества набора тестов является то, насколько полно, в смысле возможных путей выполнения и возможных случаев, встречающихся в коде, тестируемый код покрыт тестами. Существует множество критериев покрытия кода тестами, как общих, так и специализированных. Примером общего критерия является число (доля) покрытых строк кода (то есть, число таких строк, которые хотя бы один раз исполнялись при тестировании кода на рассматриваемом наборе тестов). Примером специализированного критерия для тестирования баз данных SQL является доля покрытых тестами операторов SELECT.

Цель работы

В работе рассматривается автоматическая генерация набора тестов, покрывающего код согласно заданному критерию, с применением эволюцион-

ных алгоритмов [3, 4]. Рассматриваются как широко распространенные критерии покрытия кода, так и новые критерии, ориентированные на покрытие значений, в особенности — граничных значений.

Критерии покрытия кода тестами

Существует множество критериев покрытия кода тестами. Перечислим наиболее известные из них:

- покрытие путей;
- покрытие условий;
- покрытие ветвлений;
- покрытие строк кода;
- покрытие методов (функций, процедур);
- покрытие классов.

Покрытие путей — наиболее точный, но практически мало используемый на практике критерий, поскольку в большинстве случаев выполнение тестируемого кода может идти по бесконечному или зависящему экспоненциально от размера кода числу различных путей. Покрытие методов и покрытие классов — достаточно простые (для реализации стопроцентного покрытия) критерии, но, как правило, они не отражают реальную протестированность кода. Покрытие ветвлений — это облегченный вариант покрытия условий. Разницу между этими двумя критериями можно проиллюстрировать с помощью листинга 1.

Листинг 1

Иллюстрация покрытия условий и покрытия ветвлений

```
function a(x, y, z) {
    if (x && y && z) {
        a();
    } else {
        b();
    }
}
```

Покрытие ветвлений (каждая ветвь кода выполнена не менее одного раза) можно достичь двумя тестами:

- `x = y = z = false;`
- `x = y = z = true.`

В то же время покрытие условий потребует как минимум четырех тестов, что позволяет тщательнее проверить корректность условия ветвления:

- `x = y = z = true;`
- `x = y = true, z = false;`
- `x = true, y = z = false;`
- `x = y = z = false.`

Кроме данных критериев покрытия, можно ввести специальные критерии, которые призваны проверять различные крайние и граничные случаи, не обрабатываемые в явном виде с помощью операторов ветвления, например:

- минимальные и максимальные значения;
- значения `null` и не-`null` для ссылочных типов;
- пустые, одноэлементные и многоэлементные структуры данных;
- целочисленное деление на ноль;
- экстремальные значения аргументов функций (например, $\operatorname{arctg}(\pi/2)$);
- значения каждого бита в битовой маске;
- ожидаемые и неожидаемые значения аргумента оператора побитового сдвига.

Такое *покрытие значений* может быть особенно полезным при тестировании кода, работающего с математическими или битовыми операциями, а также со сложными структурами данных. Задача о покрытии тестами кода с данными свойствами часто встречается при подготовке тестов для олимпиадных задач по программированию.

Расстояние до ветви и до значения

Расстояние до ветви — это мера того, насколько требуется изменить входные данные для того, чтобы данная ветвь кода была выполнена. Для каждого типа условий требуется разработать новую функцию, задающую расстояние до ветви. Простейший пример приведен в листинге 2.

Листинг 2

Оператор ветвления и соответствующее расстояние до ветви

```
if (a * 72 > 1336) {
    //Требуемая ветвь
}
```

```
d(a)= 1336 - a * 72
```

Для более сложных условий ветвления функции расстояния до ветви строятся соответствующим образом, при этом может понадобиться профи-

лирование или модификация библиотечного кода (листинг 3). Аналогичным образом можно строить функции *расстояния до значения*.

Листинг 3

Более сложные примеры условий ветвления

```
if (str.startsWith("example")) { /* ... */ }
if (str.matches("[0..9]+")) { /* ... */ }
```

На основе функций ветвления далее будут строиться функции приспособленности, описывающие то, насколько тест близок к покрытию того или иного фрагмента кода или значения переменной.

Описание предлагаемого подхода

Предлагаемый подход к генерации покрывающих тестов принадлежит методологии «тестирования белого ящика», то есть, требует доступа к коду тестируемой программы (исходных кодов или исполняемого кода). Он состоит из двух этапов. На первом этапе, в соответствии с конечной целью генерации набора тестов, выбирается критерий покрытия. Может быть выбран как один из традиционных критериев, такой как доля покрытых инструкций кода или доля покрытых ветвей кода, так и какой-либо другой критерий, который лучше подходит для поставленной задачи.

Второй этап состоит в генерации тестов. Для этого вначале строится несколько случайных тестов, тесты добавляются в итоговый набор, и анализируется полученное ими покрытие. Затем для некоторого непокрытого фрагмента программы, согласно выбранному критерию, строится оптимизируемая функция (или, в терминах эволюционных алгоритмов, функция приспособленности). Функция вычисляется при выполнении тестируемой программы путем модификаций ее исходного кода. В данной работе она имеет сложный вид, зависящий от последовательности выполненных операций и значений, принимаемых переменными в процессе выполнения. Чем «ближе» траектория выполнения программы на тесте подходит к выбранному фрагменту, тем «лучше» значение функции приспособленности.

Для оптимизации указанной функции производится запуск эволюционного алгоритма. Особью алгоритма является тест. Когда фрагмент оказывается покрытым некоторым тестом, этот тест добавляется в итоговый набор, затем вновь анализируется покрытие программы текущим итоговым набором, и описанная процедура повторяется, пока покрытие не будет признано удовлетворяющим поставленной задаче.

Функции приспособленности

В литературе [5, 6] встречаются, главным образом, два варианта задания функции приспособленности теста по отношению к еще не покрытому

фрагменту кода. Первый из них состоит в задании предопределенного пути в графе управления тестируемого кода. Функция приспособленности состоит из двух компонент — длины общего префикса заданного и фактического пути и расстояния до непосещенной ветви в узле, где эти пути расходятся.

Второй вариант отличается от первого тем, что расстояние до непосещенной ветви измеряется не в первом узле, где пути разошлись, а в ближайшем таком узле к цели покрытия. Данная модификация мотивируется тем, что запланированный и фактический пути выполнения после расхождения могут вновь объединяться.

Автором настоящей работы предлагается новый метод, состоящий в том, что путь до целевого узла не задается. Значения функции приспособленности при этом подходе состоят из идентификатора ближайшего к целевому узла, принадлежащего фактическому пути, и значения расстояния до непосещенной ветви в этом узле. При этом сравнимыми оказываются не все такие значения. Таким образом, можно рассматривать аналог Парето-фронта для тестов, приближающегося к цели покрытия по мере улучшения составляющих его тестов. Данный метод призван использовать параллелизм исследования пространства поиска, присущий эволюционным алгоритмам, работающим с поколениями из большого числа особей, таким как генетические алгоритмы. Можно отметить общие черты описанного метода с алгоритмами многокритериальной оптимизации.

Литература

1. *Myers G.* The Art of Software Testing, Second Edition. John Wiley & Sons, 2004.
2. *Бек К.* Экстремальное программирование. Питер, 2002.
3. *Mitchell M.* An Introduction to Genetic Algorithms. MIT Press, 1996.
4. *Luke S.* Essentials of Metaheuristics. Lulu, 2009.
5. *Goldberg A., Wang T. C.,* and *Zimmerman D.* Applications of feasible path analysis to program testing // In Proceedings of International Symposium on Software Testing and Analysis (ISSTA), 1994.
6. *Baars A., Harman M* et al. Symbolic Search-Based Testing // In Proceedings of 26th IEEE/ACM International Conference on Automated Software Engineering, 2011.

ПРИМЕНЕНИЕ МУРАВЬИНЫХ АЛГОРИТМОВ ДЛЯ ПОСТРОЕНИЯ КОНЕЧНЫХ АВТОМАТОВ

Д. С. Чивилихин, В. И. Ульянцев

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Постановка задачи

В рамках парадигмы автоматного программирования ключевыми компонентами программ и программных комплексов являются конечные автоматы. В некоторых случаях автоматы могут быть построены вручную, однако для большинства реальных задач эвристическое построение автоматов является весьма трудоемким. Для автоматической генерации конечных автоматов обычно применяются различные эволюционные алгоритмы.

Управляющий конечный автомат задается множеством состояний, начальным состоянием, множеством входных событий и множеством выходных воздействий, а также функцией перехода, сопоставляющей каждому состоянию автомата и каждому входному событию другое состояние и выходное воздействие.

Для каждой задачи, функция приспособленности — вещественнозначная функция, заданная на множестве всех конечных автоматов с заданными параметрами. В общем случае, задача генерации конечного автомата по заданной функции приспособленности может быть сформулирована следующим образом: по заданному числу состояний, множеству входных событий и множеству выходных воздействий найти конечный автомат с целевым значением заданной функции приспособленности.

Муравьиные алгоритмы — семейство методов решения задач оптимизации, которые могут быть сведены к поиску по графу, таких как, например, задача о коммивояжере. В рамках муравьиных алгоритмов решения строятся набором агентов-муравьев, использующих при выборе пути в графе некоторую стохастическую стратегию. Решения могут быть представлены как путями в графе, так и отдельными его вершинами. В данной работе исследуется возможность применения муравьиных алгоритмов для построения конечных автоматов по заданной функции приспособленности путем сведения этой задачи к поиску целевой вершины в графе.

Цель работы

Целью настоящей работы является разработка метода генерации конечных автоматов для заданной функции приспособленности на основе муравьиных алгоритмов и оценка его эффективности.

Описание предлагаемого подхода

В качестве представления пространства поиска, то есть множества всех конечных автоматов с заданными параметрами, рассматривается граф, вершины которого соответствуют решениям (конечным автоматам) а ребра — *мутациям* конечных автоматов. Под мутацией конечного автомата понимается небольшое изменение его структуры, например, изменение выходного воздействия на переходе или изменение состояния, в которое ведет переход.

Каждому ребру графа ставится в соответствие некоторое вещественное число — значение *феромона*. Значение феромона на ребре может быть увеличено, если по этому ребру прошел муравей. Также, на каждой итерации алгоритма происходит испарение — уменьшения значений феромона на всех ребрах графа в одинаковое число раз. Алгоритм генерации автомата может быть разделен на несколько этапов:

1. **Построение решений муравьями-агентами.**
   Муравей, находясь в определенной вершине графа, выбирает следующую вершину руководствуясь одним из двух правил:
   a) построить новые ребра графа, производя мутации автомата, соответствующего текущей вершине и выбрать лучшее из новых решений;
   b) выбрать следующую вершину исходя из значений феромона на ребрах, ведущих из текущей вершины по классической формуле выбора пути в муравьиных алгоритмах.
2. **Обновление значений феромона на ребрах.**
3. **Проверка условий останова.**

Шаги 1–3 повторяются до тех пор, пока не будет найдено решение, удовлетворяющее условиям задачи, или не выполнится другое условие останова, например, истечет заданное максимальное число итераций алгоритма

Результаты

Был разработан и реализован метод генерации конечных автоматов по заданной функции приспособленности, основанный на муравьином алгоритме. Эффективность разработанного метода была оценена путем сравнения с генетическими алгоритмами для задачи об «Умном муравье» и задаче о генерации автоматов на основе тестовых примеров. Эксперименты показали, что среднее время работы предлагаемого алгоритма для построения целевых автоматов в этих задачах в несколько раз меньше времени работы генетического алгоритма.

ПОСТРОЕНИЕ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ НА ОСНОВЕ ВЕРИФИКАЦИИ И СЦЕНАРИЕВ РАБОТЫ

К. В. Егоров

аспирант кафедры компьютерных технологий;
kegorof@gmail.com

Ф. Н. Царев

аспирант кафедры компьютерных технологий;
fedor.tsarev@gmail.com

А. А. Шалыто

д.т.н., профессор, заведующий кафедрой технологии программирования;
shalyto@mail.ifmo.ru

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Аннотация: Настоящая работа описывает способ построения автоматов управления на основе верификации, позитивных и негативных сценариев работы. Предлагается применять генетический алгоритм, где особь в каждом поколении — это конечный автомат, а временные утверждения и сценарии работы учитываются при вычислении функции приспособленности, мутации и скрещивании.

Введение

Автоматное программирование — это парадигма программирования, в рамках которой программы предлагается проектировать в виде совокупности взаимодействующих автоматизированных объектов управления [1]. В автоматных программах выделяют три типа объектов: поставщики событий, система управления и объекты управления. Система управления представляет собой конечный автомат или систему взаимодействующих конечных автоматов. Поставщики событий генерируют события, а система управления по каждому событию может совершать переход, считывая значения входных переменных у объектов управления для проверки условия перехода.

Существуют различные способы построения автоматов управления со сложным поведением. Чаще всего такие системы строятся эвристически, но они могут содержать ошибки и требуют дополнительных проверок. В работе [2] был предложен способ построения автоматов с помощью генетического программирования на основе тестовых примеров. Однако такой вариант построения конечных автоматов требовал дополнительной валидации и верификации, а в случае обнаружения ошибки, необходимо было изменять тестовые примеры и заново строить систему. В работах [3–5] было предло-

жено строить систему на основе обучающих примеров совместно с темпоральными формулами. Формулы записываются на языке логики линейного времени (*Linear Temporal Logic, LTL*) и позволяют утверждать, что построенная система соответствует заявленной спецификации. Результат верификации учитывается при мутации, скрещивании и при вычислении функции приспособленности, причем вклад каждой темпоральной формулы — значение на отрезке [0, 1]. При этом: 0 — формула нарушается сразу же в стартовом состоянии, 1 — формула выполняется.

Построение автоматов управления на основе сценариев работы

Позитивный сценарий работы представляет собой последовательность пар: входное воздействие и соответствующий ему список выходных воздействий. Такой сценарий должен выполняться в требуемом конечном автомате. В тестовых примерах [2] не было однозначного соответствия между входными воздействиями и выходными, каждый тест записывался как входная последовательность и ожидаемая выходная последовательность воздействий. Теперь же предлагается перейти к интуитивно понятному представлению теста, когда мы заранее знаем список действий на каждое из событий. Это в некоторой степени сужение задачи, которое позволяет ускорить построение конечного автомата управления.

Негативный сценарий представляет собой последовательность входных воздействий, которая, в противоположность позитивной последовательности, не должна выполняться в автомате управления. Причем все префиксы негативного сценария выполняются, только последний переход не должен совершаться. Как неоднократно замечалось в предыдущих работах [4, 5], нельзя утверждать о правильности построенной системы только на основе тестов. Это же утверждение переносится и на сценарии работы (позитивные и негативные), так как они не имеют такой же выразительности как временные утверждения.

Особь в каждом поколении представляет собой конечный автомат с событиями на переходах и с переменным числом выходных воздействий, подобно методу на основе тестовых примеров из работы [2]. Выходные воздействия для каждого из переходов определяются алгоритмом расстановки пометок. Его основная идея состоит в том, что среди всех полностью или частично проходящих позитивных сценариев работы выбирается та последовательность выходных воздействий для выбранного перехода, которая чаще всего встречается.

Темпоральные формулы и сценарии работы используются при вычислении функции приспособленности, при скрещивании и при мутации. Вклад позитивных сценариев вычисляется при помощи редакционного расстояния [6]. Для его вычисления выполняются следующие действия: на вход авто-

мату подается каждая из последовательностей *Input[i]*. Обозначим последовательность выходных воздействий, которую сгенерировал автомат на входе *Input[i]*, как *Output[i]*. После этого вычисляется величина FF_{ptest}:

$$FF_{ptest} = \frac{\sum_{i=1}^{n} \left(1 - \frac{ED(Output[i], Answer[i])}{\max(|Output[i]|, |Answer[i]|)}\right)}{n}.$$

Здесь как *ED(A, B)* обозначено редакционное расстояние между строками *A* и *B*, как *Answer[i]* обозначена эталонная выходная последовательность, которую должен генерировать автомат на входе *Input[i]*. Отметим, что значения этой функции лежат в пределах от 0 до 1, при этом, чем «лучше» автомат соответствует позитивному сценарию, тем больше значение функции приспособленности.

Вклад в функцию приспособленности каждого негативного сценария дискретен. Если существует последовательность событий в конечном автомате, на которой сценарий выполняется, значит он проходит, и его вклад — –1. Если последовательности не существует, значит вклад сценария равен 0. Вклад всех негативных сценариев считается как отношение суммы вкладов каждого из сценария к общему числу позитивных сценариев. Конечная формула функции приспособленности принимает вид:

$$FF = FF_{ptest} + FF_{ptest} \times FF_{LTL} - FF_{ptest} \times FF_{ntest}.$$

Здесь FF_{ptest} — вклад позитивных сценариев работы, FF_{LTL} — вклад LTL-формул, FF_{ntest} — вклад негативных сценариев. Каждый из вкладов находится в интервале [0, 1].

Скрещивание двух автоматов управления описано в работах [2, 4]. Данный процесс может учитывать как LTL-формулы, так и сценарии работы (позитивные и негативные). Скрещивание по тестам [2] переносится на позитивные сценарии без изменений, его основная идея заключается в том, что выбирается несколько лучше всего проходящих сценариев, и переходы, задействованные в выбранных сценариях, переходят в новую особь из двух родителей без изменений, а оставшиеся распределяются между потомками случайным образом.

Мутация может случайным образом изменить входное воздействие перехода, его конечное состояние или удалить переход конечного автомата. Переход, нарушающий LTL-формулу или позволяющий пройти негативному сценарию, с большей вероятностью подвергается мутации, чем остальные.

Экспериментальные исследования

Были проведены экспериментальные исследования на примере автомата управления дверьми лифта из работы [4]. Было проведено 1000 построений

для каждого из перечисленных выше методов. При использовании только тестовых примеров построенный автомат менее чем в 1% случаев соответствовал спецификации, при использовании совместно верификации и тестовых примеров среднее число вычислений функции приспособленности оказалось равным — 827 857. При совместном использовании тестов и контрактов [5] — 710 882. При использовании верификации совместно с позитивными и негативными сценариями работы — 161 592.

Заключение

В работе исследована возможность применения верификации совместно с позитивными и негативными сценариями работы для автоматического построения автоматов управления систем со сложным поведением, предложен метод мутации и скрещивания на основе негативных сценариев работы, проведено экспериментальное исследование метода на примере построения автомата управления дверьми лифта.

В результате проведенных экспериментов было показано, что применение сценариев работы приводит к ускорению построения автомата управления. Однако заметим, что как тестовыми примерами, так и сценариями работы не всегда можно полностью описать поведение системы, так что целесообразно использовать их совместно с LTL-формулами.

Литература

1. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2010.
2. *Царев Ф. Н.* Метод построения автоматов управления системами со сложным поведением на основе тестов с помощью генетического программирования // Материалы Международной научной конференции «Компьютерные науки и информационные технологии». Саратов: СГУ, 2009. С. 216–219.
3. *Johnson C.* Genetic Programming with Fitness based on Model Checking. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2007. V. 4445. Pp. 114–124.
4. *Егоров К. В., Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе обучающих примеров и спецификации // Научно-технический вестник СПбГУ ИТМО. 2010. № 69. С. 81–85.
5. *Егоров К. В., Шалыто А. А.* Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе контрактов и тестовых примеров // Сборник научных трудов VI Международной научно-практической конференции «Интегрированные модели и мягкие вычисления в искусственном интеллекте. М.: Физмалит, 2011. С. 610–615.
6. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады Академии Наук СССР. 163.4. С. 845–848.

МЕТОД СБОРКИ КОНТИГОВ ГЕНОМНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ НА ОСНОВЕ СОВМЕСТНОГО ПРИМЕНЕНИЯ ГРАФОВ ДЕ БРЮИНА И ГРАФОВ ПЕРЕКРЫТИЙ[1]

А. В. Александров

магистрант кафедры компьютерных технологий; alexandrov@rain.ifmo.ru

С. В. Казаков

магистрант кафедры компьютерных технологий; svkazakov@rain.ifmo.ru

С. В. Мельников

студент кафедры компьютерных технологий; melnikov@rain.ifmo.ru

А. А. Сергушичев

магистрант кафедры компьютерных технологий; alserg@rain.ifmo.ru

Ф. Н. Царев

аспирант кафедры компьютерных технологий; fedor.tsarev@gmail.com

А. А. Шалыто

д.т.н., проф., зав. кафедрой технологий программирования; shalyto@mail.ifmo.ru

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Аннотация: В работе предлагается метод сборки контигов геномных последовательностей. Особенностью этого метода является разбиение процесса сборки контигов на два этапа — сборка квазиконтигов из чтений и сборка контигов из квазиконтигов. На первом этапе используется граф де Брюина, на втором — граф перекрытий. Описываются результаты экспериментального исследования разработанного метода на чтениях генома рыбы *Maylandia zebra*, размер генома которой оценивается в один миллиард нуклеотидов. С помощью разработанного метода контиги генома этой рыбы были собраны за пять суток на компьютере с двумя 4-ядерными процессорами и 32 ГБ оперативной памяти.

[1] Исследования выполняются в рамках государственных контрактов № 07.514.11.4010 (заключен в рамках Федеральной целевой программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007–2013 годы») и № 16.740.11.0495 (заключен в рамках Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России на 2009–2013 годы»).

Введение

Многие современные задачи биологии и медицины требуют знания геномов живых организмов, который состоит из нескольких нуклеотидных последовательностей молекул дезоксирибонуклеиновой кислоты (ДНК). Поэтому возникает необходимость в дешевом и быстром методе секвенирования — определения последовательности нуклеотидов в образце ДНК.

Существующие секвенаторы — устройства для чтения ДНК — не позволяют считать за один раз всю молекулу ДНК. Вместо этого они позволяют читать фрагменты генома небольшой длины. Длина фрагмента может быть разной, она является важным параметром секвенирования — от нее напрямую зависит стоимость секвенирования и время, затрачиваемое на чтение одного фрагмента: чем больше длина считываемого фрагмента, тем выше стоимость чтения и тем дольше это чтение происходит. В связи с этим сейчас получил распространение следующий дешевый и эффективный подход: сначала вычленяется случайно расположенный в геноме фрагмент длиной около 500 нуклеотидов, а затем считываются его префикс и суффикс (длиной порядка 80–120 нуклеотидов каждый). Эти префикс и суффикс называются *парными чтениями*. Описанный процесс повторяется такое число раз, чтобы обеспечить достаточно большое покрытие генома чтениями. Указанным образом работают, например, секвенаторы компании *Illumina* [1].

Отметим, что описанные выше префикс и суффикс читаются с разных нитей ДНК: один — с прямой, другой — с обратно-комплементарной, причем неизвестно, который откуда. Поэтому удобно рассматривать геном и чтения, дополненные своими обратно-комплементарными копиями.

Задачей сборки генома является восстановление последовательности ДНК (ее длина составляет от миллионов до миллиардов нуклеотидов у разных живых существ) на основании информации, полученной в результате секвенирования.

Предлагаемый метод

Процесс сборки делится, как правило, на следующие этапы:

1. Исправление ошибок в данных секвенирования.
2. Сборка *контигов* — максимальных непрерывных последовательностей нуклеотидов, которые удалось восстановить.
3. Построение *скэффолдов* — последовательностей контигов, разделенных промежутками, для длины которых найдены верхние и нижние оценки.

Одной из наиболее часто используемых при сборке генома математических моделей является так называемый граф де Брюина [2]. На его использовании основаны следующие программные средства: *Velvet* [3], *Allpaths* [4], *AbySS* [5], *SOAPdenovo* [6], *EULER* [7].

Одним из недостатков, которым обладают перечисленные программные средства, является большой объем оперативной памяти, необходимый им для сборки генома, сходного по размерам с геномом человека (2–3 миллиарда нуклеотидов). Так, например, *SOAPdenovo* необходимо порядка 140 ГБ оперативной памяти, а *ABySS* — 21 компьютер с 16 ГБ каждый (всего — 336 ГБ). Такие затраты памяти обусловлены наличием ошибок секвенирования в исходных данных (такие ошибки ведут к увеличению размера графа де Брюина), а также неоптимальным методом хранения этого графа. Другим недостатком существующих методов является отсутствие внутреннего контроля качества сборки.

В настоящей работе предлагается метод, лишенный указанных недостатков. Построение скэффолдов в настоящей работе не рассматривается.

Сборка контигов в предлагаемом методе выполняется в два этапа:

4. Сборка квазиконтигов из чтений геномной последовательности. Квазиконтигами называются последовательности, которые, с одной стороны, длиннее чтений, но, с другой стороны, не являются контигами в смысле невозможности наращивания вправо и влево. Этот этап выполняется с использованием графа де Брюина.
5. Сборка контигов из квазиконтигов. Выполняется с использованием графа перекрытий и метода *Overlap-Layout-Consensus* [8].

Экспериментальные исследования

Экспериментальные исследования разработанного метода проводились в рамках проекта *Assemblathon 2* [9], организованного Калифорнийским университетом в Дэвисе (University of California, Davis) , на одном из наборов данных, который был подготовлен организаторами, — наборе чтений рыбы *Maylandia zebra*. Размер генома этой рыбы оценивается примерно в один миллиард нуклеотидов.

Для сборки контигов использовался набор чтений со средним размером фрагмента 180 и 60-кратным покрытием. Общий объем исходных данных составлял 140 ГБ (в сжатом виде), из них авторами были использованы только 44 ГБ.

Алгоритмы сборки генома были реализованы на языке программирования *Java*. Для запуска программ использовался компьютер с 32 ГБ оперативной памяти и двумя 4-ядерными процессорами. Суммарное время работы всех трех этапов — исправления ошибок, сборки квазиконтигов и сборки контигов — составило пять суток. Опишем подробнее результаты каждого из этапов.

Перед исправлением ошибок чтения были обрезаны, чтобы вероятность отдельной ошибки в каждом нуклеотиде не превышала 10 %. После этого длина всех чтений в среднем уменьшилась на 20 %. Исправление ошибок работало в течение 42 часов. В результате было найдено 150 миллионов ис-

правлений. Всего чтений было 600 миллионов, поэтому было исправлено в среднем каждое четвертое чтение. Сборка квазиконтигов заняла 38 часов. Квазиконтиги были получены из 60 % чтений. Сборка контигов выполнилась за 26 часов. В результате было получено 734 165 контигов, суммарный размер которых составляет $680 \cdot 10^6$ нуклеотидов. Длина максимального составляет 23 514 нуклеотидов, средняя длина — 927, значение метрики N50 — 1799.

Заключение

Предложен метод сборки контигов геномных последовательностей, основанный на совместном использовании графа де Брюина и графа перекрытий. Экспериментальное исследование этого метода проведено в рамках проекта *Assemblathon 2*. Это экспериментальное исследование показало, что с помощью разработанного метода можно собирать геномы размером в миллиард нуклеотидов быстрее чем за неделю.

Литература

1. Illumina, Inc. [Электронный ресурс]. — Режим доступа: http://www.illumina.com/, свободный. Яз. англ. (дата обращения 17.04.2012).
2. *Pevzner P. A.* 1-Tuple DNA sequencing: computer analysis // J. Biomol. Struct. Dyn. 1989. Vol. 7. Pp. 63–73.
3. *Zerbino D. R., Birney E.* Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. Genome Research. 2008. Vol. 18. Pp. 821–829.
4. *Butler J., MacCallum I., Kleber M., Shlyakhter I. A., Belmonte M. K., Lander E. S., Nusbaum C., Jaffe D. B.* ALLPATHS: De novo assembly of wholegenome shotgun microreads // Genome Research. 2008. Vol. 18. Pp. 810–820.
5. *Simpson J. T., Wong K., Jackman S. D., Schein J. E., Jones S. J., Birol I.* ABySS: A parallel assembler for short read sequence data // Genome Research. 2009. Vol. 19. Pp. 1117–1123.
6. *Li R., Zhu H., Ruan J., Qian W., Fang X., Shi Z., Li Y., Li S., Shan G., Kristiansen K.*, et al. De novo assembly of human genomes with massively parallel short read sequencing // Genome Research. 2010. Vol. 20. Pp. 265–272.
7. *Pevzner P. A., Tang H., Waterman M. S.* EULER: An Eulerian path approach to DNA fragment assembly // Proc. Natl. Acad. Sci. 2001. No. 98. Pp. 9748–9753.
8. International Human Genome Sequencing Consortium. 2001. Initial sequencing and analysis of the human genome // Nature. Vol. 409. No. 6822. Pp. 860–921.
9. Проект Assemblathon 2. [Электронный ресурс]. — Режим доступа: http://www.assemblathon.org, свободный. Яз. англ. (дата обращения 17.04.2012).

МЕТОД СБОРКИ ГЕНОМА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ MAPREDUCE[1]

А. В. Александров

магистрант кафедры компьютерных технологий; alexandrov@rain.ifmo.ru

С. В. Казаков

магистрант кафедры компьютерных технологий; svkazakov@rain.ifmo.ru

С. В. Мельников

студент кафедры компьютерных технологий; melnikov@rain.ifmo.ru

А. А. Сергушичев

магистрант кафедры компьютерных технологий; alserg@rain.ifmo.ru

Ф. Н. Царев

аспирант кафедры компьютерных технологий; fedor.tsarev@gmail.com

А. А. Шалыто

д.т.н., проф., зав. кафедрой технологий программирования; shalyto@mail.ifmo.ru

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Аннотация: В работе представлены алгоритмы сборки протяженных фрагментов геномной последовательности (контигов), основанные на технологии *MapReduce*. Данные алгоритмы являются сверхмасштабируемыми параллельными и предназначены для сборки геномных последовательностей. Они применимы, в том числе на кластерах петафлопсного и экзафлопсного уровней производительности.

Введение

Многие современные задачи биологии и медицины требуют знания геномов живых организмов, которые состоят из нескольких нуклеотидных последовательностей молекул дезоксирибонуклеиновой кислоты (ДНК). Поэтому возникает необходимость в дешевом и быстром методе секвенирования — методе определения последовательности нуклеотидов в образце ДНК.

Изучение генома человека и других живых существ имеет важное прикладное значение. На основании результатов сборки генома конкретного

[1] Исследования выполняются в рамках государственного контракта № 16.740.11.0495 (заключен в рамках Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России на 2009–2013 годы»).

При проведении работ был использован суперкомпьютер «Ломоносов» МГУ имени М. В. Ломоносова.

человека возможна реализация персонифицированной медицины — определения предрасположенности человека к различным болезням, создание индивидуальных лекарств и т. д. Кроме этого, на основе результатов исследования геномов растений и животных с использованием методов биоинженерии могут быть выведены новые их виды, обладающие определенными свойствами.

Задача разработки методов сборки геномных последовательностей является, в определенном смысле, центральной среди всех задач биоинформатики. Это объясняется тем, что без ее решения нельзя приступить к детальному изучению генома живого существа и его анализу с применением других алгоритмов биоинформатики.

В середине первого десятилетия XXI века широкое распространение получили так называемые технологии *next generation sequencing* (технологии секвенирования нового поколения). По оценкам экспертов [1] эти технологии в настоящее время развиваются существенно быстрее, чем компьютерные технологии и алгоритмы сборки геномных последовательностей.

Сборка генома из набора фрагментов, полученных на секвенаторе, — алгоритмически и вычислительно сложная задача, решение которой невозможно без использования кластеров. Например, каждая сборка генома печеночного сосальщика, основанная на данных нескольких запусков секвенатора, требует до недели работы кластера из двух десятков узлов по восемь ядер и 8 Гб оперативной памяти в каждом, объединенных по интерфейсу *MPI* [2]. Однако одного запуска почти всегда недостаточно — таких сборок может быть несколько из-за необходимости подбора оптимальных параметров алгоритма и добавления новых экспериментальных данных.

На сегодня процедура работы секвенатора и сборки генома на кластере отличаются по времени в зависимости от конкретного оборудования и используемых алгоритмов, но в целом — это величины одного порядка. Выше было отмечено, что в ближайшие годы темпы роста производительности секвенаторов ожидаются более высокими по сравнению с ростом производительности кластеров, и поэтому «узким» местом в получении геномной последовательности будет именно процедура восстановления генома, выполняемая после получения результатов работы секвенатора.

Использование существующих в настоящее время алгоритмов на персональных компьютерах приведет к тому, что сборка одного генома займет месяцы, а может растянуться и на год. Для успешного решения этой задачи необходимо переходить на новые алгоритмы для кластеров, в том числе петафлопсного и экзафлопсного уровней производительности.

В настоящей работе предлагается сверхмасшитабируемый параллельный метод сборки геномных последовательностей, который основан на использовании технологии *MapReduce* [4]. Сборку генома предлагается осуществлять в три этапа — исправление ошибок в чтениях, сборка квазиконтигов, сборка контигов. Алгоритмы для каждого из этапов отличаются от своих «последо-

вательных» версий, предложенных авторами в работах [5, 6]. Предлагаемые алгоритмы построены на основе распараллеливания по данным.

Основной идеей масштабирования алгоритма исправления ошибок в данных секвенирования (наборе чтений геномной последовательности) является независимая обработка групп *k*-меров с различными префиксами. Для каждой такой группы *k*-меров независимо определяются, какие k-меры являются достоверными, и в них нет ошибок чтения, а в каких вероятность ошибки высока. Для тех *k*-меров, которые не являются достоверными, определяется, какие нуклеотиды в них были прочитаны с ошибками, и на какие нуклеотиды их необходимо исправить. После этого для каждого исходного чтения геномной последовательности выполняется исправление ошибок на основании информации о том, как надо исправлять каждый *k*-мер.

Идея распараллеливания алгоритма последующих шагов: сборки квазиконтигов и сборки контигов также состоит в распараллеливании по данным — исходные данные (чтения геномной последовательности с внесенными в них исправлениями) разбиваются на группы, в каждой из которых они были прочитаны из близких позиций исходной геномной последовательности. Далее эти группы обрабатывается независимо друг от друга алгоритмом, аналогичным алгоритму сборки квазиконтигов [5]. Для такого разбиения разработан алгоритм кластеризации чтений геномной последовательности. Он основан на построении графа общих *k*-меров чтений, в котором вершины соответствуют чтениям, а ребра числу общих *k*-меров у соответствующих чтений, и последующем выделении компонент с большим числом ребер внутри них. Для выделения таких компонент применяется аналог алгоритма обхода в ширину, реализованный при помощи технологии *MapReduce* [3]. В каждую компоненту входит некоторая вершина этого графа и вершины расположенные на расстоянии, не превосходящем заданную величину.

Для сборки контигов из квазиконтигов последние разбиваются на группы, близких по положению в геноме. Для каждой из таких групп применятся алгоритм, основанный на подходе *Overlap-Layout-Consensus*. Для осуществления разбиения квазиконтигов на группы применяется алгоритм, аналогичный описанному выше алгоритму кластеризации чтений геномной последовательности.

Для увеличения размера получаемых контигов, выполняется несколько итераций сборки контигов, при этом контиги, полученные на предыдущей итерации, используются в качестве входных данных для следующей.

Предложенные алгоритмы были реализованы на программном фреймворке *Apache Hadoop* [4]. Были проведена сборка генома бактерии *E. Coli* [7] на кластере НИИ НКТ (НИУ ИТМО). В результате эксперимента были получены контиги с метрикой N50 более 2000, что показывает применимость данного алгоритма для сборки бактериальных геномов.

Также была проведена сборка чтений искусственного генома, использовавшегося в проекте *de novo Genome Assembly Assessment Project* (размер ге-

нома 1,8 миллиарда нуклеотидов) [8] на суперкомпьютере «Ломоносов» МГУ имени М. В. Ломоносова. При запуске на 30000 процессорных ядер были исправлены ошибки в чтениях указанного генома, однако проблемы с записью в файловую систему не позволили провести остальные этапы сборки. Данный эксперимент показал, что предложенный метод исправления ошибок масштабируется на большое число узлов.

Литература

1. *Зубов В. В.* Приборы для чтения ДНК // Химия и жизнь. 2010. №7. С. 4–7; www.dubna-oez.ru/images/data/gallery/10_2948_.pps [дата просмотра: 20.04.2012].
2. *Прохорчук Е. Б.* Код жизни: прочесть не значит понять. http://biomolecula.ru/content/778/ [дата просмотра: 20.04.2012].
3. *Cohen J.* Graph Twiddling in a MapReduce World // Computing in Science & Engineering. 2009. Vol. 11. No. 4. Зр. 29–41.
4. Apache Hadoop. [Электронный ресурс]. — Режим доступа: http://hadoop.apache.org/, свободный. Яз. англ. [дата просмотра 11.04.2012].
5. Разработка метода сборки геномных последовательностей на основе восстановления фрагментов по парным чтениям. Отчет за второй этап. НИУ ИТМО. 2011.
6. *Александров А. В., Казаков С. В., Мельников С. В., Сергушичев А. А., Царев Ф. Н., Шалыто А. А.* Метод исправления ошибок в наборе чтений нуклеотидной последовательности // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. 2011. № 5. С. 81–84.
7. NCBI: Experiment: SRX000429 — Illumina sequencing of Escherichia coli str. K-12 substr. MG1655 genomic paired-end library
8. De novo Genome Assembly Assesment Project. [Электронный ресурс]. — Режим доступа: http://cnag.bsc.es, свободный. Яз. англ. (дата обращения 11.04.2012).

DEPENDENT POLYVARIADIC FUNCTIONS

Jan Malakhovski

graduate student at NRU ITMO; trojan@rain.ifmo.ru

Abstract: The proposition that polyvariadic functions could be defined within dependently typed setting is a folklore in the functional programming community. We could not, however, find any explicit evidence of this fact.

Here we present an explicit implementation of this folklore problem and propose a novel approach for implementing programs usually defined using templates with dependently typed polyvariadic functions.

It is impossible to replace conventional term generation techniques altogether with our approach, because it does not provide any means for reflection. However, it covers a considerable amount of code usually generated with them, yet does not require any special support from the compiler and is completely orthogonal to other type-system extensions.

Our approach is formalized in *Agda* programming language.

Introduction

In functional programming languages, such as Haskell [1], polymorphic data types and functions are usually defined by the abstraction over type variables. In mainstream imperative languages, such as C++ [2], generic programming is done with overloading and different types of templates. Functions that are generic not only by types of variables they receive, but by the *structure* of data they process (e.g. `show` function from `Show` type class in Haskell) as well, are usually defined directly by the compiler/interpreter (e.g. *deriving* construction in Haskell, `str` and `repr` functions in Python [3]) or with templates and reflection.

From type theoretic point of view, templates are functions from *non-ground* type universe, and so their execution is done at compile time. Indeed, templating engines of all mainstream programming languages use methods that are very close to β-reduction from lambda calculus. This property means that conventional generic programming leads to code duplication at different levels, because every pure function has at least two representations: an imperative one for runtime, and a functional one for template engine.

Reflection machinery like Template Haskell [4], that represents terms of language with a description that does not describe itself, usually does not allow to reflect terms that operate on the description language level and above, because each new level of such reflection needs a new set of operational primitives. The work [5] proposes a possible self-supporting representation for datatypes, which in effect allows to describe description language in terms of base language without any external primitives.

... Data-generic programming thus becomes ordinary programming. ...

There is, however, a special case of generic programming, which apparently could not be described by templates with simple substitution model, yet does not need any extensions of conventional dependent type systems: polyvariadic programming [6], i.e. programming generic by the number and/or types of function arguments.

Classical examples of this kind of functions are:

- `printf` function from *C* programming language, which could be type-safely defined with Template Haskell or Variadic Templates from the new C++11 standard [7];
- numerous *lifting* functions, which lift plain *n*-ary functions into *n*-ary functions that preprocess their arguments before applying them to a given plain function.

Here, we propose and study typing schemes for functions polymorphic by the number of their arguments. We directly formalize our definitions in universe polymorphic way within dependently typed Agda programming language [8]. We define base combinators, which allow us to describe desired functions in close to *point-free* style, deliberately avoiding definitions that need anything more powerful than natural induction. In doing so, we pinpoint any subtle properties of type systems that complicate our definitions. Once we defined all obvious combinators arising from our typing scheme, we show how one can use them to define lifting functions on the concrete example of embedded domain specific language (*eDSL*) for logical expressions over arbitrary data types.

We aim to provide simple usable combinators for implementing functions commonly written with templates and reflection, yet without using this machinery. Every generalization (universe polymorphism, indexing) we introduce is due to real use cases we have implemented with our approach.

Each section of this document is *Literate Agda* program with module and import declarations stripped for space-saving. Everything missing on listings is defined in the standard library.

Basic Combinators

Consider the simplest possible universe polymorphic definition of the function composition:

```
infixr 9 _∘′_
_∘′_ : ∀ {ℓ₁ ℓ₂ ℓ₃} {A : Set ℓ₁}
         {B : Set ℓ₂} {C : Set ℓ₃}
     → (B → C) → (A → B) → (A → C)
f ∘′ g = λ x -> f (g x)
```

From the point of view of its first argument (function `f`) function composition changes the domain of `f`. Dually, from the point of view of function `g` it changes its the codomain. In other words, we could argue that function composition is actually a specialized form of two different functions (pseudo-Agda code):

```
changeDomain : (X → A) → (A ~> B) → (X ~> B)
changeCodomain : (B → X) → (A ~> B) → (A ~> X)
```

where the second argument of each of these functions is an element (arrow) of some category. I.e. `changeCodomain` is a `fmap` (functor map) between coslice categories of the form `(A ~>)` and `changeDomain` is a dual to `fmap` for slice categories `(~> B)`.

There are two most simple (without introducing new type variables) possible generalizations of the type `A → B` within the function space:

- `A → A → ··· → A → B` where `A` is repeated *n* times. We'll write types like that as `A ^ n → B` from here on.
- `A → B → B → ··· → B` where `B` is repeated *m* times.

Note, however, that the latter type is equivalent to `A → (B ^ (m - 1) → B)`, which is a function from `A` to an element of the special case of the former type. That means there is only one simple general form: `A ^ n → B`.

We would like to define this form directly in Agda in universe polymorphic way:

```
_^_→_ : ∀ {a b} → Set a → N → Set b → Set (a ⊔ b)
A ^ zero → B = B
A ^ (suc n) → B = A → (A ^ n → B)
```

but Agda's universe hierarchy is not cumulative and this code wouldn't type check. We could fix that by lifting `B` with `Lift` type from Agda standard library's `Level` module. However, in this case, lifting and lowering would significally complicate all our following definitions. That's why we'll make our definition a bit less general by disallowing types without at least one arrow.

```
_^_→_ : ∀ {a b} → Set a → N → Set b → Set (a ⊔ b)
A ^ zero → B = A → B
A ^ (suc n) → B = A → (A ^ n → B)
```

Note that we could generalize this type more by introducing more complicated type schemes. E.g., in the most obvious case, by allowing indexing by `Vec`tors of elements of `Set`s to generate types like `A → B → A → B → ··· → C` and alike and/or making `A`s and `B`s dependent on each other. In that follows we show that at least these simple generalizations in conjunction with simple base combinators are just useful syntax sugar, not the theoretical generalization.

We can now define our `change(Co)Domain` functions:

```
cdom : ∀ {ℓ₁ ℓ₂ ℓ₃} {A : Set ℓ₁}
         {B : Set ℓ₂} {C : Set ℓ₃} n
     → (A → B) → (B ^ n → C) → (A ^ n → C)
cdom zero g f = f °′ g
cdom (suc n) g f = λ x -> cdom n g $ f °′ g $ x

ccodom : ∀ {ℓ₁ ℓ₂ ℓ₃} {A : Set ℓ₁}
           {B : Set ℓ₂} {C : Set ℓ₃} n
       → (B → C) → (A ^ n → B) → A ^ n → C
ccodom zero f g = f °′ g
ccodom (suc n) f g = λ x -> ccodom n f (g x)
```

Note that because of our decision on evading lifting we have to deal with "off by one" definitions.

If universe hierarchy of our base language were cumulative (or lifting were automatic), we would be able to give simpler definitions for the first (`zero`) cases to both of these and for every following function. On the other hand, repetition of subterms in definitions above shows relation between those two functions and function composition more directly.

Functions `_°_` (usual dependent function composition) and `_°′_` (its special form) could be used interchangeably in all our definitions from this section. Starting from the next listing we'll use `_°_` instead of `_°′_` for simplicity and ease of future abstractions.

Let us utter the programming way to describe transformations (for all $n > 0$) `cdom (n-1)` and `ccodom (n-1)` functions do:

- `cdom (n-1)` takes two functions `g` and `f` (first is one argument function, later has *n* arguments) and returns an *n*-ary function that applies `g` to every argument before passing it to `f`;
- `ccodom (n-1)` takes two functions `f` and `g` (as before, 1-ary and *n*-ary) and returns an *n*-ary function that applies its *n* arguments to `f` and then gives the result to `g`.

Next, we shall define a generalization of `flip` function. Conventional `flip` swaps the order of first two arguments of a function. Our generalization `cflip n`[1] flips the first argument of a given function with *(n+1)* following arguments. I.e. it transforms a function of type `E → A ^ n → B` into a function of type `A ^ n → (E → B)`.

```
cflip : ∀ {ℓ₁ ℓ₂ ℓ₃} {E : Set ℓ₁}
          {A : Set ℓ₂} {B : Set ℓ₃} n
```

[1] In "cdom" and "ccodom" prefix "c" stands for "change", here we would like to use prefix "g" for "generalized", but we've chosen to spell it as "counted" for uniformness.

```
      → (E → A ^ n → B) → (A ^ n → (E → B))
cflip zero f = λ x e -> f e x
cflip (suc n) f = λ x -> cflip n (λ e -> f e x)
```

Last base function combinator we need is the parallel composition, `cparcomp`, of two $(n+1)$-ary functions. I. e. a function that transforms two functions into one function operating on pairs.

```
cparcomp : ∀ {ℓ₁ ℓ₂ ℓ₃ ℓ₄}
             {A : Set ℓ₁} {B : Set ℓ₂}
             {A′ : Set ℓ₃} {B′ : Set ℓ₄} n
         → (A ^ n → B) → (A′ ^ n → B′)
         → (A × A′) ^ n → (B × B′)
cparcomp zero f g = λ p
  -> (f ° proj₁ $ p) ,′ (g ° proj₂ $ p)
cparcomp (suc n) f g = λ p
  -> cparcomp n (f ° proj₁ $ p) (g ° proj₂ $ p)
```

Type `A × B` is a direct product of types `A` and `B`, constructor `_,′_` is a constructor for `×`, $\texttt{proj}_1$ and $\texttt{proj}_2$ are the eleminators for `×`.

An eDSL Example

Suppose, we're writing a system which has a lot of logic in the subject domain. With `if`, `with` (`case` in *Haskell*) and *guard* constructions from usual functional programming languages, even dependent ones, there is no easy way to check that the given program is total, has no not contradictions and doesn't have dead branches in the presence of arbitrary logical expressions. In Turing-complete language like *Haskell* it's completely impossible in general case. With total languages like *Agda* it's hard to prove first two properties because every input should be examined case by case and it is impossible to prove the last one because there is no notion of reachability. In the end, programmer is forced to write either unchecked code or boiler-plate, that checks every possible input.

Note however, that if our subject domain allows us to restrict ourselves to `Bool`ean variables only, we can use any SAT-solver to check for totality, contradictions and dead branches explicitly. For such use cases we would like to have an eDSL with flat construction similar to `with` (or `case`) which allows us to write code like (pseudo-Agda):

```
example0 = select $
  X1 ·∧ X2      —> ...
  X1 ·∧ ·not X2 —> ...
  ·not X1       —> ...
```

To implement that, we shall define data type for logic expressions:

```
data Logic (V : Set) : Set where
  Latom : V → Logic V           -- Variable
  Ltrue Lfalse : Logic V
  Lnot : Logic V → Logic V
  Land Lor : Logic V → Logic V → Logic V
```

which could be used with arbitrary type of variables:

```
data Var : Set where
  X1 X2 X3 : Var

example1 : Logic Var
example1 = Lor
  (Land (Latom X1) (Latom X2))
  (Lor (Latom X3) (Lnot (Latom X1)))

example2 : Logic Var
example2 = (Land (Latom X1) (Latom X3))
```

Correctness checking

Amusingly, if we overlook horrible syntax, that simple definition is already usable for our purposes. Given *environment*, a function from variables to values, we could compute the value of a given logical expression:

```
compute : ∀ {V} → Logic V → (V → Bool) → Bool
compute (Latom v) e = e v
compute (Ltrue) _ = true
compute (Lfalse) _ = false
compute (Lnot x) e = not $ compute x e
compute (Land x1 x2) e =
  (compute x1 e) ∧ (compute x2 e)
compute (Lor x1 x2) e =
  (compute x1 e) ∨ (compute x2 e)
```

and check satisfability of a formula with simple SAT-solving algorithm[1]:

```
-- Boolean algebra of satisfying sets
SSet : Set → Set
```

[1] The idea of this algorithm is based on “sequent calculus” [9] derivation search algorithm. The original method is defined imperatively and can’t be directly translated into Agda program passing the termination checker. We reformulated it in terms of Boolean algebras and implemented in less efficient, but well-formed recursive way.

```
   SSet V = List (List V)

   -- "0" element
   szero : ∀ {V} → SSet V
   szero = []

   -- "1" element
   sone : ∀ {V} → SSet V
   sone = [ [] ]

   -- Lift from V to SSet
   ssingle : ∀ {V} → V → SSet V
   ssingle v = [ [ v ] ]

   -- Union
   _sor_ : ∀ {V} → SSet V → SSet V → SSet V
   x sor y = x ++ y

   -- Intersection
   _sand_ : ∀ {V} → SSet V → SSet V → SSet V
   x sand y = concatMap (λ xel -> map (λ yel -> (xel ++
yel)) y) x

   -- Now we define calculation of satisfying set in a
well-defined
   -- recursive manner.
   -- First Bool argument of a function: whenever ex-
pression we analize

   -- should be true of false.
   calc : ∀ {V} → Bool → Logic V → SSet (Bool × V)
   calc any (Latom x) = ssingle (any , x) --
   calc true   Ltrue = sone
   calc false  Ltrue = szero
   calc true  Lfalse = szero
   calc false Lfalse = sone
   calc any (Lnot x) = calc (not any) x
   calc true  (Land x1 x2) =
     (calc true x1)  sand (calc true x2)
   calc false (Land x1 x2) =
```

```
    (calc false x1) sor  (calc false x2)
  calc true  (Lor x1 x2) =
    (calc true x1)  sor  (calc true x2)
  calc false (Lor x1 x2) =
    (calc false x1) sand (calc false x2)

  -- Given satisfying set transform
  -- it into two sets:
  -- 1. variables that should be true
  -- 2. variables that should be false
  trans : ∀ {V : Set} → List (Bool × V)
        → List V × List V
  trans xs = go xs [] [] where
    go : ∀ {V} → List (Bool × V) → List V
       → List V → List V × List V
    go [] ts fs = ts , fs
    go ((true  , x) ∷ xs) ts fs = go xs (x ∷ ts) fs
    go ((false , x) ∷ xs) ts fs = go xs ts (x ∷ fs)

  private
    module Dummy {V : Set}
                 (_==_ : Decidable _≡_) where
      _===_ : V → V → Bool
      a === b with a == b
      ... | (yes _) = true
      ... | (no _) = false

      _elem_ : V → List V → Bool
      x elem ls = or $ map (_===_ x) ls

      _intersects_ : List V → List V → Bool
      ls intersects rs =
        or (map (flip _elem_ rs) ls)

      -- Given logical expression calc it and filter
      -- out contradictive satisfying sets (same
      -- variable should be true and false).
      solve : Logic V → List (List V × List V)
      solve = filter (not ° uncurry′ _intersects_)
            ° map trans ° calc true

  open Dummy public using (solve)
```

Now, given a list of pairs `Logic V × E` (`E` is arbitrary, `V` has decidable propositional equality), where type `E` is a type of result expressions, we could implement the following functions (they are trivial and we don't give their definitions for space-saving):

```
AreTotal : ∀ {ℓ} {E : Set ℓ} {V : Set}
         → List (Logic V × E) → Set
AreTotal = {!!}

AreSatisfyable : ∀ {ℓ} {V : Set} {E : Set ℓ}
               → List (Logic V × E) → Set
AreSatisfyable = {!!}

AreNonContradictional : ∀ {ℓ} {V : Set}{E : Set ℓ}
                      → List (Logic V × E) → Set
AreNonContradictional = {!!}


select : ∀ {ℓ} {V : Set} {E : Set ℓ}
  → (l : List (Logic V × E))
  → {_ : AreTotal l} → {_ : AreSatisfyable l}
  → {_ : AreNonContradictional l}
  → (V → Bool) → E
select = {!!}
```

`AreTotal` checks that a disjunction of all first elements of pairs of a list is a tautology (negation is not solvable). If it is a tautology, `AreTotal` returns ⊤ (the type with only one element), otherwise it returns ⊥ (the type without elements). For ⊤ (top) compiler infers the only element and substitutes it, for ⊥ (bottom) it generates an unsolved constraint. Moreover, we could return not just the bottom type, but a type isomorphic to bottom, which has an error description packed into its term, if we wish. This description would be then printed in an unsolved constraint message generated by the compiler. Respectively, `AreSatisfyable` checks that every element could be satisfied (no dead branches) and `isNonContradictional` checks that a pairwise conjunction of arbitrary two `Logic V` expressions is not satisfiable (each input defines exactly one output). `select` linearly scans through the list and returns the second element of the first (and the only, by the definition) pair which has `true` as a result of `compute`.

We might also consider sugaring the syntax for defining lists of these pairs. To obtain a syntax similar to the described above, it's enough to define a new constructor for pairs:

```
infix 10 _—>_
_—>_ : ∀ {ℓ} {V : Set} {E : Set ℓ} → Logic V
     → E → Logic V × E
a —> b = a ,′ b
```

Example usage:

```
exampleDef = select $
  example1 —> 0 ::
  example2 —> 1 ::
  []
```

This would look scary if we substitute the definitions of `example1` and `example2`, but it already checks all the desired properties at compile time.

Syntax sugar

In order to rewrite logical expressions with the following desired syntax sugar:

```
example1′ : Logic Var
example1′ = X1 ·∧ X2 ·∨ X3 ·∨ ·not X1

example2′ : Logic Var
example2′ = ·not X1 ·∧ X3
```

we should write infix functions `·∧`, `·∨` and `·not` which are able to receive arguments of different types. For example, in the code above `·∧` receives the value of type `Var` at the first occurrence and the value of type `Logic Var` at the second.

In Haskell this problem could easily be solved by typeclasses. In Agda, a type class becomes a dependent tuple of elements `A : Set` and `A × I A`, i.e. the type `A`, the element of this type and the implementation of the interface `I` for this type. With that knowledge we could define needed functions right away, but let us check what else we don't like in our definitions first.

Clearly, we don't like our definitions around `select` because they interpret logical expressions with `compute`, also we don't like `select` that makes a linear search. For the later problem we don't know any simple solutions. For the former problem we could do better by lifting native Agda's or Haskell's `Boolean` functions into functions of type `(V → Bool) → Bool`.

For this purpose we need to define the interface:

```
record Liftable (V : Set) (F : Set) : Set where
  field
    llift : F → Logic V
    flift : F → (V → Bool) → Bool
```

`llift` lifts (transforms) F into its `Logic`al representation, `flift` lifts `F` into a function from any environment to `Bool`.

We still wish to analyze our expressions, thus we shall store their logical and functional representations at the same time:

```
AlreadyLifted : Set → Set
AlreadyLifted V = (Logic V) × ((V → Bool) → Bool)
```

We are now able to define our type class in Agda:

```
ELiftable : Set → Set → Set
ELiftable V L = L × Liftable V L

CLiftable : Set → Set1
CLiftable V = Σ Set (ELiftable V)

llift : ∀ {V} → CLiftable V  → Logic V
llift (t , a) = uncurry (flip $
  Liftable.llift {F = t}) a

flift : ∀ {V} → CLiftable V → (V → Bool) → Bool
flift (t , a) = uncurry (flip $
  Liftable.flift {F = t}) a
```

and the default implementations:

```
mkAnyLiftable : ∀ V → Liftable V V
mkAnyLiftable _ = record
  { llift = Latom
  ; flift = λ v e → e v }

mkLogicLiftable : ∀ V → Liftable V (Logic V)
mkLogicLiftable _ = record
  { llift = id
  ; flift = compute }

mkAlreadyLiftable : ∀ V
                  → Liftable V (AlreadyLifted V)
mkAlreadyLiftable _ = record
  { llift = proj₁
  ; flift = proj₂ }
```

Lifting $(n+1)$-ary Boolean function `f` into a function from an environment to `Bool` is simple: given an environment `e`, we need to lift every argument into a

function, apply `e` to each of them, and then apply results to `f`. That is exactly what `cdom` function does. After this transformation we end up with a function of the type `(V → Bool) → A → ⋯ → A → Bool` (where each `A` is a specialization of `CLiftable`).

`cflip`ping will convert it into a function with the desired type `⋯ → (V → Bool) → Bool`:

```
lift2Func : ∀ {V} n → (Bool ^ n → Bool)
          → (CLiftable V) ^ n → ((V → Bool) → Bool)
lift2Func n f = cflip n
  (\e -> cdom n (flip flift e) f)
```

Lifting into `Logic`al expression is a bit more complicated. For a single argument function we'll write the resulting logic expression explicitly. Functions of bigger arity are obtained by:

- abstracting by the first argument of the resulting expression (`a`),
- fixing first argument of `f`, lifting resulting expression, then `cflip`ping (to move `Bool` argument behind arguments of the lifted version of a function),
- applying all other arguments (here we end up with a function of the type `Bool → Logic V`)
- and then constructing resulting expression from two possible values of `a`.

The last two steps are exactly what `ccodom` function does. The resulting expression is defined by:

```
lift2Logic : ∀ {V} n → (Bool ^ n → Bool)
           → (CLiftable V) ^ n → Logic V
lift2Logic zero f = repr f where
  repr : ∀ {V} → (Bool → Bool) → CLiftable V →
Logic V
  repr f a with f true | f false
  ... | true  | true  = Ltrue
  ... | true  | false = llift a
  ... | false | true  = Lnot $ llift a
  ... | false | false = Lfalse
lift2Logic (suc n) f = λ a →
  ccodom n (lift2L a) ° cflip n $
 λ x -> lift2Logic n (f x) where
  lift2L : ∀ {V} → CLiftable V
         → (Bool → Logic V) → Logic V
  lift2L a liftN = Lor
    (Land (llift a)         (liftN true))
    (Land (Lnot $ llift a) (liftN false))
```

Note again, that this definition would be less complicated if we had cumulative universe hierarchy or automatic lifting between universes.

Now, lifting any Boolean function to `AlreadyLifted` is just a parallel composition of `lift2Logic` and `lift2Func`:

```
lift2Already : ∀ {V} n → (Bool ^ n → Bool)
             → (CLiftable V) ^ n → AlreadyLifted V
lift2Already n f = cdom n (λ x → (x ,' x)) $
  cparcomp n (lift2Logic n f) (lift2Func n f)
```

That `lift2Already` still won't give us our desired syntax because we'll have to wrap every value into `CLiftable` by hand. Implicit arguments and instance arguments should solve this problem, but there is no way to use them with our current type scheme.

Therefore, we shall define a type scheme for functions which take $(n+1)$ arguments of arbitrary type, where each of them has an implementation of interface somewhere in the context:

```
_^I_→_ : ∀ {ℓ₁ ℓ₂ ℓ₃} → (Set ℓ₁ → Set ℓ₂)
       → N → Set ℓ₃ → Set (Level.suc ℓ₁ ⊔ ℓ₂ ⊔ ℓ₃)
_^I_→_ {ℓ₁} I zero B =
  {A : Set ℓ₁} → (a : A) → {{i : I A}} → B
_^I_→_ {ℓ₁} I (suc n) B =
  {A : Set ℓ₁} → (a : A) → {{i : I A}}
  → (I ^I n → B)
```

We are able to define a definition similar to `cdom` to adapt this type scheme to our previous type scheme:

```
cidom : ∀ {ℓ₁ ℓ₂ ℓ₃ ℓ₄} {B : Set ℓ₂}
          {C : Set ℓ₃} {I : Set ℓ₁ → Set ℓ₄}
      → (n : N) → (Σ (Set ℓ₁) (λ A → A × I A) → B)
      → (B ^ n → C) → (I ^I n → C)
cidom zero g f = λ {A} a {{I}}
  -> f ° g $ (A , (a , i))
cidom (suc n) g f = λ {A} a {{i}}
  -> cidom n g $ f ° g $ (A , (a , i))
```

We could define a backward transformation in a similar way, but we shall skip this definition for space-saving. From a philosophical standpoint of view this bijection means that our type scheme is orthogonal to type system extensions which infer elements from a context. It's even orthogonal to obvious extensions which generate types like `A → B → A → B → ⋯ → C` because we can pack them into tuples in the same way.

We are now able to define `lift` function with a desired syntax:

```
NLiftable : Set → N → Set1
NLiftable V n = (Liftable V) ^I n
  → (AlreadyLifted V)

lift : ∀ {V} n → (Bool ^ n → Bool) → NLiftable V n
lift n f = cidom n id (lift2Already n f)
```

and test it:

```
infixl 80 _·∧_
_·∧_ : ∀ {V} → NLiftable V 1
_·∧_ = lift 1 _∧_

infixl 70 _·∨_
_·∨_ : ∀ {V} → NLiftable V 1
_·∨_ = lift 1 _∨_

·not : ∀ {V} → NLiftable V 0
·not  = lift 0 not

-- Everything before, except Var data type
-- definition, should be considered a library
-- code.
-- Everything from here and till the end of this
-- listing (and Var) is that a user of
-- this library might write.

-- We have to do the following for every type
-- like Var
-- because Agda doesn't specialize functions which
-- generate instances for instance arguments,
-- even if arguments are implicit.

tmpx = mkAnyLiftable Var
tmpy = mkLogicLiftable Var
tmpz = mkAlreadyLiftable Var

example1' : AlreadyLifted Var
example1' = X1 ·∧ X2 ·∨ X3 ·∨ ·not X1

test1 : proj₁ (X1 ·∧ X2) ≡ Lor
  (Land (Latom X1) (Latom X2))
```

```
    (Land (Lnot (Latom X1)) Lfalse)
test1 = refl

example1'env : Var → Bool
example1'env x with x
... | X1 = false
... | X2 = true
... | X3 = false

test2 : proj₂ example1' example1'env ≡ true
test2 = refl
```

Conclusion

This work has provided a simple explicit handling for polyvariadic functions in dependently typed setting. The combinators presented here make it possible to describe them with expressions in a point-free style. The example usage of the combinators is shown on the moderately sized example of logical eDSL.

The latter shows that presented approach covers most usages of conventional template engines: our `lift` function simultaneously generates `Logic`al expressions that are to be checked at compile time and the code to be executed at runtime.

The type scheme for dependent polyvariadic functions can be straightforwardly enhanced by type indexing, which allows to describe even more rich set of functions. Research in this area would be an important next step.

References

1. *Hudak P., Peterson J., Fasel J.* A Gentle Introduction To Haskell, Version 98. 2000. http://www.haskell.org/tutorial/ [date: 07.04.2012].
2. *Stroustrup B.* The C++ Programming Language (Third ed.). Boston: Addison–Wesley, 1997.
3. Python Programming Language. http://www.python.org/ [date: 07.04.2012].
4. Template Haskell. http://www.haskell.org/haskellwiki/Template_Haskell [date: 07.04.2012].
5. *Chapman J., Dagand P., Mcbride C., Morris P.* The Gentle Art of Levitation // ICFP '10 Proceedings of the 15th ACM SIGPLAN international conference on Functional programming. 2010. P. 3–14.
6. *Kiselov O.* Polyvariadic functions and keyword arguments. http://okmij.org/ftp/Haskell/polyvariadic.html [date: 07.04.2012].
7. C++0x/C++11 Support in GCC. http://gcc.gnu.org/projects/cxx0x.html [date: 07.04.2012].
8. Agda. http://wiki.portal.chalmers.se/agda/pmwiki.php [date: 07.04.2012].
9. *Sørensen M. H. B., Urzyczyn P.* Lectures on the Curry-Howard Isomorphism. Elsevier Science, 2006.

ГЕНЕРАЦИЯ КОНЕЧНЫХ АВТОМАТОВ С ПОМОЩЬЮ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ РЕШЕНИЯ ЗАДАЧИ О ПОИСКЕ ЦЕЛИ СЕНСОРНЫМ АГЕНТОМ В ОБЛАСТИ С ПРЕПЯТСТВИЯМИ

А. А. Соколов

студент кафедры компьютерных технологий;
ansokolmail@gmail.com

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Аннотация: В данной работе рассматривается возможность применения генетического алгоритма для генерации конечных автоматов, которые можно использовать для решения простейшего варианта задачи навигации.

Введение

Автоматное программирование — это парадигма программирования, в которой поведение программы может быть представлено в виде системы управляющих конечных автоматов [5]. Автоматный подход применяется в событийно-ориентированных приложениях. Множество ответственных технических систем являются событийно-ориентированными, а управление в них является автоматным.

Задачи навигации возникают в различных отраслях современной науки. Так, на практике часто возникают задачи о перемещении робота из одной точки в другую в различном пространстве конфигураций — от простейшего случая передвижения робота на плоскости среди препятствий до поиска оптимальной стратегии перемещения в пространстве робота-манипулятора со множеством степеней свободы.

Среди всего множества задач навигации обычно рассматривают некоторые крайние случаи, например, обладание полной информацией о местности, или, наоборот, отсутствие возможности запоминать что-либо, кроме нескольких чисел. В реальности эти случаи комбинируются, и системе управления роботом необходимо использовать доступную ему информацию наиболее эффективным способом, одновременно корректно обрабатывая нештатные ситуации. Сложность требований, предъявляемых к системе управления мобильным роботом, очень высока. Использование автоматного программирования, по крайней мере, в некоторых частях системы управления позволит уменьшить число ошибок, что приведет к повышению надежности системы. По этой причине необходимо исследовать возможность реализации алгоритмов, решающих задачи навигации, автоматными методами. Принимая

в расчет сложность этих алгоритмов, авторы данного исследования считают целесообразным исследовать автоматическое построение автоматов, решающих некоторые задачи навигации.

Рассмотрим следующую задачу навигации. Имеется некоторая двумерная область с препятствиями. Препятствия могут быть произвольной формы, однако никакие два препятствия не должны иметь общих точек. Кроме того, любой круг конечного радиуса пересекает конечное число препятствий, и любое препятствие покрывается кругом конечного радиуса.

В области, удовлетворяющей описанным требованиям, дана точка — *цель*. Агент в начальный момент времени находится в произвольной точке области, не принадлежащей ни одному препятствию. Задача агента заключается в том, чтобы добраться до цели или, если она недостижима, сообщить об этом. При этом агент знает свои координаты и координаты цели. Агент не знает, где именно находятся препятствия, но может определить, если ли препятствия в непосредственной с ним близости. Агент обладает O(1) дополнительной памяти, по этой причине он не может запоминать уже посещенную часть области, но может хранить некоторый заранее определенный объем информации (например, координаты некоторого числа точек).

Известны алгоритмы, решающие данную задачу. К ним относятся алгоритмы семейства *Bug*. Общая идея этих алгоритмов такова: пока агент может двигаться по направлению к цели, он это делает. Если агент находит препятствие, которое не дает ему двигаться, он начинает обход этого препятствия. Обход прекращается и продолжается движение к цели, когда выполнено какое-либо условие, специфичное для конкретного алгоритма. В процессе этого обхода алгоритм также может прийти к выводу, что цель недостижима.

Первые алгоритмы семейства *Bug* (*Bug1*, *Bug2*) были впервые описаны в работе [1]. Известными алгоритмами этого семейства также являются *Distant Bug* и *Tangent Bug* [2–4].

Постановка задачи

В данной работе исследуется возможность автоматического нахождения автоматной реализации алгоритмов, решающих поставленную задачу. Однако, при использовании автоматов более естественной является постановка задачи с конечным числом дискретных входных воздействий, поэтому авторы работы полагают, что для упрощеня работы имеет смысл рассмотреть вариацию данной задачи, при которой возможно описать такие входные воздействия.

Областью, на которой действует агент, будем считать бесконечное клетчатое поле. Некоторые клетки этой области заняты препятствиями, причем таких клеток конечное число. Цель находится в одной из клеток, не занятых препятствием. Агент также занимает одну клетку и может перемещаться только в клетки, смежные по стороне с текущей.

Агент управляется конечным автоматом — входные воздействия для этого автомата формируются на основе состояния агента и наблюдаемого им состояния среды, выходные воздействия автомата управляют движением агента. Цель работы — вырастить конечный автомат с помощью генетического алгоритма **[6]**, который бы управлял агентом таким образом, чтобы, если цель достижима из начального положения агента, агент за конечное число шагов достиг бы этой цели и, находясь там, сообщил бы о завершении работы. Если же цель недостижима, то агент за конечное время должен это определить и завершить работу, сообщив о том, что цель недостижима.

Объект управления

Агент, являющийся объектом управления, находится на поле, занимая одну клетку. Помимо координат на поле, у агента также есть ориентация — в каждый момент времени он направлен в одну из сторон («север», «запад», «юг», «восток»). Внешне наблюдаемые действия агента ограничиваются следующими:

- перейти «вперед» (в ту сторону, в которую он направлен) на соседнюю клетку;
- повернуться по часовой стрелке;
- повернуться против часовой стрелки;
- закончить работу при достижении цели;
- закончить работу при выяснении, что цель недостижима.

По причинам того, что столкновения с препятствиями «запрещены», для анализа окружающей его среды агент должен уметь непосредственно узнавать, есть ли препятствия в некоторых клетках в его непосредственной окрестности. В данной работе агент способен узнать, есть ли препятствие непосредственно перед ним.

Перечислим все величины, хранящиеся в агенте (включая константные величины и величины, получаемые «от датчиков»):

- абсцисса и ордината цели;
- абсцисса, ордината и ориентация текущего положения агента;
- абсцисса, ордината и ориентация сохраненного положения агента;
- есть ли препятствие в клетке, находящейся непосредственно перед агентом.

Первый, второй и четвертый набор величин в совокупности образуют наблюдаемое агентом состояние среды. Третий же набор образует вычислительное состояние агента.

Так же в работе рассматривается задача с расширенным набором данных, которые агент может дополнительно получить от датчиков следующие величины:

- есть ли препятствие в клетке, находящейся слева от агента;
- есть ли препятствие в клетке находящейся справа от агента.

Система управления

В данной работе система управления агентом представлена конечным автоматом (автоматом Мили). Входные воздействия автомата, непосредственно получающиеся из условий задачи, приведены в табл. 1. Авторы работы также рассмотрели расширенный набор, дополнительно включающий следующие входные воздействия (табл. 2).

Таблица 1

Входные воздействия

Метка	Название в коде	Описание
x1	CAN_MOVE_FORWARD	Отсутствует ли перед агентом препятствие
x2	IS_MOVE_FORWARD_COOL	Приблизится ли агент к цели, если сделает шаг вперед
x3	IS_AT_FINISH	Находится ли агент на финише
x4	IS_AT_SAVED	Находится ли агент в сохраненной точке
x5	IS_BETTER_THAN_SAVED	Ближе ли текущая клетка к финишу, чем сохраненная

Таблица 2

Расширение набора входных воздействий

Метка	Название в коде	Описание
x6	CAN_MOVE_LEFT	Отсутствует ли слева от агента препятствие
x7	IS_MOVE_LEFT_COOL	Приблизится ли агент к цели, если перейдет на клетку соседнюю слева
x8	CAN_MOVE_RIGHT	Отсутствует ли справа от агента препятствие
x9	IS_MOVE_RIGHT_COOL	Приблизится ли агент к цели, если перейдет на клетку соседнюю справа

Выходные воздействия автомата приведены в табл. 3.

Таблица 3

Выходные воздействия

Метка	Название в коде	Описание
z1	DO_NOTHING	Ничего не делать
z2	MOVE_FORWARD	Сделать шаг «вперед»
z3	ROTATE_POSITIVE	Повернуться по часовой стрелке
z4	ROTATE_NEGATIVE	Повернуться против часовой стрелки
z5	SAVE_POSITION	Сохранить текущее расположение
z6	REPORT_REACHED	Выдать сообщение о прибытии на финиш
z7	REPORT_UNREACHABLE	Цель недостижима

В качестве представления состояний и переходов автомата используются деревья решений **[7]**. В листьях деревьев решений находятся пары из выходного действия и номера состояния, в которое нужно осуществить переход. Внутренние узлы соответствуют входным воздействиям, у каждого узла две потомка, один для случая, если соответствующее узлу входное воздействие имеет значение true, другой для значения false.

Использовался островной генетический алгоритм — это такой параллельный генетический алгоритм, в котором популяция разбивается на несколько подпопуляций. Для каждой из них параллельно запускается генетический алгоритм. Каждые несколько поколений процессы обмениваются между собой лучшими особями.

Результаты

Для первого набора предикатов произведено 800 запусков генетического алгоритма. В каждом запуске был найден корректный автомат, решающий задачу. Минимальное число поколений, за которое был найден автомат, равнялось 63, максимальное — 82 598. Среднее время вычисления составило 3.75 часа, причем минимальное время вычисления равнялось 1.5 минутам, а максимальное — 47 часам. Для расширенного набора предикатов было проведено 20 запусков. В каждом из запусков также был найден корректный автомат, решающий задачу. Число поколений, необходимых для этого, варьировалось от 4996 до 290 983. Время работы каждого запуска принимало значение от 12 до 80 часов.

Заключение

В ходе выполнения работы было показано, что возможно получать алгоритмы для решения задач поиска пути, реализованные в виде управляющего конечного автомата, с помощью генетического алгоритма. Получены автоматы, которые корректно решают поставленную задачу на любом заданном поле. Таким образом, генетический алгоритм был успешно применен для данной задачи.

Литература

1. *Lumelsky V. J., Stepanov A. A.* Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica, 2: 403–430, 1987.
2. *Lumelsky V. J., Skewis T.* Incorporating range sensing in the robot navigation function. IEEE Transactions on Systems, Man, and Cybernetics, 20(5): 1058–1068, 1990.
3. *Kamon I, Rimon E, Rivlin E.* A New Range-Sensor Based Globally Convergent Navigation Algorithm for Mobile Robots. CIS — Center of Intelligent Systems 9517, Computer Science Dept., Technion, Israel, 1995.

4. *Liu Y. H., Arimoto S.* Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. International Journal of Robotic Research, 11 (4): 376–382, 1992.
5. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2010.
6. *Holland John H.* Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor. 1975
7. *Данилов В. Р.* Технология генетического программирования для генерации автоматов управления со сложным поведением. СПбГУ ИТМО, 2007. Бакалаврская работа. http://is.ifmo.ru/papers/danilov_bachelor

ПРИМЕНЕНИЕ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ УДОВЛЕТВОРЕНИЯ ОГРАНИЧЕНИЙ ДЛЯ ПОСТРОЕНИЯ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ ПО СЦЕНАРИЯМ РАБОТЫ

В. И. Ульянцев, Ф. Н. Царев

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Введение

При применении парадигмы автоматного программирования для реализации сущности со сложным поведением выделяется система управления и объект управления. На начальном этапе проектирования программы выделяются события, входные переменные и выходные воздействия. После этого проектирование программы может идти разными путями. Один из них состоит в написании сценария работы программы, по которому далее эвристически строится автомат. Пример построения автомата таким способом приведен в работе [1]

К автоматной программе, как правило, предъявляются два требования:

- непротиворечивость — не должно быть двух переходов, исходящих из одного состояния управляющего автомата и одновременно выполнимых при некоторой комбинации события и входных переменных;
- полнота — любой комбинации события и входных переменных должен соответствовать переход в каждом состоянии.

Ранее авторами был предложен метод построения автоматных программ, удовлетворяющих требованию непротиворечивости, но не удовлетворяющих требованию полноты. В настоящей работе предлагается метод, не обладающий данным недостатком.

Цель работы

В указанной работе сначала был составлен неформальный сценарий в виде текста, далее он был формализован — был получен текст с указанием обозначений входных событий и выходных воздействий, соответствующих стадиям работы программы. После этого выполнялось построение автомата. Все три указанных этапа выполнялись вручную.

Первые два этапа, скорее всего, не могут быть автоматизированы и должны выполняться человеком. Ранее авторами был разработан метод автоматизированного построения управляющих конечных автоматов по сценариям работы [2]. Целью настоящей работы является разработка метода построения

управляющих автоматов с гарантией не только непротиворечивости системы переходов, но и ее полноты.

Базовые положения исследования

Сценарием работы программы будем называть последовательность троек $<e, f, A>$, где e — входное событие, f — булева формула от входных переменных, задающая охранное условие, A — последовательность выходных воздействий. На вход разрабатываемому алгоритму подается набор сценариев работы. Построение управляющего автомата осуществляется в пять этапов:

1. Построение дерева сценариев.
2. Построение графа совместимости вершин дерева сценариев.
3. Построение ограничений на целочисленные переменные, задающей требования к раскраске построенного графа и выражающей непротиворечивость и полноту системы переходов результирующего автомата.
4. Запуск сторонней программы, решающей задачу удовлетворения построенных ограничений (constraint satisfaction problem, CSP).
5. Построение автомата по найденной выполняющей подстановке.

Основной результат

Разработан метод автоматизированного построения управляющих конечных автоматов по сценариям работы. Этот метод основан на сведении указанной задачи к задаче удовлетворения ограничений. Работоспособность метода проверена на задаче построения автомата управления часами с будильником. На этой задаче соответствующий управляющий автомат был построен корректно, а время работы алгоритма составляло меньше секунды на персональном компьютере с процессором Intel Core 2 Quad

Q9400 (2,67 ГГц), что позволяет говорить о достаточно высокой производительности разработанного метода.

Литература

1. *Мазин М. А., Парфенов В. Г., Шалыто А. А.* Разработка интерактивных приложений Macromedia Flash на базе автоматной технологии. http://is.ifmo.ru/download/flash.pdf
2. *Ulyantsev V., Tsarev F.* Extended Finite-State Machine Induction using SAT-Solver // Proceedings of the Tenth International Conference on Machine Learning and Applications, ICMLA 2011, Honolulu, HI, USA, 18–21 December 2011. IEEE Computer Society, 2011. Vol. 2. P. 346–349.

МЕТОД ПОВЫШЕНИЯ КАЧЕСТВА ВЕБ-ПРИЛОЖЕНИЙ НА ОСНОВЕ АВТОМАТНОГО ПОДХОДА

А. Ю. Законов

аспирант кафедры компьютерных технологии;
andrew.zakonov@gmail.com

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Аннотация: В течение последних лет существенно выросла как сложность веб-приложений, так и требования к их надежности. Зачастую по причине сжатых сроков разработки и часто меняющихся требований заказчика проверка качества веб-приложений осуществляется вручную без формальных критериев. Отсутствие моделей и формальной спецификации делает процесс проверки качества трудоемким и не позволяет гарантировать отсутствие ошибок. В статье предложен метод автоматического выделения явных состояний и построения автоматной модели для веб-приложений при помощи динамического анализа. Наличие автоматной модели позволит формализовать требования спецификации и использовать Model Checking для автоматизации процесса тестирования. Также разработан алгоритм поиска схожих состояний, благодаря которому даже для сложных веб-приложений, возможно автоматически строить модели с небольшим количеством состояний, которые будут понятны разработчику.

Введение

Сложность веб-приложений стремительно растет так же, как и требования к их надежности и безопасности. Вопрос проверки качества веб-приложений является актуальным, так как множество программ реализуются в виде веб-приложений и многие из них связаны с финансовыми транзакциями и конфиденциальной информацией: интернет-магазины (Amazon.com), интернет-банкинг, почтовые клиенты (GMail), социальные сети (Facebook), системы документооборота (Alfresco) и многие другие.

Веб-приложение — клиент-серверное приложение, в котором клиентом выступает браузер, а сервером — веб-сервер. Логика веб-приложения распределена между сервером и клиентом, обмен информацией происходит по сети [1]. Тестирование веб-приложений так же, как и тестирование пользовательских интерфейсов, тяжело автоматизировать, так как это требует эмуляции действий пользователя, таких как ввод текста, заполнение форм, переходы по ссылкам. Для сложных веб-приложений различных последовательностей действий пользователя может быть бесконечно много, поэтому проверить все варианты взаимодействия пользователя с системой невоз-

можно. На практике зачастую проверяется только небольшое подмножество возможных вариантов действий пользователя, что оставляет возможность появления разнообразных критических ошибок.

В статье предлагается метод автоматизации тестирования веб-приложений, основанный на автоматном подходе. Конечные автоматы позволяют удобно описывать взаимодействие системы с пользователем [2, 3]. Состояние веб-приложения — веб-страница, которую видит пользователь. Входные воздействия, которые могут менять состояние приложения, — это действия пользователя и ответы сервера на AJAX-запросы. Состояние приложения может меняться за счет перехода на новую веб-страницу или путем динамического изменения структуры (DOM-дерева) текущей страницы.

Исходный код приложения является наиболее точным описанием логики работы, но это представление неудобно для тестирования, так как не позволяет понять высокоуровневую логику приложения. Для этой цели в статье предложен метод автоматического построения автоматных моделей существующих веб-приложений. Автоматная модель обладает следующими преимуществами:

- описывает логику работы приложения в визуально понятном и читаемом формате;
- позволяет удобно записать требования спецификации, используя формальную логику, например, язык LTL, и автоматически проверить их, используя Model Checking;
- пригодна автоматической генерации тестов с заданными критериями (покрытие всех переходов/состояний).

Цель исследования: предложить метод автоматического построения автоматной модели веб-приложения, метод применения Model Checking и автоматизации тестирования для повышения качества веб-приложений.

В статье решаются следующие задачи:

1. Разработать метод автоматического выделения явных состояний и переходов между ними для веб-приложений.
2. Оценить применимость методов Model Checking для автоматных моделей веб-приложений и предложить соответствующие инструменты для верификации построенных моделей.
3. Предложить метод автоматической генерации тестов по построенной модели.

II раздел статьи содержит описание существующих методов проверки качества веб-приложений, их преимущества и недостатки. III раздел описывает предложенный подход к построению автоматной модели веб-приложений. В IV разделе подход проиллюстрирован на примерах существующих популярных веб-приложений.

Обзор существующих решений

В работе [4] авторы исследуют 24 различных метода тестирования и верификации веб-сайтов. В большинстве подходов предлагается начинать разработку с построения модели или строить модель вручную [5, 6, 7]. В работе [8] авторы предлагают вручную создать модель приложения и применять Model Checking для проверки свойств этой модели. Эти подходы не применимы к существующим сложным веб-приложениям, так как построение модели вручную трудоемко и не может гарантировать соответствие модели коду приложения. Более того, при изменениях кода приложения придется вручную обновлять модель.

В статье [9] описан метод построения конечного автомата, который будет описывать заданное свойство веб-приложения. Полученные модели могут состоять из тысяч состояний и не могут быть использованы как понятное описание веб-приложения. В предложенном в данной работе подходе предлагается строить модель полностью автоматически для существующего приложения и затем, вручную проанализировав модель, описывать свойства веб-приложения, для проверки которых можно будет использовать Model Checking.

Попытка автоматизировать верификацию и построение моделей предпринята в работе [10], но со значительными ограничениями: поддерживаются только приложения, разработанные с использованием фреймворка Struts и технологии Java Server Page templates, и не поддерживаются AJAX-приложения. Предложенный в данной работе подход применим к значительно более широкому кругу веб-приложений и не накладывает ограничения на используемые технологии. В том числе принципиально важна поддержка технологии AJAX, которая повсеместно используется для создания динамических и интерактивных приложений.

Данная работа описывает подход, который развивает и обладает преимуществами по сравнению с существующими подходами, в том числе поддерживает одновременно как AJAX-приложения, так и переходы между страницами по гиперссылкам. Предложенный метод построения автоматной модели позволяет автоматически строить модель всего веб-приложения без ограничений. Полученная модель позволит описывать требования спецификации ко всему приложению в целом и автоматизировать их тестирование и проверку методом Model Checking.

Автоматическое построение модели веб-приложения

Под состоянием веб-приложения будем понимать исходный код веб-страницы, на которой находится пользователь. События — действия пользователя, которые включают в себя нажатия на кнопки и прочие управляющие элементы, ввод текста и заполнение форм.

Алгоритм обнаружения возможных состояний веб-приложения

Для автоматического построения модели веб-приложения необходимо обнаружить максимальное количество возможных состояний приложения. Данная задача неразрешима в общем случае, так как различных комбинаций действий пользователя может быть бесконечно много. В данной работе предложен алгоритм обнаружения новых состояний, основанный на случайной генерации действий пользователя:

1. Исходный код страницы анализируется и составляется список всех возможных действий пользователя actionlist, на которые реагирует данная страница приложения.
2. Выбирается случайное действие из списка возможных действий actionlist и выполняется программой, которая эмулирует действия пользователя.
3. Все действия запоминаются в виде троек *<state1,action,state2>*, где state1 — состояние приложения до выполнения действия, state2 — состояние приложения после выполнения действия, action — обозначение выполненного действия.
4. Перейти к шагу 1, если за последние *N* итераций было обнаружено хотя бы одно новое событие, не встречавшееся ранее (*N* — подбирается под конкретную задачу). Данное условие останова будет срабатывать в том случае, если все состояния приложения были обнаружены или если приложение попало в некоторое тупиковое состояние, которое не позволяет перейти к новым состояниям, а только возвращаться в предыдущие.

Предложенный алгоритм не может гарантировать обнаружение всех возможных состояний и переходов между ними, но при долгом времени работы может быть получена достаточно точная модель веб-приложения. Недостаток полученной таким образом модели заключается в том, что она может состоять из сотен и тысяч различных состояний и разработчик не сможет с ней работать. Для преодоления этого недостатка и построения читаемой модели предложен метод упрощения полученной модели.

Алгоритм упрощения модели методом объединения похожих состояний

Решить проблему излишне большого количества состояний можно путем объединения похожих состояний. Под состоянием мы подразумеваем исходный HTML-код страницы. Будем работать с HTML-кодом как с DOM-деревом и введем следующее рекурсивное определение похожести: вершины DOM-дерева *A* и *B* похожи, то есть *similar(A,B)==True* если и только если они одного типа, имеют одинаковый набор атрибутов и одинаковое количество подвершин, где каждая подвершина *A* похожа с соответствующей подвершиной *B*.

При сравнении вершин текстовые значения вершин игнорируются, сравниваются только структуры DOM-деревьев. Например, вершины text и othertext будут признаны похожими. Также при сравнении двух DOM-деревьев выполняется два дополнительных шага: фильтрация вершин по их типу и сворачивание деревьев. Те вершины, которые не влияют непосредственно на поведение веб-страницы и обработку действий пользователя, отфильтровываются: link, script, meta и другие. Сворачивание дерева позволяет считать похожими страницы, которые отличаются только количеством однотипных элементов, но не отличаются своим поведением. Примером таких веб-страниц может быть список писем в почтовом ящике или список результатов поиска. Список DOM-вершин x_1, ..., x_n будет свернут, если $\forall i, j \in [1; n]$ similar(x_i, x_j) == True && x_i.parent == x_j.parent. В этом случае список вершин будет заменен на список из одной вершины x_1. Сворачивание выполняется следующим алгоритмом:

1. Выполнять обход DOM-дерева, начиная с листьев.
2. Для данной вершины составить список подвершин list_c.
3. Сравнить все возможные пары $x_i, x_j \in \text{list}_c$ и, если similar(x_i, x_j) == True, удалить подвершину x_j.

Два состояния веб-приложения будут похожими, если корневые вершины соответствующих DOM-деревьев, после шагов фильтрации и сворачивания будут похожими.

Пример использования

Для апробации предложенного подхода разработан набор инструментов на языке Python 2.7. Для работы с веб-страницами и эмуляции действий пользователя используется фреймворк Selenium [11]. Для укладки графа и представления автоматной модели в виде PNG-изображения используется фреймворк GraphViz. Разработанный в рамках данного исследования инструмент для заданного веб-приложения автоматически создает модель в виде конечного автомата, который описывает обнаруженные состояния приложения и возможные переходы между ними. Метки на состояниях отражают заголовки страниц, а метки на переходах действия, которые приводят к смене состояний, записанные в формате: «нажмите на объект L» или «введите текст A в поле B». Указания на объекты внутри страницы записаны на языке XPath.

На рис. 1 приведены примеры результата работы инструмента для веб-приложений TadaList.com и m.VK.com. Для TadaList за 9 минут работы инструментом было выполнено 250 случайных действий на странице приложений, обнаружено 216 различных состояний и переходов между ними. После применения алгоритма поиска и слияния похожих состояний была получена модель, состоящая из 15 состояний. Для m.VK.com за 6 минут было выполнено 200 случайных действий, что позволило обнаружить 170 различных состояний и переходов между ними. Алгоритм поиска и слияния похожих состояний позволил упростить модель до 19 состояний.

Рис. 1. Автоматически построенные модели веб-приложений

Заключение

Проверка качества веб-приложений является сложным и трудоемким процессом, так как приложения предполагают активное взаимодействие с пользователем. Вариантов различных последовательностей действий пользователя может быть неограниченно много для сложных приложений, поэтому нет возможности проверить все приложение вцелом. В работе сделана попытка автоматизировать процесс проверки качества веб-приложений.

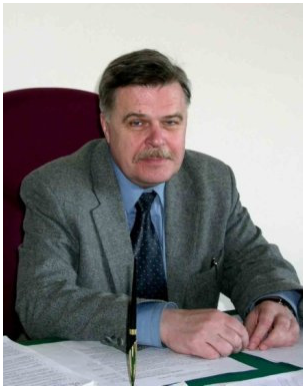
Предложен метод автоматического построения автоматных моделей существующих веб-приложений с учетом специфики заданной области. Даже для сложных веб-приложений возможно построение наглядных и понятных моделей, благодаря разработанному алгоритму поиска и слияния похожих состояний. Модели могут быть использованы для формализации требований спецификации и автоматической проверки выполнения этих требований при помощи верификатора. При обнаружении верификатором контрпримера, этот контрпример может быть автоматически переведен в выполнимый тест, запуск которого приведет к ошибке в самом веб-приложении. Также наличие модели может существенно автоматизировать процесс создания набора тестов, который позволит проверять заданные части веб-приложения.

Литература

1. Wikipedia, Web-Application article. http://en.wikipedia.org/wiki/Web_application
2. *Шалыто А. А., Туккель Н. И.* Программирование с явным выделением состояний // Мир ПК. 2001. № 8. С. 116–121; № 9. С. 132–138.
3. Программирование мобильных устройств. http://is.ifmo.ru/science/MD–Mobile.pdf
4. *Alalfi, M. H., Cordy, J. R.,Dean, T. R.* Modelling methods for web application verification and testing: state of the art. Softw. Test., Verif. Reliab. (2009) 265–296.
5. *Hassan A. E., Holt R. C.* Architecture recovery of web applications. Proceedings of the 24th International Conference on Software Engineering ICSE, ACM Press: New York, NY, USA, 2002; 349–359.
6. *Antoniol G., Di Penta M., Zazzara M.* UnderstandingWeb Applications through Dynamic Analysis. Proceedings of the 12th International Workshop on Program Comprehension IWPC, 2004; 120–131.
7. *Di Lucca G. A., Di Penta M*. Integrating Static and Dynamic Analysis to improve the Comprehension of Existing Web Applications. Proceedings of the Seventh IEEE International Symposium on Web Site Evolution WSE, IEEE Computer Society: Washington, DC, USA, 2005; 87–94.
8. *Sylvain Hallé, Taylor Ettema, Chris Bunch, Tevfik Bultan*. Eliminating navigation errors in web applications via Model Checking and runtime enforcement of navigation state machines. ASE 2010: 235–244.
9. *Haydar M.* Formal Framework for Automated Analysis and Verification of Web-Based Applications. In ASE(2004) 410–413.

10. *Atsuto Kubo, Hironori Washizaki, Yoshiaki Fukazawa.* Automatic Extraction and Verification of Page Transitions in a Web Application // Apsec. 14th Asia-Pacific Software Engineering Conference. 2007. Pp. 350–357.
11. Antawan Holmes, Marc Kellogg. Automating Functional Tests Using Selenium // Proceedings of the conference on AGILE 2006. July 23–28, 2006. Pp. 270–275.

Синтез элементов компьютерной архитектуры



Леонов
Геннадий Алексеевич

председатель оргкомитета конференции

д.ф.-м.н., профессор, чл.-корр. РАН
декан математико-механического факультета СПбГУ
заведующий кафедрой прикладной кибернетики СПбГУ

НЕЛИНЕЙНЫЙ АНАЛИЗ УСТРОЙСТВА ДВОИЧНОЙ ФАЗОВОЙ МАНИПУЛЯЦИИ

М. В. Юлдашев

аспирант; maratyv@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В данном сообщении рассмотрены основные методы анализа схемы Костаса, проанализированы недостатки каждого из них и предложена нелинейная модель исследования указанной схемы.

Введение

Существует множество подходов для обработки сигналов двоичной фазовой манипуляции (Binary Phase Shift Keying, BPSK) [1–3]. Большое распространение получил подход с использованием цифровой схемотехники. Однако скорость работы устройств на их основе ограничена параметрами аналого-цифровых преобразователей (АЦП). Один из первых аналоговых подходов связан с использованием систем фазовой автоподстройки частоты (ФАП) с квадратором. Однако основной элемент, позволяющий использовать систему ФАП с BPSK сигналом, сложен в реализации. В данном сообщении проведен анализ схемы, предложенной американским инженером Джоном Костасом.

Анализ схемы Костаса

Схема Костаса была предложена в 1950 г. и приобрела широкое распространение [1] и инженерами разработано множество подходов для её анализа [2,3]. Из основных методов можно выделить три направления: моделирование, линейный анализ и создание прототипов.

Рассмотрим блок-схему устройства на основе схемы Костаса (рис. 1).

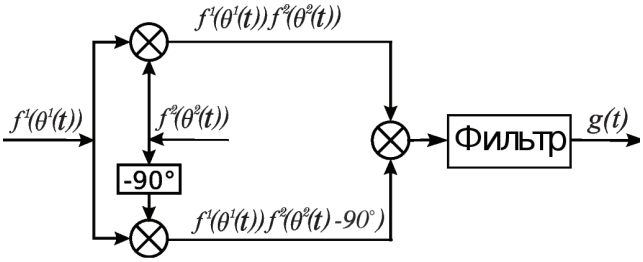


Рис. 1. Схема Костаса

где ⊗ — перемножители, а блок –90 сдвигает фазу входного сигнала. Результирующий сигнал поступает на вход линейного фильтра низких частот. Таким образом, в схеме присутствует сразу три нелинейных элемента. Хорошо известно, что линейный анализ нелинейных схем может приводить к неверным результатам, кроме того, ограничен малым промежутком времени исследования. Моделирование, в свою очередь, затруднено высокочастотной природой сигналов, для которых применяется данная схема [4, 5]. Это требует использования малого шага дискретизации, что ведет к увеличению времени моделирования.

В 1960-х годах, А. Витерби предложил следующий подход к изучению подобных схем. Вместо рассмотрения самих сигналов, было предложено исследовать поведение их фаз. Данных метод лишен приведенных недостатков, однако требует определения характеристики фазового детектора. Для не гармонических сигналов, для нахождения характеристики удобно пользоваться методом, аналогичным для систем ФАП[4]. Указанный метод позволяет прямо найти искомую характеристику, не прибегая к моделированию или прототипированию.

Заключение

Предложенный подход к анализу схемы Костаса может быть сведен к анализу динамических систем для устройств ФАП. Это позволяет существенно сократить время моделирования и расширить диапазон рабочих параметров подобных систем.

Литература

1. *J. Costas*. Synchoronous communications // Proc. IRE. Vol. 44. 1956. Pp. 1713–1718.
2. *E. Ronald.* Phase-Lock Loops: Design, Simulation and Application. 2003.
3. *F. Gardner*. Phase-lock techniques. New York: John Wiley, 1966.
4. *N. Kuznetsov, G. Leonov, P. Neittaanmki, S. Seledzhi, M. Yuldashev, and R. Yuldashev.* High-frequency analysis of phase-locked loop and phase detector characteristic computation // 8th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2011). INSTICC Press, 2011. Pp. 272–278, doi:10.5220/0 003 522 502 720 278.
5. *D. Abramovitch*. Efficient and flexible simulation of phase locked loops, part I: simulator design // American Control Conference. Seattle, WA, 2008. Pp. 4672–4677.

ЭФФЕКТИВНОЕ МОДЕЛИРОВАНИЕ СИСТЕМ ФАЗОВОЙ АВТОПОДСТРОЙКИ[1]

Р. В. Юлдашев

аспирант кафедры прикладной кибернетики;
renatyv@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В данной работе предложены аналитические методы, позволяющие эффективно моделировать системы ФАП. Для высокочастотных сигналов получена характеристика фазового детектора, необходимая для проведения моделирования систем ФАП.

Введение

Несмотря на то, что системы фазовой синхронизации (ФАП, PLL — Phase Locked Loop) являются одним из самых распространенных устройств управления [1–3], теоретический анализ таких систем отстает от практики [4]. В частности, полное численное моделирование ФАП встречается довольно редко.

Способ моделирования ФАП зависит от его внутреннего устройства [5, 6], но всем типам ФАП присуще проблема одновременного присутствия сигналов разных частот. А именно, на вход фазового детектора (ФД) поступают сигналы высокой частоты (до нескольких гигагерц) с подстраиваемого и эталонного генератора. Выход же фазового детектора после прохождения через фильтр содержит сигнал низкой частоты (до нескольких килогерц). Этот сигнал подается на вход подстраиваемого генератора, обеспечивая синхронизацию. Наличие сигналов высокой частоты требует, чтобы шаг дискретизации для входов фазового детектора был достаточно мал, с другой стороны шаг должен быть достаточно большим, чтобы изучить поведение сигнала на входе подстраиваемого генератора на больших промежутках времени.

Система ФАП в частотно-фазовом пространстве

Альтернативный подход состоит описании модели системы ФАП в пространстве фаз [7]:

$$dx/dt = Ax + \phi(\Delta\theta),$$

$$d\Delta\theta/dt = \Delta\Omega + Lc \cdot x.$$

[1] Работа выполнена при финансовой поддержке Министерства Образования и Науки РФ, Академии Наук Финляндии и Санкт-Петербургского Государственного университета.

Это модель не содержит сигналов высокой частоты, но требует отыскания функции (т. н. характеристики ФД), определяющей поведение ФД в частотно-фазовом пространстве. Классической реализацией ФД является перемножитель. Характеристика $\phi(\Delta\theta)$ такого ФД для случая синусоидальных входных сигналов хорошо известна. Автором получена характеристика ФД для случая несинусоидальных сигналов [5].

Заключение

В документе были представлены основные стили текста, которые могут быть использованы при форматировании тезисов конференции СПИСОК-2012. Собственные тезисы рекомендуется набирать в этом документе, заменяя текст и заголовки на свои.

Литература

1. W. Lindsey. New Jersey: Prentice-Hall, 1972.
2. *Smith S. W.* The Scientist and Engineer's Guide to Digital Signal Processing. San Diego, California: California Technical Publishing, 1999.
3. *Viterbi A.* Principles of coherent communications. New York: McGraw-Hill, 1966.
4. *Abramovitch D.* Phase-Locked Loops: A control Centric Tutorial // Proceedings of the American Control Conference. Vol. 1. 2002. Pp. 115.
5. *N. V. Kuznetsov, G. A. Leonov, M. V. Yuldashev, R. V. Yuldashev.* Analytical methods for computation of phase-detector characteristics and PLL design // International Symposium on Signals, Circuits and Systems. IEEE press, 2011. Pp. 710.
6. *G. Leonov, S. Seledzhi, N. Kuznetsov, P. Neittaanmaki*. Asymptotic analysis of phase control system for clocks in multiprocessor arrays // International Conference on Informatics in Control, Automation and Robotics, Proceedings. Vol. Signal processing, Systems Modeling and Control (ICINCO-SPSMC). INSTICC Press, 2010. Pp. 99102.
7. *G. Leonov, S. Seledghi*. Stability and bifurcations of phase-locked loops for digital signal processors // International journal of bifurcation and chaos. 2005. Vol. 15. No. 4. Pp. 1347–1360.

НЕЛИНЕЙНЫЙ АНАЛИЗ АНАЛОГОВЫХ СИСТЕМ ФАЗОВОЙ СИНХРОНИЗАЦИИ, ИСПОЛЬЗУЕМЫХ В СОВРЕМЕННЫХ КОМПЬЮТЕРНЫХ АРХИТЕКТУРАХ И ТЕЛЕКОММУНИКАЦИЯХ[1]

Н. В. Кузнецов, Г. А. Леонов, С. М. Селеджи

кафедра прикладной кибернетики, математико-механический факультет

Санкт-Петербургский государственный университет

Аннотация: Системы фазовой автоподстройки широко распространены в радио, телевидении и компьютерных архитектурах. В настоящем сообщении приведен обзор существующих подходов к изучению и синтезу таких систем, рассмотрены различные математические модели аналоговых систем фазовой автоподстройки.

Введение

Системы фазовой автоподстройки (ФАП) были изобретены в начале 30-х годов прошлого века и широко используются в радио и телевидении (демодуляция и восстановление несущей, синхронизация и синтез генераторов) [1–5]. Существует несколько типов систем ФАП (классическая, цифровая, аналогово-цифровая и т. д.), которые используются в вычислительной технике, телекоммуникациях и др. Принцип работы ФАП остоит в формировании электрического сигнала (напряжения), фаза которого автоматически подстраивается к фазе эталонного генератора, т.е. устраняется расфазировка между сигналами [4–8]. Сигналы подстраиваемого и эталонного генератора поступают на фазовый детектор (ФД). Это устройство является нелинейным элементом, позволяющим получить корректирующий сигнал, соответствующий разности фаз между двумя входными сигналами. Для выделения управляющего сигнала, выход ФД соединяют со входом фильтра низких частот. Полученный управляющий сигнал подается на вход подстраиваемого генератора.

Методы анализа систем фазовой автоподстройки

Одной из главных характеристик ФАП является полоса захвата, т.е. полоса частот подстраиваемого генератора, в которой обеспечивается синхронизация с эталонным генератором (ЭГ) [8–10]. Таким образом, одной из важных задач при проектировании ФАП является определение характеристик системы (параметров входящих блоков) обеспечивающих требуемые свойства

[1] Работа поддержана Министерством Образования и Науки РФ.

работы ФАП. Для решения этой задачи применяются эксперименты на самих стройствах, аналитические методы и компьютерное моделирование математических моделей ФАП. Эти инструменты позволяют получить критерии устойчивости рабочих режимов, оценки их областей притяжения и оценки времени переходного процесса.

Заметим, однако, что для проведения строгого математического анализа ФАП необходимо принимать во внимание, что принципы работы устройства, изложенные выше, требуют построения адекватных нелинейных моделей в частотно-фазовом пространстве [10–13]. Кроме того, необходимо обосновать построение такой модели. Несмотря на это, хорошо известным экспертом в системах фазовой автоподстройки Д. Абрамовичем было отмечено, что «основным направлением изучения ФАП в современной литературе является применение упрощенных линейных моделей, приближенных расчетов и компьютерного моделирования». Однако, хорошо известно, что применение линеаризации и компьютерного моделирования для систем управления может приводить к неверным результатам (напр., эффекты Перрона смены знака Ляпуновской экспоненты, гипотезы Айзермана и Калмана об абсолютной устойчивости, гипотеза фильтра, локаллизация скрытых аттракторов) и требует специального обоснования.

В данной работе рассмотрен общий подход к нелинейному анализу и проектированию аналоговых систем фазовой синхронизации, который основан на построении нелинейных математических моделей в пространстве фаз и применении методов асимптотического анализа высокочастотных колебаний.

Литература

1. *G. A. Leonov, N. V. Kuznetsov, M. V. Yuldashev, R. V. Yuldashev.* Nonlinear models of Costas loop // Doklady Mathematics, 2012 (in print).
2. Патент на полезную модель № 112555 «Модулятор параметров фазового детектора», Леонов Г. А., Селеджи С. М., Кузнецов Н. В., Юлдашев М. В., Юлдашев Р. В., от 10.01.2012.
3. Патент на изобретение № 2010149471 «Способ для определения рабочих параметров фазовой автоподстройки частоты генератора и устройство для его реализации», Леонов Г. А., Селеджи С. М., Кузнецов Н. В., Юлдашев М. В., Юлдашев Р. В., от 27.10.2011.
4. Свидетельство о государственной регистрации программы для ЭВМ № 2011613388 «Программа для определения и моделирования основных характеристик систем фазовой автоподстройки частоты», Юлдашев М. В., Юлдашев Р. В., Кузнецов Н. В., Леонов Г. А., Селеджи С. М., от 29 апреля 2011.
5. Свидетельство о государственной регистрации программы для ЭВМ № 2011616770 «Программа для определения и моделирования основных характеристик систем Costas Loop», Юлдашев М. В., Юлдашев Р. В., Кузнецов Н. В., Леонов Г. А., Селеджи С. М., от 31 августа 2011.

6. *G. A. Leonov, S. M. Seledzhi, N. V. Kuznetsov, P. Neittaanmaki.* Asymptotic analysis of phase control system for clocks in multiprocessor arrays // ICINCO 2010. Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics, 3. 2010. P. 99–102 (doi: 10.5220/0002938200990102).
7. *N. V. Kuznetsov, G. A. Leonov, S. M. Seledzhi.* Phase locked loops design and analysis // ICINCO 2008. 5th International Conference on Informatics in Control, Automation and Robotics, Proceedings. SPSMC, 2008. 3. 114–118 (doi:10.5220/0001485401140118)
8. *G. Leonov.* Computation of phase detector characteristics in phase-locked loops for clock synchronization // Doklady Mathematics. 78 (1). 2008. 3. 643–645.
9. *N. V. Kuznetsov, G. A. Leonov, S. M. Seledzhi*. Analysis of phase-locked systems with discontinuous characteristics of the phase detectors // IFAC Proceedings Volumes (IFAC-PapersOnline), 1 (PART 1). 2006. 3. 107–112 (doi:10.3182/20060628–3-FR-3903.00021)
10. *G. A. Leonov.* Phase-Locked Loops // Theory and Application. Automation and remote control. 10. 2006. P. 47–55.
11. *G. A. Leonov, S. M. Seledzhi.* Design of phase-locked loops for digital signal processors // International journal of Innovative Computing, Information and Control. Vol. 1. No. 4. 2005. Pp. 779–789.
12. *G. Leonov, S. Seledzhi*. Stability and bifurcations of phase-locked loops for digital signal processors // International journal of bifurcation and chaos. Vol. 15. No. 4. 2005. Pp. 1347–1360.
13. *Г. А. Леонов, С. М. Селеджи*. Системы фазовой синхронизации в аналоговой и цифровой схемотехнике. СПб.: Невский диалект, 2002.

СОТРУДНИЧЕСТВО С ИТ КОМПАНИЯМИ НА КАФЕДРЕ ПРИКЛАДНОЙ КИБЕРНЕТИКИ: РАБОТА И СТАЖИРОВКА СТУДЕНТОВ В КОМПАНИИ ИНФОРМАТИКА

***Н. В. Кузнецов*[1]**

nkuznetsov239@gmail.com

***С. В. Кузнецов*[2], *Г. А. Леонов*[1], *С. М. Селеджи*[1]**

[1] Санкт-Петербургский государственный университет

[2] Informatica Corporation (R&D Center branch in Russia)
(*http://www.informatica.com/us/*)

Информатика является международной компанией, штаб-квартира которой находится в городе Редвуд, штат Калифорния, США. Она является признанным лидером в интеграции данных. Офисы компании расположены в 60 странах мира. Программа сотрудничества между компанией и Санкт-Петербургским Государственным Университетом началась в 2011 году как совместная инициатива двух авторов С. Кузнецова — главы центра исследований и разработки компании Информатика в России и Г. Леонова — декана математико-механического факультета.

Корпорация Информатика имеет офисы в трех главных городах России — Санкт-Петербург, Москва и Казань — с главным офисом в Санкт-Петербурге. Компания понимает важность построения прочных отношений с Санкт-Петербургским Университетом для достижения цели совместного обучения программистов управлению и интеграции данных и уже имеет опыт программ сотрудничества в России, США и Индии. Российская программа сотрудничества сосредоточена в Санкт-Петербурге.

Основная цель программы состоит в подготовке студентов и аспирантов для разработки промышленного программного обеспечения. Так же программа предоставляет широкие возможности для последующей карьеры в области информационных технологий.

Сотрудничество между Санкт-Петербургским Государственным Университетом и корпорацией Информатика является выгодным для студентов с многих точек зрения. Они дополняют свои научные и технические навыки примерами коммерческого бизнес программирования из реальной жизни, увеличивают свои знания в области управления данными — самой быстрорастущей области информационных технологий. Университет предоставляет студетнам гибкое расписание занятий, позволяя проводить полный день внутри компании. Это, в частности, помогает познакомиться с большим количеством практик, что является крайне важным. Студентам предлагается

работать над рядом проектов под руководством главных инженеров и менеджеров корпорации Информатика. Проекты представляют собой различные области, что позоляет студентам выбирать из таких направлений как оптимизация алгоритмов, оценка передовых технологий, облачные вычисления и т. д. Такой подход дает возможность будущим программистам как увидеть и попробовать различные технологии, так и принять участие в командной работе. Работа над реальными проектами дополняется университетскими курсами, повышая вероятность успешного выполнения проектов.

При успешном завершении программы, лучшим студентам предлагается продолжить стажировку в компании, переходящую в работу с неполной занятостью. Конечно, не только успешная работа студента над проектом, но и макроэкономическое положение внутри компании влияет на такое решение. Не смотря на это, даже без предложения работы в компании, опыт и навыки полученные в течении стажировки открывают широкие возможности для плодотворного взаимодействия в будущем.

Более подробно о взаимодействии кафедры Прикладной Кибернетики с другими ИТ компаниями, такими как *Exigen Services*, *Hewlett-Packard*, *Intel*, *Motorola и Samsung Electronics можно узнать в* [1, 2].

Литература

1. *G. A. Leonov, V. I. Kiyaev, N. V. Kuznetsov, V. V. Onossovski, & S. M. Seledzhi*. Computers and software engineering: developing new models for educating mathematicians / S. Abramovich (Ed.) // Computers in education. Vol. 2. Hauppauge, NY: Nova Science Publishers, 2012. (In press.)
2. *Г. А. Леонов, В. И. Кияев, Н. В. Кузнецов, В. В. Оносовский, С. М. Селеджи*. Некоторые аспекты подготовки IT-менеджеров: специальные курсы и производственная практика — органически связанные части образовательного процесса // Леонид Витальевич Канторович: математика, менеджмент, информатика / Под ред. Г. А. Леонова, В. С. Катькало, А. В. Бухвалова. СПб.: Изд-во Высшая школа менеджмента, 2010. С. 297–528.

Математические методы и алгоритмы в системах хранения данных высокой производительности

Нестеров
Вячеслав Михайлович

д.ф.-м.н., генеральный директор
Центра разработок компании EMC в Санкт-Петербурге

Фёдоров
Андрей Рюрикович

генеральный директор компании Digital Design

МОДЕЛИРОВАНИЕ ИНТЕНСИВНОСТИ ВХОДНОГО ПОТОКА В СИСТЕМЕ ХРАНЕНИЯ ДАННЫХ

В. С. Дужин

аспирант кафедры комплексной защиты информации;
duzhin@vu.spb.ru

Санкт-Петербургский государственный университет аэрокосмического приборостроения

Аннотация: Целью данной работы является построение модели входного потока интенсивностей к многоуровневой системе хранения данных. Рассматриваются такие модели, как самоподобие и скрытая марковская модель. Самоподобие является одним из популярных направлений для моделирования трафика от большого количества источников в Интернете. Данная модель была рассмотрена, т. к. к системам хранения данных также обращается большое количество независимых источников. Скрытая марковская модель была рассмотрена, т. к. позволяет выделить несколько состояний во входном потоке.

Введение

Рассмотрим систему хранения данных, на вход которой поступает поток запросов от приложений. Запросы характеризуются временем прихода, запрашиваемым адресом, типом (чтение/запись), количеством считываемых или записываемых байт и др. Каждое приложение имеет требование к производительности. Система хранения включает в себя ряд параметров, изменяя которые можно добиться выполнения данных требований.

Постановка задачи

Целью данной работы является построение модели интенсивности входного потока запросов к системе хранения данных. Под потоком интенсивностей будем понимать поток, каждый отсчёт которого представляет собой количество запросов, пришедших к системе за определённый временной промежуток. Входная интенсивность представляет существенный интерес, поскольку с ростом данной характеристики растёт и количество уникальных адресов во входном потоке. Это оказывает влияние на алгоритмы, работающие с адресами, в частности, на алгоритмы управления кэшем. Также интенсивность влияет на время доступа к данным, т.к. при высоких интенсивностях образуются очереди на дисках. Модель интенсивности входного потока может, во-первых, способствовать предсказанию дальнейшего поведения потока (параметры модели могут быть оценены в реальном времени и использованы в управляющих алгоритмах). Во-вторых, с помощью модели можно сгенерировать потоки, близкие к исходным, для тестирования системы.

Самоподобие

Определение самоподобия

Одной из рассмотренных моделей является модель самоподобия. Данная модель часто применяется для описания web трафика, полученного от множества источников [1, 2]. Поскольку к системам хранения данных также обращается большое количество независимых приложений, было принято решение попробовать применить модель самоподобия для описания интенсивности потока.

Рассмотрим поток интенсивностей

$$X_1 = \{X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, \ldots\}.$$

Агрегированным потоком порядка 2 мы будем считать поток, полученный последовательным усреднением пар отсчётов потока X_1:

$$X_2 = \left\{ \frac{X_{11} + X_{12}}{2}, \frac{X_{13} + X_{14}}{2}, \ldots \right\}.$$

Дисперсия процесса X_2 определяется следующим образом (не умаляя общности, положим $M(X_1) = M(X_2) = 0$),

$$D_2 = M\left(\left(\frac{x_{2i} + x_{2i+1}}{2} \right)^2 \right) = \frac{1}{4}(D_1 + 2K_1(1) + D_1),$$

где $i \in N$; D_1 и $K_1(1)$ — дисперсия и корреляционный момент соседних отсчётов исходного процесса соответственно.

Учитывая, что $K_1(1) = D_1 \cdot r_1(1)$, где $r_1(1)$ — коэффициент корреляции между соседними отсчётами исходного процесса, получаем, что

$$D_2 = \frac{D_1}{2}(1 + r_1(1)). \qquad (1)$$

Из равенства (1) видно, что скорость уменьшения дисперсии при усреднении процесса зависит от степени корреляции соседних значений процесса. Если эти значения не коррелированы $(r_1(1) = 0)$, то при усреднении в 2 раза дисперсия также уменьшится в 2 раза. Если же между соседними значениями существует значительная корреляция $(r_1(1) \to 1)$, то при усреднении дисперсия будет уменьшаться крайне медленно.

Известно [2], что самоподобные процессы сохраняют свою структуру при агрегации. Процессы, не обладающие свойством самоподобия, при агрегации сглаживаются, их дисперсия падает.

Строго самоподобным процессом с коэффициентом Хёрста

$$H = 1 - \frac{\beta}{2}, \ 0 < \beta < 1$$

называется процесс, автокорреляционная функция (АКФ) которого имеет следующий вид:

$$r(k) = \frac{1}{2} \cdot \left((k+1)^{2-\beta} - 2k^{2-\beta} + (k-1)^{2-\beta} \right) = g(k).$$

Важным свойством данного процесса является постоянство АКФ при агрегации:

$$r_m(k) = g(k), \ \ m \in N.$$

Приняв в (1) автокорреляционную функцию $r_1(1) = g(1)$, после преобразований получим:

$$\frac{D_1}{D_2} = 2^{\beta}. \qquad (2)$$

Асимптотически самоподобный процесс — это такой процесс, АКФ которого при агрегации стремится к g(k):

$$\lim_{m \to \infty} r_m(k) = g(k), \ \ m \in N.$$

Поскольку на практике строго самоподобные процессы практически не встречаются [2], модель строгого самоподобия не является приемлемой. В дальнейшем под самоподобием будем понимать асимптотическое самоподобие.

Проверка гипотезы о самоподобии

Пусть задан эмпирически полученный случайный процесс X_1. Рассмотрим два подхода для проверки, является ли процесс самопободным.

1. **Метод АКФ.**

   Идея метода заключается в исследовании поведения АКФ процесса при агрегации. Если кривые АКФ агрегированных процессов стремятся к виду $g(k)$, гипотеза считается подтверждённой. В противном случае процесс считается несамоподобным.

2. **Метод дисперсий.**

   Рассматривается последовательность агрегаций исходного процесса с порядками, соответствующими степеням числа 2: $\{X_2, X_4, X_8, X_{16}, ...\}$. Строится график зависимости отношения дисперсий соседних процессов данной последовательности от порядка агрегации. Из формулы (2) можно сделать вывод, что для строго самоподобного процесса данный график должен представлять собой горизонтальную прямую. График асимптотически самоподобного процесса должен стремиться к данной горизонтальной прямой с ростом порядка агрегации.

Отметим, что агрегированные процессы строятся до тех пор, пока объём выборки очередного процесса не станет равным 100–200 отсчётам (при меньших выборках ошибки вычисления АКФ и дисперсий становятся недопустимо высокими).

Результаты экспериментов

На рисунках 1 и 2 представлены графики, построенные по реальному потоку запросов к системе хранения данных с использованием метода АКФ и метода дисперсий соответственно.

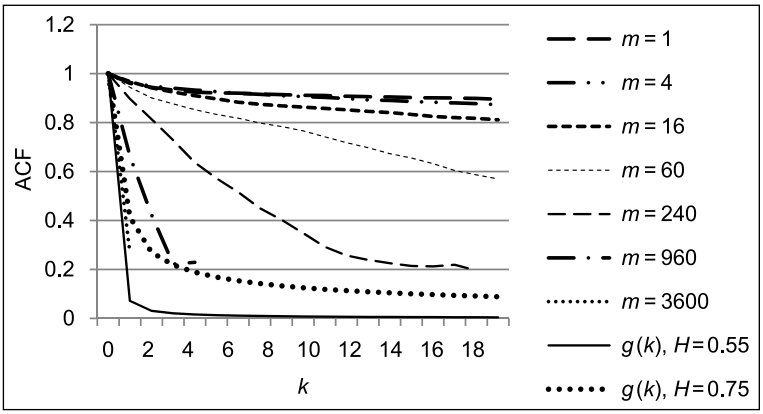


Рис. 1. АКФ исходного и агрегированных потоков

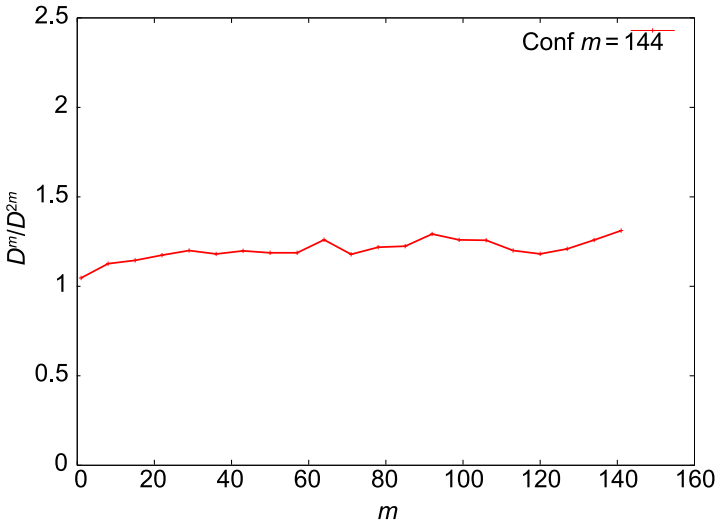


Рис. 2. Зависимость отношения дисперсий от порядка агрегации потока

На рисунке 1 изображены графики АКФ исходного потока ($m = 1$) и агрегированных потоков, а также кривые $g(k)$ строго самоподобных потоков с коэффициентом Хёрста 0.55 и 0.75. Из данного рисунка следует, что в исследуемом потоке отсутствует сильное самоподобие, т.к. кривая АКФ при агрегации по 3600 отсчётов расположена ниже кривой $g(k)$ с $H = 0.75$. Однако, сделать вывод о наличии самоподобия нельзя, поскольку при данной выборке невозможно определить, устремится ли АКФ потока к нулю при дальнейшей агрегации.

На рисунке 2 изображена зависимость отношения дисперсий от порядка агрегации. Поскольку на графике не прослеживается асимптотическое стремление линии регрессии к постоянной величине, можно заключить, что при данной выборке определить, является ли процесс самопободным, не представляется возможным.

Оценка объёма выборки

Существует проблема оценки минимально необходимого объёма выборки, при котором возможно проверить гипотезу о самоподобии процесса. Заметим, что данный объём зависит от коэффициента Хёрста H, т. к. с ростом H функция $g(k)$ убывает медленнее и для схождения АКФ исходного процесса к виду $g(k)$ требуется больше усреднений.

Поскольку данный коэффициент нам неизвестен, у нас нет возможности оценить необходимый объём выборки. Поэтому было решено провести ряд экспериментов с моделью типа On-Off [2], которая позволяет сгенерировать самоподобные потоки. Затем была проведена попытка доказать наличие самоподобия в сгенерированных потоках с помощью описанных выше методов. Данные эксперименты показали, что при выборке исходного потока, равной 10 млн. отсчётов, предел отношения дисперсий $\frac{D_m}{D_{2m}}$ всё ещё не был достигнут. Поскольку размер самого длинного из исследованных реальных потоков не превышает 500 тысяч отсчётов, было решено не использовать модель самоподобия.

Скрытая марковская модель

При анализе ряда входных потоков было выяснено, что в них зачастую можно выделить несколько ярко выраженных уровней интенсивности (см. рис. 3, *а*).

Каждый такой уровень может быть рассмотрен как состояние марковской цепи. Состояния нумеруются порядковыми номерами: S_1, S_2 и т.д. Положим, что процесс, находящийся в состоянии S_i, имеет вероятность перехода P_{ij} в состояние S_j. Это представлено графически на рисунке 4: круги соответствуют состояниям, а стрелки — вероятностям перехода.

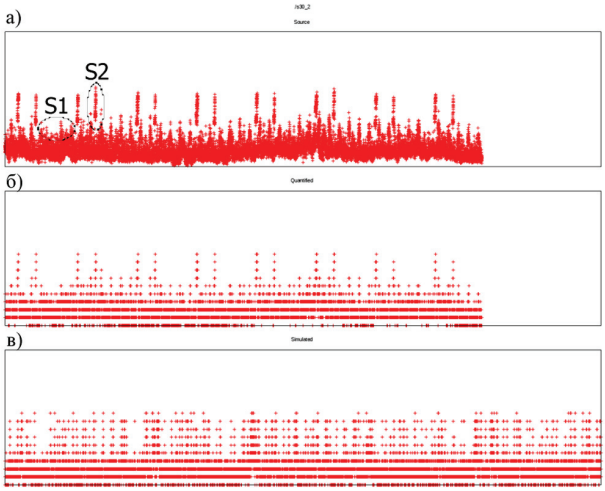


Рис. 3. Зависимость интенсивности от времени:
а — исходного, *б* — квантованного и *в* — сгенерированного потоков

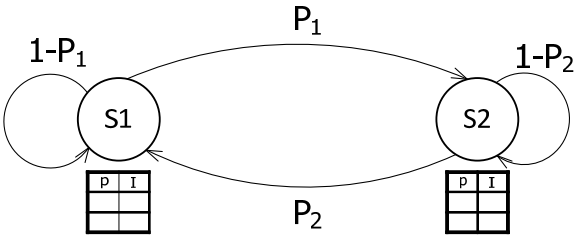


Рис. 4. Диаграмма переходов и матрицы интенсивностей скрытой марковской модели из двух состояний

Основным свойством марковской цепи является отсутствие зависимости вероятностей от истории переходов [3]. Скрытая марковская модель включает в себя марковскую цепь, обеспечивающую динамику процесса (изменение его состояний), а также набор распределений (по одному для каждого состояния), с помощью которых генерируется выходная интенсивность.

В данной работе для построения скрытой марковской модели был применён алгоритм Баума — Велча. Идея данного алгоритма состоит в поиске параметров модели, при которых вероятность генерации потока, идентичного исходному, будет максимальной.

Потоки, сгенерированные с помощью скрытой марковской модели, обладают схожими вероятностными харктеристиками с исходными потоками. К данным характеристикам относятся гистограмма плотности вероятностей, математическое ожидание, дисперсия. Однако, графики интенсивностей су-

щественно отличаются (см. рис. 3, *б*, *в*). Это объясняется тем, что данный подход не позволяет смоделировать длительное пребывание в одном состоянии. Поэтому предполагается использовать полумарковскую модель, т. к. вероятность остаться в том же состоянии в ней задаётся не одним значением, а выбирается из некоторого распределения вероятностей.

Заключение

Были рассмотрены такие модели интенсивности потока, как самоподобие и скрытая марковская модель. Первая модель была отклонена, поскольку при анализе входных потоков наличие самоподобия в них не было установлено по причине недостаточной выборки. С помощью второй модели были сгенерированы потоки, обладающие теми же характеристиками, что и исходные, но имеющие существенные отличия во внешнем виде. В дальнейшем предполагается рассмотреть полумарковскую модель, которая должна позволить решить эту проблему.

Литература

1. *D. G. Feitelson.* Workload Modeling for Computer Systems Performance Evaluation. Book draft, since 2005. http://www.almaden.ibm.com/cs/people/dmodha/iwqos.pdf [дата просмотра: 20.04.2012].
2. *Цыбаков Б. С.* Модель телетрафика на основе самоподобного случайного процесса // Радиотехника. № 5. 1999. С. 24–31.
3. *L. R. Rabiner.* A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition // Proceedings of the IEEE. 1989.

ИССЛЕДОВАНИЕ И ТЕСТИРОВАНИЕ СЭМПЛИРУЮЩЕГО МЕТОДА ПРОФАЙЛИНГА НА ПРИМЕРЕ ПРОФИЛИРОВЩИКА ПРОИЗВОДИТЕЛЬНОСТИ INTEL VTUNE AMPLIFIER XE 2011

Р. С. Одеров

студент кафедры системного программирования;
roman.oderov@gmail.com

С. А. Серко

студент кафедры системного программирования;
serko.sergej@gmail.com

Санкт-Петербургский государственный университет

Введение

В последнее время наблюдается тенденция к повышению производительности программного обеспечения. Высокопроизводительные программные продукты пользуются огромной популярностью, например, в сфере хранения и управления большими объемами данных. Такие системы должны уметь оптимально распоряжаться имеющимися ресурсами.

Для оптимизации ПО необходимы специальные средства, позволяющие оценить его производительность и понять, какую часть необходимо оптимизировать. Этим и занимается профайлер — программа, способная подсказать программисту, что замедляет исполнение написанного им кода.

Существует два подхода к профилированию: инструментирование и сэмплирование.

Суть метода инструментирования заключается в том, что в исходный или скомпилированный код вставляются дополнительные команды, которые, в свою очередь собирают информацию о ходе исполнения программы.

Сэмплирование — метод, при котором производятся замеры системных счетчиков программы через постоянный интервал времени. На основе полученной информации анализируется работа приложения.

В данной работе рассматривается сэмплирующий профайлер Intel VTune Amplifier XE 2011, описывается набор стресс-тестов и анализируются их результаты.

Постановка задачи

Заинтересовавшись процессом оптимизации ПО, мы решили разобраться подробнее, как он осуществляется, какими методами пользуются современные анализаторы производительности при профилировании. Последние вер-

сии наиболее известных профайлеров работают на сэмплировании. Выбрав в качестве рассматриваемого продукта Intel VTune Amplifier XE 2011, мы решили проанализировать его возможности (в основном рассматривались анализы HotSpots и LightWeightHotSpots). Для этого был реализован примитивный тестовый пример «EmptyFunction». Эта программа представляла собой одну «почти пустую» функцию:

```
int main(int argc, char **argv){
      int a = 0;
      for (int i = 0; i < 10; i++)
            a += i;
      return a;
}
```

При попытке профилирования был получен неожиданный для нас результат: программа не была отражена в результатах анализа. На основании этого была сформулирована следующая задача: выявить недостатки профайлера (точнее, подхода к профилированию, основанного на сэмплировании), его слабые места и влияние на систему в целом.

Решение поставленной задачи

Для решения поставленной задачи был написан небольшой начальный набор тестов, который, по нашему мнению, должен был каким-то образом отразить реальную ситуацию: либо подтвердить наши гипотезы, либо опровергнуть. Не все получилось так, как предполагалось, и мы решили представить предварительные результаты и обозначить дальнейшее направление движения.

Описание тестов

1. Отложив вышеописанный тест в сторону, мы, первым делом, захотели убедиться, что профайлер не ловит ничего лишнего. Т. е. в результатах анализа не должно было присутствовать ничего, кроме описанных в коде программы функций.

   Для этого реализован тест «SimpleAsm», содержащий элементарные действия:

   ```
   mov rcx, 10000000000
   l1:
   mov rax,rbp
   loop l1
   ```

   С учетом результата теста, с которого мы начали исследование, было выбрано достаточно большое количество итераций (10 000 000 000) и за-

пущен цикл, чтобы наполнить программу кодом, который все-таки будет замечен профайлером. Amplifier XE проявил себя достойно: отразил эти действия в результатах анализа и не показал ничего лишнего.

2. Один из следующих тестов реализует цепочку вызовов функций. С его помощью можно понять, ошибается ли сэмплирование при анализе вложенных функций. Схематично код программы можно изобразить так:

```
Main
---> ExternalFunc
      ---> MiddleFunc
            ---> InternalFunc
```

Т. е. происходит последовательный вызов трех вложенных функций. Тело каждой из них не содержало никаких действий, и, как уже должно быть очевидно, профайлер не заметил их. Затем функции были наполнены некоторыми примитивными арифметическими действиями (вычисление факториала числа). После этого результат анализа оказался удивительным, на первый взгляд, но ожидаемым при более детальном рассмотрении. Профайлер распознал только `Main` и `MiddleFunc` функции. Было выдвинуто предположение, что в дело вмешалась оптимизация компилятора. При отключении всех оптимизирующих опций мы получили ожидаемый результат.

Итак, профайлер смог раскрыть стек и не потерять ни одной функции.

3. Следующий тест — приложение, в котором создается множество новых процессов.

```
NewProcessCreateAndExit(){
      <...>
      CreateProcess(«EmptyFunction.exe»,...);
      <...>
}
<…>
void Main(){
      for (i = 0; i < N; i++)
            NewProcessCreateAndExit();
}
```

Amplifier XE выдал правильный результат, отследив все созданные процессы, но время работы данной программы под профайлингом увеличилось в сотни раз.

Время без профайлинга (создание 500 процессов): ~15 секунд

Время с профайлингом (создание 500 процессов): ~1000 секунд

К тому же стоит отметить внушительное время, затраченное на сбор, формализацию и визуализацию результата: около получаса.

4. Цель следующего теста — оценить количество пропущенных вызовов функции в результате сэмплирования.

   Идея: вызывать в циклах (в разных местах программы П1) функцию `IncCounter()`, замерить время работы функции с профайлером/без профайлера, оценить замедление (накладные расходы по времени) и количество пропущенных вызовов.

   Для этого написана отдельная программа (П2), реализующая подсчет времени работы только функции `IncCounter()`.

	HotSpots (HS)	LightWeight HS
IncCounter в программе П2 (итераций $75 * 10^8$)	575054 ms	25980 ms
Программа П2	628673 ms	652538 ms
IncCounter в программе П1 (итераций $3 * 47910^8$)	24708 ms	1158 ms
Программа П1	27121 ms	27893 ms

интервал сэмплирования — 10 миллисекунд
`IncCounter()` без профайлера 21585 миллисекунд (итераций $3*10^8$)
В программе П1 выполняется $3*10^8$ итераций `IncCounter()`.

Введем обозначения:

It» — количество итераций, замеченных профайлером;
It — общее кол-во итераций;
T_1 — время работы программы П1 без профайлинга;
T_2 — время работы программы П1 с профайлингом;
T_0 — время работы $3*10^8$ `IncCounter()` без профайлинга;
T_x — время работы $3*10^8$ `IncCounter()` с профайлингом;
$K = T_2/T_1$ — коэффициент замедления всей программы;
T_2.IncCounter — время работы функции `IncCounter()` в программе П1 с профайлингом.

Выдвинуты гипотезы (формулы подсчета), касающиеся вычисления процента потерь вызовов функции `IncCounter()` при профилировании:

- $It»/It = T_2.IncCounter/T_x$;
- $It»/It = T_2.IncCounter/(T_0 * K)$.

Тогда имеем следующий противоречивый результат:

	T_2.IncCounter/T_x	T_2.IncCounter/(T_0 * K)	K
LightWeight HS	1.1143	0.048084	1.11572
HS	1.0742	1.055163	1.08484

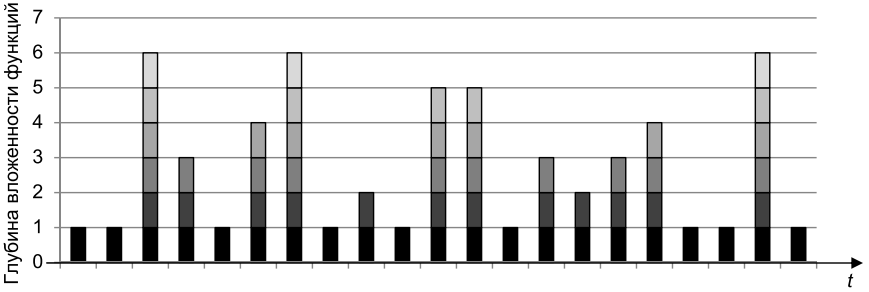
Т. о. можно увидеть из таблицы, что профайлер «заметил» больше функций, чем есть на самом деле. Попытавшись разобраться с этой проблемой, мы выяснили, что Amplifier XE использует статистический метод анализа, аппроксимируя реальную работу программы.

5. Для более полного понимания работы сэмплирующего метода профайлинга планируется написать следующий тест.

   Цель: из стека вызовов функций соорудить неравномерно распределенную (во времени) «пилу».

   Пример реализации: нужно вызывать различные функции *N* раз «в глубину», причем *N* будет меняться случайным образом.

   Графически это можно пояснить так:



С помощью описанного выше теста планируется выявить ошибки сэмплирующего подхода, вызванные аппроксимирующей техникой.

Заключение

В документе был представлен набор тестов, выявляющих недостатки сэмплирующего подхода к анализу производительности приложений. Проведен предварительный анализ полученных результатов и обозначено направление дальнейших исследований в рассматриваемой области.

Литература

1. http://software.intel.com/en-us/forums/showthread.php?t=81390
2. Intel Corporation // «Intel(R) VTune(TM) Amplifier XE 2011 Getting Started Tutorials».

РАЗРАБОТКА УТИЛИТЫ ДЛЯ ЭМПИРИЧЕСКОЙ ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ АЛГОРИТМОВ БЛОЧНОЙ ОБРАБОТКИ ДАННЫХ В ВЫСОКОНАГРУЖЕННЫХ СИСТЕМАХ

А. Г. Поваляев, В. В. Башун, В. О. Минченков, В. А. Ковтун, В. А. Роща

Аннотация: Целью представленной работы являлось создание необходимого инструментария (стенда) и проведение исследований по изучению влияния особенностей программно-аппаратной платформы на производительность алгоритмов блочной обработки данных. В качестве базы была выбрана операционная система Linux, работающая на процессорах с вычислительными узлами типа ccNUMA. Объектом исследования является программная реализация заданного алгоритма (здесь и далее просто алгоритм), представленная в виде подключаемой к стенду библиотеки. Для обмена данными между стендом и библиотекой выбран интерфейс, позволяющий задавать только входной, выходной буферы и буфер для хранения промежуточных результатов. Передачи других дополнительных данных интерфейс не предусматривает. Фактически для стенда функции, реализующие обработку данных, представляются «черным ящиком». Это является важной особенностью данного подхода.

В работе приводится общее описание архитектуры разработанного стенда и способы создания нагрузки в системе, рассматриваются и анализируются результаты, полученные в ходе экспериментальных исследования алгоритмов шифрования AES (CBC), AES (CTR), и алгоритмов сжатия LZO, quicklz и bCodec.

1. Введение

В данный момент существует большое количество различных программ для системы Linux, которые позволяют протестировать производительность различных модулей ОС либо производительность аппаратной части. Например, утилита Flexible IO Tester [1] позволяет провести тестирование системы IO и оценить производительность работы с жесткими дисками. Набор тестов Geekbench — cross-platform benchmark [2] позволяет провести анализ производительности оперативной памяти и скорости работы с целыми числами и с числами с плавающей запятой. Система SysBench — system performance benchmark [3] позволяет оценить производительность планировщиков Linux. Для получения результатов каждая из таких программ использует свой собственный специфичный набор нагрузок на систему и ряд специфических тестов. При этом в большинстве случаев сами тесты достаточно статичны и не имеют дополнительных параметров.

В представленной работе была поставлена задача создать инструмент, который бы позволил производить тестирование не самой системы, а оценивать эффективность работы алгоритмов в системе при различных типах нагрузок. Алгоритм рассматривался как набор функций, содержащихся во внешней подключаемой библиотеке.

При разработке устанавливалось, что стенд не обладает какой-либо информацией о характере исследуемого алгоритма, реализации, и все взаимодействие с библиотекой происходит через достаточно ограниченный интерфейс, общий для всех алгоритмов: передача блока данных на обработку и получение результата. Такой принцип «черного ящика» расширяет сферу применения данного подхода и делает возможным применение разработанного ПО к алгоритмам с «закрытой» реализацией). Для управления нагрузкой на систему был введен ряд эмуляторов нагрузок на различные подсистемы, реализованных соответствующими модулями в составе стенда.

Исследовательская часть предполагала использование разработанного ПО для определения эффективности работы алгоритмов при различных типах нагрузках, оценки степени их влияния в режиме ядра. Выборка составляется на основе задания блоков входных данных, которые подаются последовательно на вход алгоритма (размер входных данных может выбираться произвольно).

2. Описание исследуемых параметров

В качестве основной характеристики исследуемых алгоритмов использовалось время обработки блока данных заданного размера; при этом на среду вычисления (систему) действуют различные факторы, различные виды нагрузок. Входными параметрами задавались способ применения и интенсивнось создания той или иной нагрузки в различных комбинациях (конфигурация).

Архитектура разработанного стенда позволяет расширять инструментарий для исследования различных факторов влияния, задавать новые типы нагрузок. На данном этапе исследование производительности алгоритмов проводилось с использованием следующих факторов: нагрузка на CPU, нагрузка на кэш TLB и нагрузка на шину данных.

Нагрузка на CPU

Эмулятор нагрузки на CPU был реализован с использованием разноплановых вычислений с целыми числами, при этом минимизировалось количество обращений к оперативной памяти. Работа этого эмулятора задавалась следующими параметрами: количество потоков (влияет на интенсивность нагрузки), расположение эмулятора (в ядре или режиме пользователя) и расположение по процессорам (вычислительным ядрам).

Нагрузка на кэш TLB

Буфер ассоциативной трансляции (англ. Translation lookaside buffer, TLB) — это специализированный кэш центрального процессора, используемый для ускорения трансляции адреса виртуальной памяти в адрес физической памяти. Каждая запись содержит соответствие адреса страницы виртуальной памяти адресу физической памяти. Если происходит обращение к адресу, который отсутствует в TLB, процессор обходит таблицы страниц и сохраняет полученный адрес в TLB, что занимает примерно в 10–60 раз больше времени, чем обращение по адресу страницы, уже закешированной в TLB. Вероятность промаха TLB невысока и составляет в среднем от 0,01 % до 1 % [4]. Для оценки можно использовать утилиту oProfile, которая измеряет данный показатель.

Целью эмулятора данной нагрузки была максимизация вероятности промаха TLB. Это достигалось путем того, что параллельно производилось считывание/запись данных из памяти таким образом, чтобы вытеснить из TLB адреса страниц, с которыми работает алгоритм. Стопроцентную гарантию вытеснения обеспечить было невозможно, в том числе и по причине отсутствия дополнительной информации о способе организации TLB буфера на используемой для исследований аппаратной платформе.

Данный эмулятор имел следующий набор входных параметров: номер процессора для выполнения, размер памяти для записи (значение равное произведению количества записей в TLB на размер страницы памяти позволяло значительно увеличить вероятность вытеснения). Также было необходимо, чтобы данная нагрузка и исследуемый алгоритм, осуществляющий параллельно обработку данных, были привязаны к вычислительным ядрам, использующим один TLB.

Нагрузка на шину данных

Для описания процесса нагрузки на общую шину данных необходимо понимание архитектуры ccNUMA (cache coherency Non-Uniform Memory Access). NUMA — это схема реализации компьютерной памяти с ассиметричным доступом для процессоров [5]. Для такой архитектуры время доступа к памяти, из которой осуществляется операция чтения или записи данных, зависит от расположения памяти относительно процессора. Каждый узел состоит из одного или нескольких процессоров. Из данной схемы видно, что обращаясь к своей памяти, узел уменьшает нагрузку на общую шину. Вместе с тем, при обращении к памяти другого узла на общей межпроцессорной шине возникает дополнительная нагрузка. Скорость такого обращения будет значительно ниже, чем при обращении к локальной (связанной с данным вычислительным модулем) памяти.

Таким образом, для генерации нагрузки на общую шину необходимо обеспечить перекрестную работу с памятью, то есть процесс на одном NUMA узле должен работать с памятью другого NUMA узла.

Данный эмулятор имел следующий набор параметров: номер процессора для выполнения, размер памяти для работы, номер NUMA узла на котором выделялась память и расположение эмулятора (в ядре или режиме пользователя).

3. Общее описание компонентов стенда

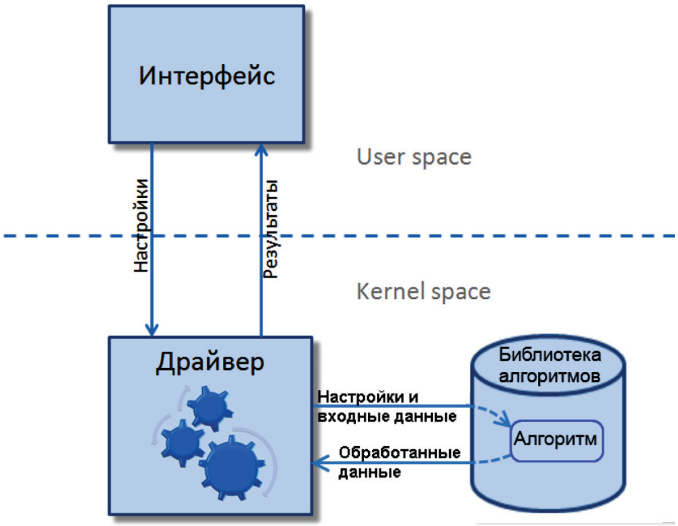


Рис. 1. Общее описание компонентов стенда.

В общем виде архитектуру стенда можно разделить на три составляющих компонента (см. рис. 1):

1. Пользовательский интерфейс — консоль управления, через которую оператор общается с низкоуровневой частью стенда (драйвер ядра) и управляет его работой. Данный компонент реализован в виде отдельного исполняемого файла. В его функции входит считывание заданной конфигурации, передача этих параметров драйверу ядра для генерации соответствующих нагрузок, инициация процесса запуска стенда, а также получение, интерпретация и сортировка результатов работы;
2. Драйвер ядра — основная часть стенда. В ней происходит запуск стенда и сбор статистики для исследуемого алгоритма, а также управление нагрузками, памятью и др. Данный компонент можно разделить на следующие компоненты (см. рис. 2):
   - управляющий модуль — координирует работу остальных модулей драйверной части. Отправляет «наверх» собранную информацию;
   - модуль профилировки — замеряет время работы исследуемого алгоритма на заданном наборе входных данных. Время работы скла-

дывается из времени обработки алгоритмом каждого блока данных. В зависимости от установленных параметров измерения производятся в миллисекундах, микросекундах, либо в наносекундах;
- модуль нагрузки — создает нагрузку, имитируя условия «реальной среды», в которых предстоит работать исследуемым алгоритмам;
- пул памяти — предназначен для работы с памятью на заданных NUMA узлах. Через пул памяти дается возможность управлять выделяемой памятью для работы самого алгоритма;

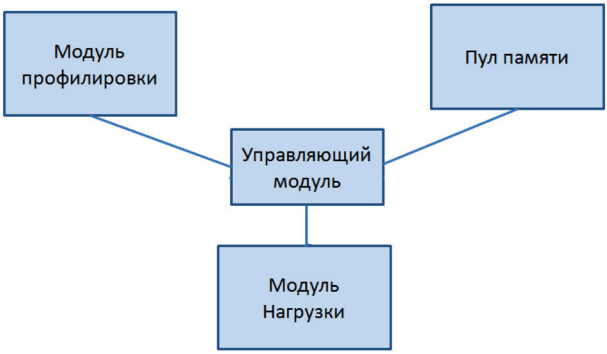


Рис. 2. Компоненты драйвера ядра

3. Библиотека алгоритмов — представляет собой набор алгоритмов.

Библиотека алгоритма предоставляет ядру свои функции, адреса и имена которых записываются в системную таблицу имен. В свою очередь стенд, зная имя функции, может получить ее адрес. После подключения функций алгоритма к стенду, передача информации происходит через общий интерфейса. Для этого в интерфейсе должна быть описана функция, обеспечивающая доступ к алгоритму:

```
static noinline int ProcessDataWithAlgorithm(char *inp_mem, int inp_mem_size, char *outp_mem, int outp_mem_size, int pull_mem_node, allocator pull_malloc, char *error);
```

где:

- `inp_mem` — указатель на входные данные, которые необходимо обработать;
- `inp_mem_size` — размер входных данных;
- `outp_mem` — указатель на буфер, куда будут записаны выходные данные;
- `outp_mem_size` — размер буфера выходных данных;
- `pull_mem_node` — NUMA узел, на котором алгоритму разрешено выделять память;

- `pull_malloc` — указатель на функцию выделения динамической памяти, имеющая следующий прототип:
- `void *allocator(unsigned int size, unsigned int node)`, где:
  - `size` — размер выделяемой памяти;
  - `node` — NUMA узел, на котором будет выделена память;
  - `error` — буфер, куда алгоритм может записать сообщение об ошибке, ограничен 256 байтами.

Входные данные передаются функции через буфер inp_mem. После вызова модулем ядра функция производит обработку данных в соответствии с реализованным алгоритмом. По завершении обработки результат записывается в буфер выходных данных outp_mem, и функция возвращается количество байт, записанных в буфер выходных данных, либо –1 в случае ошибки.

4. Проведение измерений

Измерения проводились на сервере, построенном на базе процессоров Intel XEON E5620 с поддержкой архитектуры NUMA. Согласно архитектуре, описанной на сайте производителя http://intel.com, начиная с процессоров серии 5500, процессор разделен на две части, названые CORE и UNCORE. В CORE часть входят все ядра, а также кэши первого и второго уровня, а в UNCORE часть входит третий уровень кэша, контроллер локальной оперативной памяти и системная шина, предназначенная как для связи между процессорами, так и для связи с южным мостом (рис. 3).

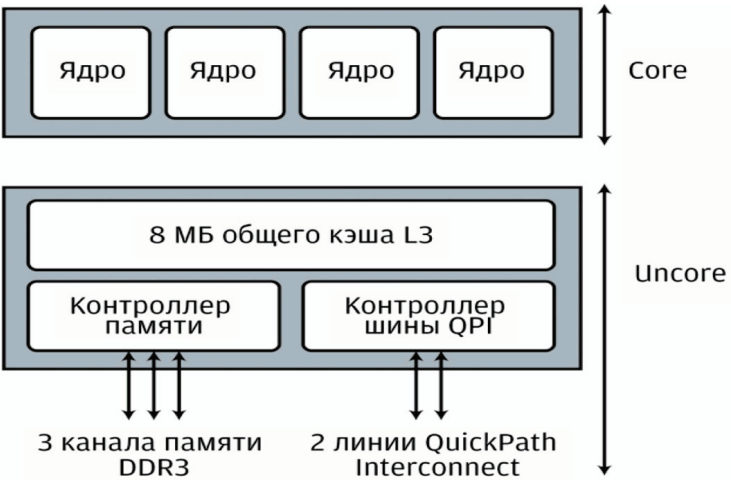


Рис. 3. Архитектура процессора серии 5500

Также о системе известны следующие параметры:

- Ядро Linux версии 2.6.32–71.el6.x86_64.
- Два 4-х ядерных процессора Intel Xeon E56202.40 GHz.
- Hyper-threading включен.
- Количество используемых логических процессоров — 16.
- Узлов NUMA — 2.

В качестве исследуемых алгоритмов были выбраны реализации шифрования AES (CBC) и AES (CTR) из криптографического пакета с открытым исходным кодом OpenSSL [6], и открытые реализации алгоритмов сжатия LZO [7] и quicklz [8]. Для подтверждения концепции исследования «черного ящика» в дополнение к этим алгоритмам также была взята закрытая реализация алгоритма сжатия bCodec с известным интерфейсом для использования.

Все исследуемые алгоритмы являются алгоритмами блочной обработки данных, соответственно дополнительным параметром для исследования был выбран параметр размера обрабатываемого блока за одну итерацию. Размеры блоков брались 4 кБ, 8 кБ, 16 кБ, 32 кБ и 64 кБ. Количество итераций для каждого эксперимента составляло $2 * 10^5$, а размер обрабатываемых данных равнялся $2 * 2^{30}$ байт.

Ниже приведены результаты исследования алгоритмов. По оси ординат обозначено время в микросекундах, которое в среднем тратилось на обработку 4 Кбайт данных. По оси абсцисс отложены размеры блоков, которые передавались функции алгоритма на обработку. На каждом графике приведены по четыре кривых: минимальное значение времени, максимальное значение времени, среднее время и среднее время без использования нагрузки.

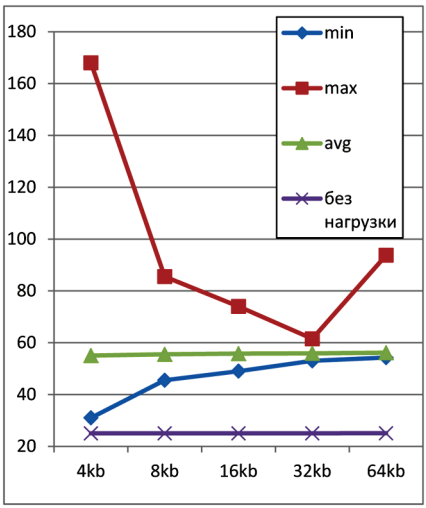


Рис. 4. График алгоритма AES-CBC. Нагрузка CPU

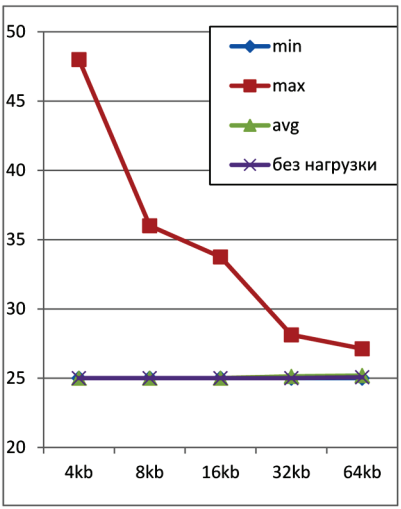


Рис. 5. График алгоритма AES-CBC. Нагрузка TLB

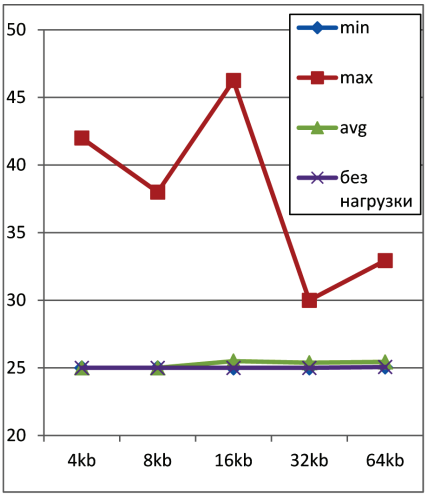


Рис. 6. График алгоритма AES-CBC. Нагрузка Шины

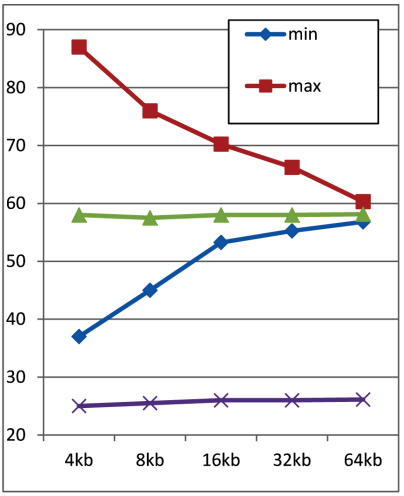


Рис. 7. График алгоритма AES-CTR. Нагрузка CPU

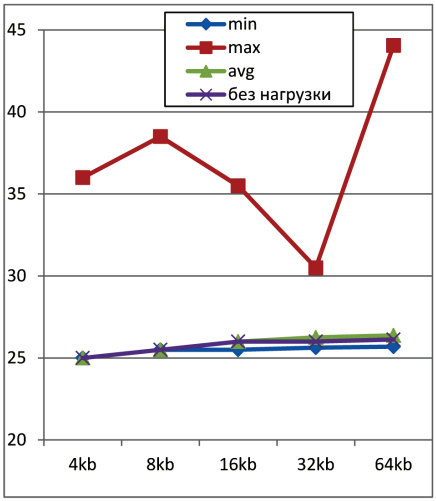


Рис. 8. График алгоритма AES-CTR. Нагрузка TLB

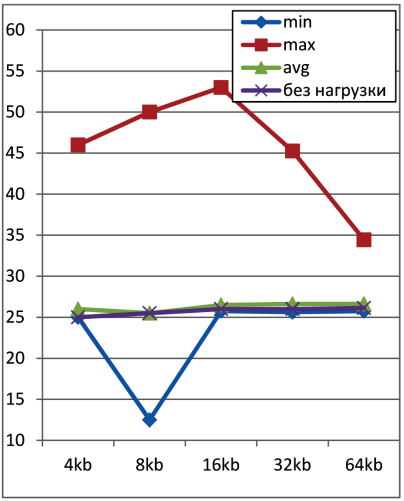


Рис. 9. График алгоритма AES-CTR. Нагрузка Шины

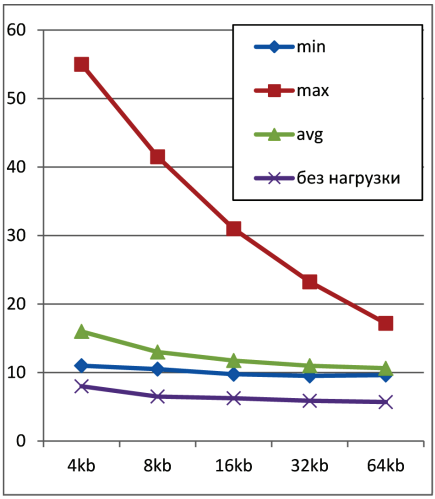


Рис. 10. График алгоритма quicklz. Нагрузка CPU

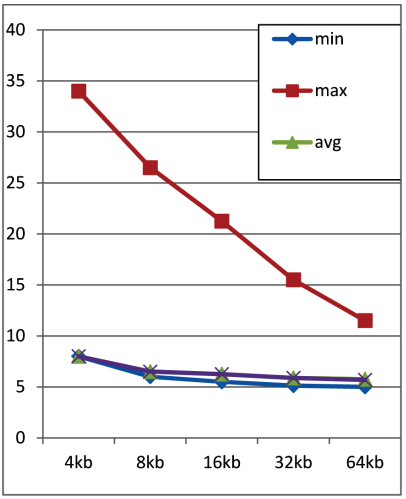


Рис. 11. График алгоритма quicklz. Нагрузка TLB

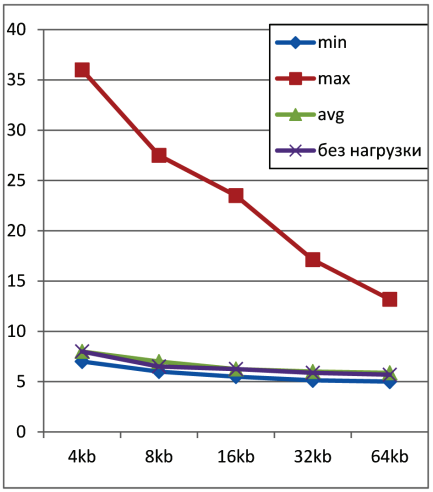


Рис. 12. График алгоритма quicklz. Нагрузка шины

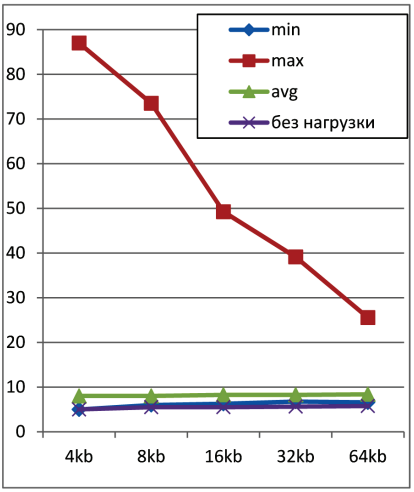


Рис. 13. График алгоритма LZO. Нагрузка CPU

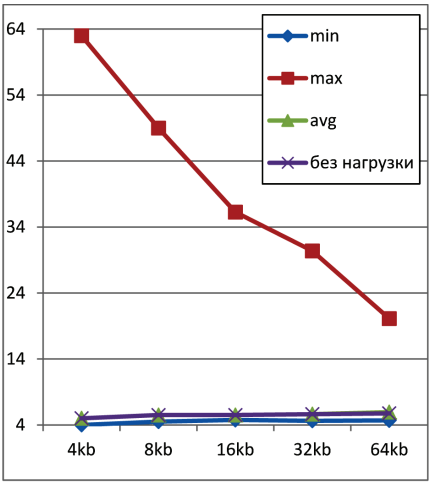


Рис. 14. График алгоритма LZO. Нагрузка TLB

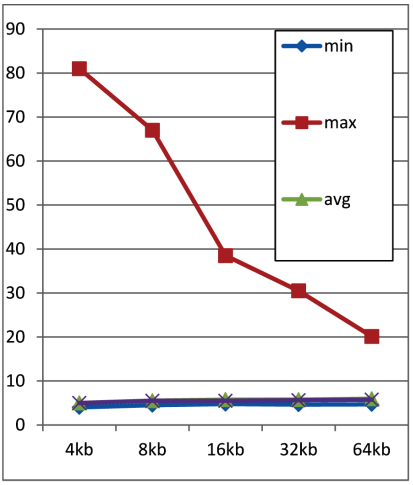


Рис. 15. График алгоритма LZO. Нагрузка шины

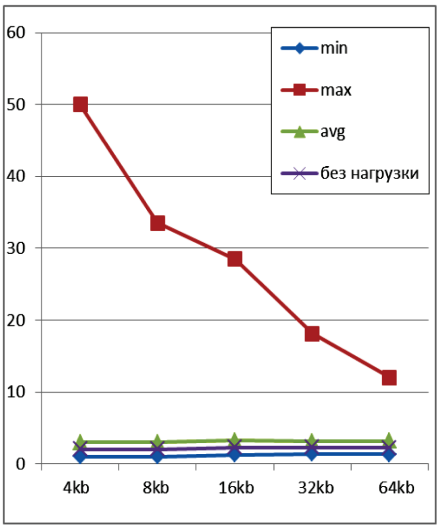


Рис. 16. График алгоритма bCodec. Нагрузка CPU

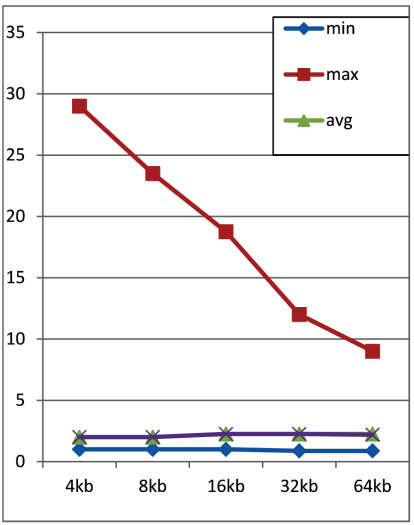


Рис. 17. График алгоритма bCodec. Нагрузка TLB

Из представленных выше результатов можно сделать следующие выводы:

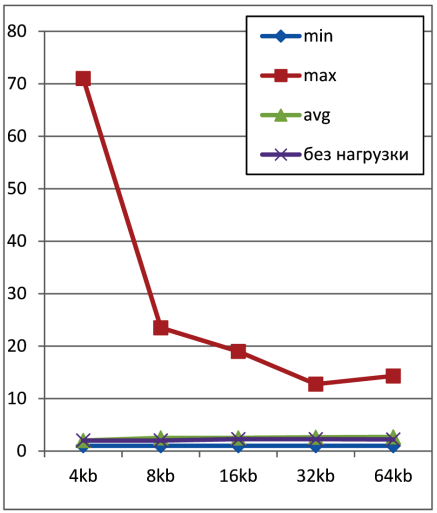


Рис. 18. График алгоритма bCodec. Нагрузка шины

1. Предполагалось, что при отсутствии нагрузки скорость обработки данных (время, затрачиваемое на 4 Кбайт) не должна сильно зависеть от размера блока, которыми данные «скармливаются» алгоритму. На практике увеличение размера блока при работе алгоритма носило положительный характер для алгоритма quicklz. А для алгоритма LZO, как показали измерения, увеличение размера блока с 4кб до 64 кб привело к потере производительности в среднем на 4.7%. Данные по остальным алгоритмам также показали отрицательно влияние увеличения размера блока на скорость обработки данных в условиях существования нагрузок;
2. Результаты показывают, что в данном случае наибольшее влияние вносила нагрузка на процессор. Это можно объяснить тем, что в данных экспериментах нагрузка создавалась на том же физическом ядре, где проводилась обработка данных алгоритмом, и их взаимовлияние преимущественно определялось используемой технологией Hyper-threading. Как видно из результатов, данный тип нагрузки меньше всего повлиял на алгоритмы bCodec и LZO — потери по времени составили порядка 50%. Для всех остальных — 100%.
   Данный факт объясняется тем, что время одной итерации для алгоритмов bCodec и LZO без нагрузки занимало от 2 до 10 раз меньше;
3. Намного меньшее влияние на скорость работы алгоритмов оказала нагрузка по TLB. Для алгоритмов сжатия данный вид нагрузки внес потери по времени порядка 2% при обработке данных блоками по 64 кб. Проведенный анализ показал, что нагрузка по TLB не была достаточно точно смоделирована. В качестве возможного решения рассматривается уточнение выбранной модели вычислительного узла, что будет являться объектом дальнейших исследований;
4. Нагрузка на шину данных дала ощутимые потери (в среднем 2%) только при обработке данных блоками по 64 кб. Аналогично с нагрузкой на TLB, данное влияние может быть существенным только при активной работе алгоритма с оперативной памятью;

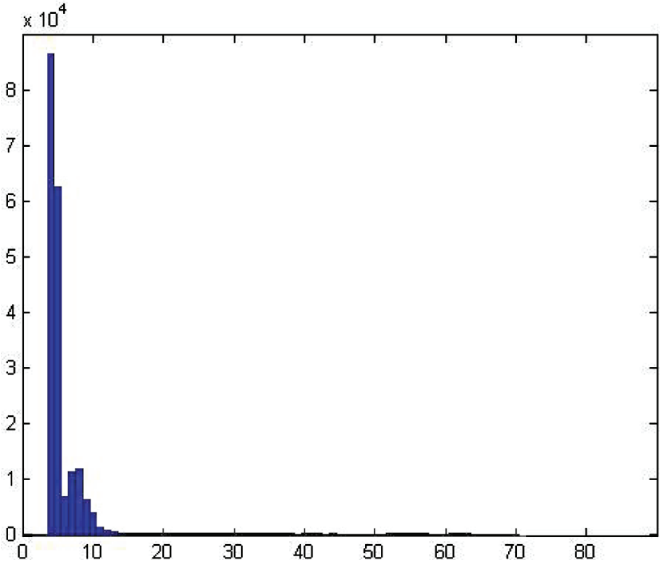


Рис. 19. Распределение времени для алгоритма LZO (блок 4 кб)

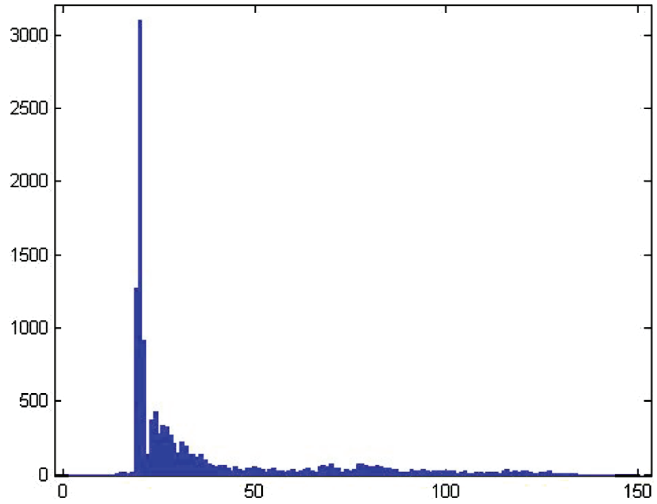


Рис. 20. Распределение времени для алгоритма LZO (блок 64 кб)

5. Для объяснения характера поведения графика максимального времени работы были собраны дополнительные статистические данные. На рис. 19 и 20 приведены гистограммы распределения времени выполнения для алгоритма LZO при работе с блоками 4 кб и 64 кб при работа-

ющей нагрузке на TLB. Как видно из результатов на гистограммах, имеется ярко выраженный максимум, т.е. появление максимума по времени носило единичный характер и имело крайне малую вероятность.

В ходе данной работы был разработан стенд, архитектура которого позволяет в дальнейшем достаточно просто увеличивать количество видов применяемых нагрузок и входных параметров. Для выбранной в данном случае модели вычислительного узла (на основе знаний, полученных из указанных в списке литературы источников) в экспериментах использовались нагрузки трех типов. Исходя из этого, был сконфигурирован стенд и проведен ряд успешных экспериментов над известными алгоритмами обработки блочных данных. Результаты экспериментов позволили сделать ряд практических выводов, в том числе появилась возможность судить об оптимальном размере блока обрабатываемых данных в высоконагруженной среде для заданного алгоритма. В качестве направлений дальнейших исследования рассматривается как уточнения модели вычислительного узла с целью более точной эмуляции нагрузок и уменьшения степени их взаимовлияния, так и добавление новых типов нагрузок, которые позволили бы говорить о более полной имитации реальной высоконагруженной среды.

Работа выполнена в рамках совместного научно-исследовательского проекта с компанией EMC.

Литература

1. Flexible IO Tester. http://www.obsecurities.com/reviews/5-fio-flexible-io-tester-review
2. Geekbench — cross-platform benchmark. http://www.primatelabs.ca/geekbench/
3. SysBench — system performance benchmark. http://sysbench.sourceforge.net/
4. Computer Organization And Design. Hardware/Software interface. 4th edition. Burlington, MA 01803, USA: Morgan Kaufmann Publishers, 2009.
5. *Christoph Lameter.* Local and Remote Memory: Memory in a Linux/NUMA System Jun 20th, 2006.
6. OpenSSL. http://openssl.org/
7. LZO. http://www.oberhumer.com/opensource/lzo/
8. Quicklz. http://www.quicklz.com/

ВЕРИФИКАЦИЯ ДИЗАССЕМБЛЕРА X86-64

Е. Д. Тенсин

студент математико-механического факультета;
Egor.Tensin@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В статье описывается проблема проверки корректности работы дизассемблера архитектуры x86-64. Также в статье дается эффективный алгоритм автоматизированной верификации дизассемблера с помощью аппаратного декодера процессора, реализующего архитектуру x86-64.

Введение

Дизассемблирование — трансляция машинного кода в код на языке ассемблера. Дизассемблирование инструкции состоит из двух этапов:

- декодирование инструкции из последовательности байтов в памяти;
- представление декодированной инструкции на языке ассемблера.

Декодер можно представить в виде конечного автомата, который последовательно читает байты в памяти, пока не окажется в одном из конечных состояний:

- инструкция декодирована и допустима;
- инструкция недопустима.

Ясно, что такой декодер имеет огромное количество состояний, что повышает вероятность ошибки и сложность программирования.

История проекта

В рамках проекта «Профайлер ядра MS WS 2008» (1) (2) от компании EMC (3) был разработан прототип профайлера, инструментирующий функции kernel API, предоставляющие доступ к спинлокам (spin locks). Используемая техника требует дизассемблер для декодирования инструкций, общая длина которых должна превзойти заданное число n. Было принято решение о написании собственного дизассемблера инструкций архитектуры x86-64. В связи с сложностью отладки встал вопрос о автоматической верификации произвольного дизассемблера инструкций архитектуры x86-64.

Обзор дизассемблеров

В таблице 1 приведен краткий список популярных дизассемблеров. Ни для одного из этих продуктов не было найдено исследований, подтвер-

ждающих корректность их работы и соответствие заявленому охвату пространства инструкций.

Таблица 1

Популярные дизассемблеры

Название	Версия	Автор	Примечания
OllyDbg (4)	2.01	Oleh Yuschuk	User-mode дебаггер; дизассемблер поддерживает только x86
Objconv (5)	2.12	Agner Fog	Конвертер объектных файлов форматов COFF/PE, OMF, ELF и Mach-O
IDA Pro (6)	6.x	Hex-Rays	Профессиональный, дорогой дизассемблер
gdb (7)	7.4.1	FSF	Стандартный дебаггер проекта GNU
PEBrowse (8)	10.1.4.4	Russel Osterlund	Дизассемблер и парсер формата PE

Постановка задачи

Пусть Valid — множество допустимых инструкций x86-64, и Decoded — множество декодированных дизассемблером инструкций. Тогда для верификации дизассемблера необходимо вычислить:

- Valid\Decoded: множество допустимых, но не декодированных инструкций;
- Decoded\Valid: множество ошибочно декодированных, недопустимых инструкций.

Для корректного декодера получим пустое множество в обоих случаях.

Алгоритм

На рис. 1 схематично показан общий алгоритм верификации дизассемблера x86-64.

Эталонный декодер

Эталонным декодером является декодер процессора, реализующего архитектуру x86-64. Таким образом, множество Valid удобно строить с помощью процессора, на котором проводится верификация.

Конкретнее, для последовательности байтов $\{x_i\}$, $i = 1, 2, ..., n$ необходимо реализовать программу, проверяющую, является ли эта последовательность допустимой инструкцией.

Необходимо судить о том, является ли эта последовательность допустимой инструкцией, по косвенным признакам. В этом качестве выступают оповещения процессора об исключительной ситуации (такой, например, как недопустимая инструкция). Интересующие нас исключения можно обрабатывать программно (см. Обработка исключений с помощью SEH).

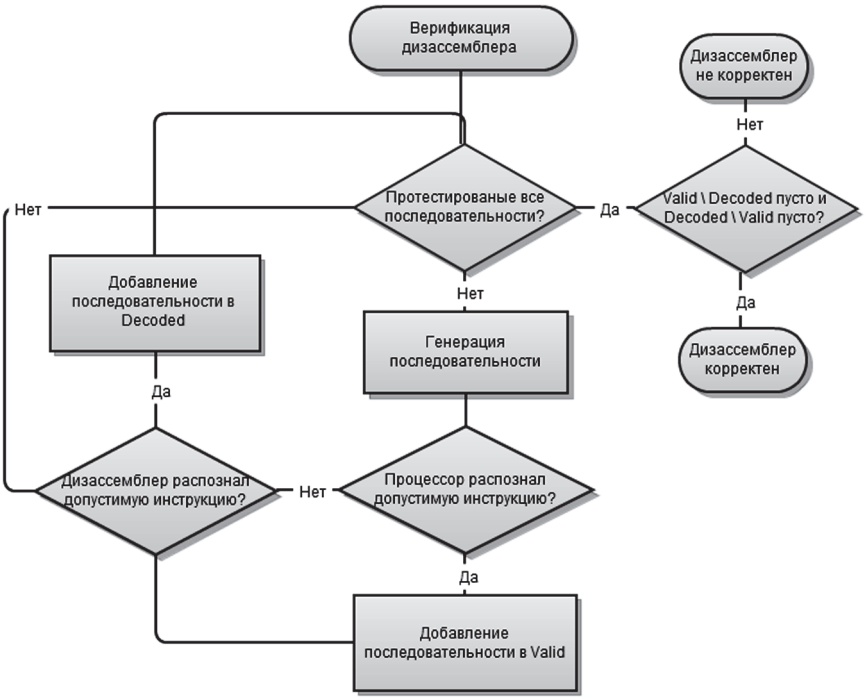


Рис. 1. Алгоритм верификации дизассемблера

Ограничения эталонного декодера

Нельзя проверить, является ли последовательность $\{x_i\}$, $i = 1, 2, ..., n$ допустимой инструкций, просто разместив ее в памяти и передав на соответствующий участок управление. Такая последовательность может кодировать 2, 3, ... n допустимых инструкции(й) (например, последовательность 0xcc 0x90 кодирует инструкции int 3 и nop). Поэтому для достоверной проверки придется по очереди тестировать на принадлежность подпоследовательности возрастающей длины. При этом после любого положительного результата тестирование можно прекращать, ведь если подпоследовательность $\{x_i\}$, $i = 1, 2, ..., m$ длины $m < n$ является допустимой инструкцией, то вся последовательность $\{x_i\}$, $i = 1, 2, \ldots, n$ в целом — нет.

Таким образом, проверка последовательности $\{x_i\}$, $i = 1, 2, \ldots, n$ разбивается на n итерации. Пусть далее start_i — адрес первого байта тестируемой подпоследовательности длины i на i-ой итерации, $\text{end}_i = \text{start}_i + n_i$.

Реализовать такой способ проверки можно, выделив две смежные страницы в памяти со специальными правами доступа. Пусть далее page_1 — адрес первой страницы, page_2 — адрес второй страницы. Первая страница получа-

ет разрешение только на чтение и исполнение, ко второй же любой доступ запрещается. Инструкция длины n размещается в $page_2$ — 1 и сдвигается на 1 байт влево на каждой итерации. Таким образом, процессор на i-ом тесте ($i = 1, 2, ..., n$) не сможет исполнить более чем i байт (помешают ограничения доступа ко второй странице).

Группы исключений

Исключения, возникающие во время декодирования и исполнения процессором тестируемой инструкции, можно поделить на группы:

- тестируемая инструкция недопустима (группа исключений FAIL);
- инструкция длины $m_j \leq n_j$ декодирована и выполнена (группа исключений SUCCESS);
- не удалось завершить выборку (группа исключений MORE).

Исключения групп FAIL и SUCCESS позволяют завершить работу программы. Исключение группы FAIL означает, что тестируемая инструкция не принадлежит множеству Valid. Заметим, что верно следующее: пусть $\{x_i\}$, $i = 1, 2, \ldots, m$, $m < n$ — допустимая инструкция; тогда $\{x_i\}$, $i = 1, 2, ..., n$ — недопустимая инструкция. Отсюда исключение группы SUCCESS обрабатывается следующим образом:

- если $n_i = n$, то последовательность кодирует допустимую инструкцию и принадлежит множеству Valid;
- иначе последовательность кодирует недопустимую инструкцию и не принадлежит множеству Valid .

Исключения группы MORE обрабатываются следующим образом:

- если $n_j = n$ (нет больше байтов для выборки), то последовательность кодирует недопустимую инструкцию и не принадлежит множеству Valid;
- иначе переходим к следующей подпоследовательности.

Обработка исключений с помощью SEH

Structured Exception Handling (9) — механизм обработки аппаратных и программных исключений. С помощью SEH можно идентифицировать исключения и сортировать по группам (см. Группы исключений).

Исключение access violation, к примеру, возникает, если:

- инструкция пытается прочитать ячейку памяти без разрешения на чтение,
- инструкция пытается записать в ячейку памяти без разрешения на запись,
- процессор производит выборку из памяти без разрешения на чтение.

При чтении и записи ячейки памяти исключение принадлежит группе SUCCESS (возникло при выполнении уже декодированной инструкции). В случае с выборкой SEH предоставляет два свойства исключения: адрес

инструкции, вызвавшей исключение ($addr_{instr}$), и адрес недоступной ячейки памяти ($addr_{unavailable}$).

- Если $addr_{unavailable} = end_i$ (произошла передача управления на адрес $addr_{unavailable}$), то это исключение относится к группе SUCCESS.
- Иначе:
  - если $addr_{instr} = start_i$ (выборка байтов для тестируемой инструкции), то исключение принадлежит группе MORE;
  - иначе ($addr_{instr} = end_i$; тестируемая инструкция декодирована и выполнена) инструкция принадлежит группе SUCCESS.

Исключения breakpoint и privileged instruction, с другой стороны, всегда относится к группе SUCCESS, а исключение illegal instruction всегда относится к группе FAIL.

Программный декодер

Программный декодер не передает декодированную инструкцию на исполнение, что позволяет во многом упростить построение множества Decoded по сравнению с построением множества Valid.

В случае с программным декодером, благодаря наличию API для декодирования ровно одной инструкции, нет нужды в проведении нескольких тестов для декодируемой инструкции. Во время работы программного декодера может возникнуть лишь access violation при чтении по адресу $page_2$. В таком случае тестируемая инструкция не принадлежит Decoded.

После благополучного завершения работы программного декодера тестируемая инструкция добавляется в Decoded в зависимости от результата:

- если программный декодер декодировал ровно n байтов, тестируемая инструкция добавляется в Decoded;
- иначе, тестируемая инструкция не принадлежит множеству Decoded.

Автоматизированная верификация

Ясно, что верификация дизассемблера должна проводиться по возможности автоматизированно. С этой целью верификация программного декодера производится единым способом — вызовом функции с заданным именем из динамически связываемой библиотеки. Таким образом, разработчику дизассемблера необходимо лишь предоставить библиотеку с реализацией требуемой функции.

Генерация тестируемых инструкций

Покрытие пространства возможных инструкций обеспечивается рациональной генерацией тестируемых инструкций. Полное покрытие пространства возможных инструкций возможно лишь с помощью полного перебора

двоичных последовательностей, не превышающих заданную длину. Учитывая максимальную длину допустимой инструкции (15 байтов согласно (10)), полный перебор займет $2^{\wedge 120}$ итераций, что недопустимо.

Генерация случайной последовательности, с другой стороны, не обеспечивает покрытия пространства возможных инструкций. В свете того, однако, что нам предстоит оптимизировать случайный перебор, генерация случайных последовательностей позволяет протестировать инструкции, пропущенные оптимизированным полным перебором.

Кодирование инструкций x86-64

Инструкция архитектуры x86-64 кодирует свою семантику и операнды. Семантика инструкции кодируется с помощью префиксов и опкода.

Опкод инструкции последовательность байтов, главным образом определяющая семантику инструкции. К примеру, опкод 0x90 идентифицирует инструкцию nop, а 0x0f05 — syscall.

Префиксы — это набор байтов, предшествующих опкоду. Префиксы влияют на семантику инструкции. К примеру, префикс lock (0xf0) гарантирует эксклюзивность доступа к ячейке памяти.

Операнд принадлежит к одному из следующих множеств:

- ячейки в памяти;
- значения в регистрах;
- целые числа.

Целые числа кодируются в самой инструкции как соответствующее представление числа в виде последовательности байтов. Ячейки в памяти и значения в регистрах кодируются с помощью более сложного механизма.

Ячейка в памяти вида [значение в регистре + смещение] кодируется с помощью байта ModRM. Байт ModRM делится на:

- поле mod (2 бита), кодирующее длину смещения;
- поле reg (3 бита);
- поле rm (3 бита), кодирующее базовый регистр.

Ячейка в памяти вида [значение в регистре + множитель * значение в регистре + смещение] кодируется с помощью байтов ModRM и SIB. Байт SIB делится на:

- поле scale (2 бита), кодирующее множитель;
- поле index (3 бита), кодирующее домножаемый регистр;
- поле base (3 бита), кодирующее базовый регистр.

Значения в регистрах кодируются либо с помощью поля ModRM.reg, либо с помощью поля ModRM.rm (если ModRM.mod == 11b). Таким образом, максимум два операнда могут быть значениями регистров.

Точно кодирование инструкции в архитектуре x86-64 определено в (11) и (10).

Оптимизация полного перебора

Перед оптимизацией полного перебора стоит задача уменьшить количество итераций, требуемых для покрытия пространства возможных инструкций. Это означает, что для каждого байта последовательности длины n необходимо выделить подмножество значений $|\{x_i\}| < 2^{\wedge 8}$.

Заметим, что, согласно (11) и (10), префиксы могут принимать лишь фиксированное количество значений.

Ясно также, что не имеет смысла перебирать все возможные операнды для заданного опкода и набора префиксов. Стоит ограничить, к примеру, значения, принимаемые операндом типа целое число следующими значениями (возможно, генерируемыми случайно):

- $0 < x < 32$ (для операций сдвига);
- значение, значительно превышающее 32;
- 0;
- отрицательное значение.

Аналогично, нет нужды перебирать все значения смещения при переборе операндов типа ячейка в памяти. Достаточно ограничиться, к примеру, следующими:

- 0;
- значение, превышающее длину страницы;
- значение, лежащее в интервале от 0 до длины страницы;
- отрицания к предыдущим двум.

Наконец, то же относится и к байтам ModRM и SIB. Можно перебирать лишь некоторые комбинации этих байтов, сделав при этом обход следующих аргументов:

- [значение регистра + смещение] с подстановкой двух различных регистров и для всех длин смещений;
- [значение регистра + множитель * значение регистра + смещение] с подстановкой 3-х случайных регистров в различных комбинациях в оба вхождения для всех множителей и длин смещений;
- два операнда типа «значение регистра» с подстановкой 3-х случайных регистров в различных комбинациях.

Заметим, что оптимизации такого рода неприменимы к байтам, кодирующим опкод инструкции.

В результате работы был разработан автоматизированный способ (см. Автоматизированная верификация), автоматической верификации заданного дизассемблера. В проектном дизассемблере (см. История проекта) было обнаружено и исправлено большое количество критических к работе ошибок.

Заключение

В статье была рассмотрена задача верификации дизассемблера архитектуры x86-64. Был предложен алгоритм автоматизированной верификации дизассемблера с помощью аппаратного декодера процессора, на котором производится верификация, состящий из двух этапов: генерация тестируемых последовательностей и проверка каждой из тестируемых последовательностей на принадлежность множеству допустимых инструкций. Проверка тестируемой последовательности на принадлежность множеству допустимых инструкций осуществляется на процессоре с помощью механизма обработки исключений. Был разработан оптимизированный алгоритм генерации последовательностей, с высокой вероятностью являющихся допустимыми инструкциями. Не был, однако, разработан алгоритм генерации последовательностей, почти наверняка не являющихся допустимыми инструкциями. Предлагаемый способ верификации, таким образом, не проверяет, не распознает ли верифицируемый дизассемблер «лишние» инструкции.

Литература

1. *Одеров Р. С., Серко С. С.* Исследование и тестирование семплирующего метода профайлинга на примере профилировщика производительности Intel VTune Amplifier XE 2011 // СПИСОК-2012. СПб.: СПбГУ, 2012.
2. *Одеров Р. С., Тенсин Е. Д.* Способы размещения своего кода в ядре ОС Microsoft Windows Server 2008 // Актуальные проблемы организации и технологии защиты информации. СПб.: СПбГУ ИТМО, 2012.
3. EMC — ведущий поставщик проверенных ИТ-решений, технологий для облачных вычислений и работы с большими данными.:. [Online] http://russia.emc.com/index.htm?fromGlobalSiteSelect
4. OllyDbg v1.10. [Online] http://www.ollydbg.de/
5. Software optimization resources. C++ and assembly. Windows, Linux, BSD, Mac OS X. [Online] http://www.agner.org/optimize/#objconv
6. IDA: About. [Online] http://www.hex-rays.com/products/ida/index.shtml
7. GDB: The GNU Project Debugger. [Online] http://sources.redhat.com/gdb/
8. Windows Disassembler. [Online] http://www.smidgeonsoft.prohosting.com/pebrowse-pro-file-viewer.html
9. Structured Exception Handling. [Online] http://msdn.microsoft.com/en-us/library/windows/desktop/ms680657(v=vs.85).aspx
10. AMD64 Architecture Programmer»s Manual Volume 3: General-Purpose and System Instructions. [Online] http://support.amd.com/us/Processor_TechDocs/24594_APM_v3.pdf.
11. Intel® 64 and IA-32 Architectures Software Developer»s Manual Combined Volumes 2A, 2B, and 2C: Instruction Set Reference, A-Z. [Online] http://download.intel.com/products/processor/manual/325383.pdf.

МОДЕЛИРОВАНИЕ ПОТОКА АДРЕСОВ В СИСТЕМЕ ХРАНЕНИЯ ДАННЫХ

К. С. Тяпочкин

студент математико-механического факультета[1]*, инженер*[2]*; tyapochkin.ks@gmail.com*

Д. А. Маличенко

асс. кафедры комплексной защиты информации[3]*; dml@vu.spb.ru*

[1] Санкт-Петербургский государственный университет

[2] Санкт-Петербургский центр разработок EMC

[3] Санкт-Петербургский государственный университет аэрокосмического приборостроения

Аннотация: В данной статье рассматривается вопрос моделирования потока адресов входных заявок к системе хранения данных. Необходимость моделирования потока входных заявок обуславливается тем, что с одной стороны сбор реальных логов является нетривиальной задачей, особенно в случае высоконагруженных систем хранения данных, и потребностью в таких логах для тестирования различных алгоритмов с другой стороны.

Рассматриваемая модель системы хранения данных включает в себя кэш, реализованный на основе алгоритма LRU и дисков нескольких типов, различающихся по производительности. Для таких систем при моделировании потока адресов входных заявок необходимо учитывать две характеристики нагрузки: временную локальность и распределение популярности адресов. В статье предложены два алгоритма, учитывающие обе характеристики.

Введение

Одним из ключевых параметров систем хранения данных является производительность. Она зависит в первую очередь от размера кэш-памяти и производительности используемых дисков. Кроме того, существенное влияние на производительность системы в целом оказывают реализованные в ней алгоритмы. К таким алгоритмам в первую очередь относится алгоритм работы кэш-памяти. Если в системе используются диски различных типов, то еще одним алгоритмом, оказывающим существенное влияние на производительность, является алгоритм автоматического переноса данных между дисками (например алгоритм FAST, реализованный в продуктах компании EMC [1]). Для проверки эффективности таких алгоритмов обычно используют реальные потоки заявок, взятые с действующих систем. Следует отметить,

что сбор таких логов с реальными потоками заявок является нетривиальной задачей, особенно в случае высоконагруженных систем хранения данных. Зачастую, количество имеющихся в наличии лог-файлов может оказаться недостаточным, а их продолжительность слишком короткой. В связи с этим задача генерации синтетической нагрузки становится особенно актуальной.

В данной статье рассматривается модель системы хранения данных с кэшем типа LRU [2, 3] и дисками нескольких типов, различающихся по производительности. Предполагается, что необходимо тестировать эффективность алгоритма работы кэша и алгоритма автоматического перемещения данных между дисками, где на более быстрые диски кладутся наиболее популярные данные (т. е. те данные, заявки к которым поступают чаще). Это означает, что необходимо учитывать временную локальность заявок и распределение популярности адресов, где под популярностью адреса понимается вероятность прихода заявки на этот адрес.

В качестве характеристики, описывающей временную локальность заявок, в работе использовано распределение стековых расстояний. Стековым расстоянием заявки к некоторому адресу назовем позицию этого адреса в LRU стеке. Или, что то же самое, количество различных адресов в потоке после предыдущего обращения к этому адресу. Для первого обращения к данному адресу стековое расстояние не определено. Например, рассмотрим поток адресов:

A B C D F C B A

Для последнего адреса A стековое расстояние будет равно 4, так как после предыдущего к нему обращения в потоке было 4 уникальных адреса: B, C, D и F.

Обе рассматриваемые характеристики могут быть вычислены из исходного потока адресов. Сложность построения гистограммы стековых расстояний составляет $O(M * \log(N))$, где M — количество адресов в исходном потоке, N — количество уникальных адресов [4], сложность вычисления гистограммы популярности адресов составляет $O(M)$ [5].

Способы моделирования потока адресов, учитывающие какую-либо одну из этих характеристик, уже разработаны [5]. Однако, моделирование потока, учитывающего сразу обе характеристики, представляет собой более трудную задачу. Далее в статье рассматриваются два алгоритма моделирования потока адресов, которые учитывают временную локальность и популярность по заданным гистограммам: модель эвристического приближения и агентная модель.

Модель эвристического приближения

Основная идея алгоритма состоит в том, что на каждом новом шаге добавления адреса характеристики текущего потока наилучшим образом приближаются к целевым.

На этапе инициализации в качестве начального возьмем поток, в котором все адреса в системе встречаются ровно по одному разу и расставлены в порядке убывания их популярности. После инициализации построим гистограммы стековых расстояний и популярности текущего (начального) потока. Гистограмма популярности будет равномерной. Заявок, имеющих стековые расстояния в текущем потоке пока нет, поэтому для определенности рассмотрим гистограмму, в которой каждому стековому расстоянию соответствует значение 0. Это пока не будет гистограммой в строгом смысле, однако на первом же шаге алгоритма это изменится.

При добавлении в поток каждого следующего адреса необходимо делать это таким образом, чтобы текущие гистограммы максимально приблизились к целевым. Как расстояние между гистограммами H_1 и H_2 выберем корень суммы квадратов разностей значений:

$$D(H_1, H_2) = \sqrt{\sum_i (x_i - y_i)^2},$$

где x_i — значение i-того столбца в гистограмме H_1, а y_i- значение i-того столбца в гистограмме H_2.

Таким образом, будем выбирать такой адрес, который минимизирует сумму квадратов двух расстояний, то есть после добавления которого величина $D(H_1^c, H_1^t)^2 + D(H_2^c, H_2^t)^2$ будет минимальной. Здесь H_1^c — текущая гистограмма стековых расстояний, H_1^t — целевая гистограмма стековых расстояний, H_2^c — текущая гистограмма популярности, H_2^t — целевая гистограмма популярности.

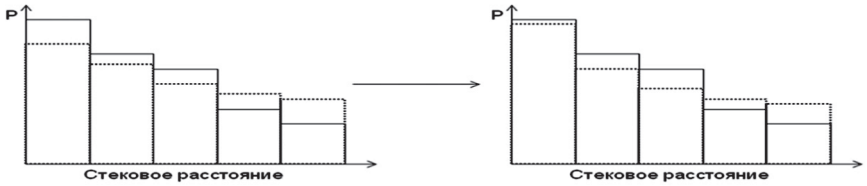


Рис. 1. Пример изменения гистограммы после добавления адреса

Агентная модель

Основная идея агентной модели — объединить адреса в несколько групп по популярности, после чего добавлять адреса в поток, выбирая группу согласно популярности, а внутри группы выбирать адрес учитывая стековые расстояния.

Разбиение на группы осуществляется таким образом, что популярности адресов в каждой группе приблизительно одинаковые. Для этого в гистограмме популярности, упорядоченной по убыванию, выбирается первый адрес (с самой большой популярностью) и в первую группу добавляются

все адреса, у которых популярность не более чем в два раза меньше первого, однако не менее трех адресов. Далее выбирается следующий адрес, который не попал в первую группу, и для него процедура повторяется. И так далее. Получаем группы адресов, в которых разница популярностей не превышает половины значения самого популярного адреса группы. Каждой группе i ставим в соответствие число p_i, равное сумме популярностей адресов внутри группы.

Далее, для каждой группы из исходного потока адресов формируется поток, в который входят адреса этой группы в том же порядке, что и в исходном потоке. После этого для этого нового потока считается гистограмма стековых расстояний. Таким образом, каждой группе ставится в соответствие своя вероятность и своя гистограмма стековых расстояний. Будем считать, что группа ведет себя как независимый агент, то есть при обращении к группе новый адрес выбирается независимо от остальных групп.

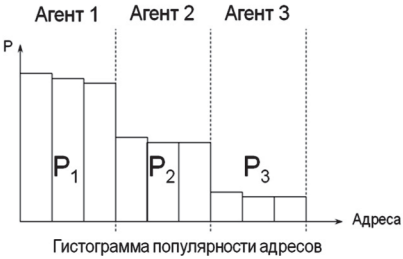


Рис. 2. Разбиение на группы по популярности

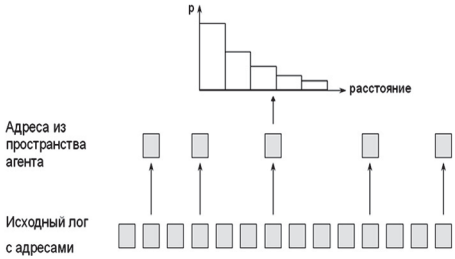


Рис. 3. Получение стековых расстояний в группах

Тогда алгоритм добавления последующего адреса в поток работает следующим образом. Сначала случайным образом выбираем группу в соответствии с имеющимися вероятностями p_i. Затем внутри этой группы выбираем элемент согласно гистограмме стековых расстояний, то есть случайным образом с вероятностями соответствующими гистограмме группы выбираем стековое расстояние, после чего добавляем элемент из этой же группы с этим же стековым расстоянием.

Сравнение моделей

Модель эвристического приближения достаточно сложно вычислима. Поскольку для добавления нового адреса в поток необходимо перебирать все возможные адреса и для них пересчитывать гистограммы, сложность алгоритма будет составлять $O(KN)$, где K — длина строящегося потока адресов, N — количество различных адресов. В то же время агентная модель несколько быстрее: оценка ее вычислительной сложности уже $O(KD)$, где D — размер наибольшей группы. Также, у модели эвристического прибли-

жения детерминированный результат, в то время как агентная модель — вероятностная.

Однако модель эвристического приближения имеет некоторые плюсы по сравнению с агентной: есть возможность легко изменять характеристики пространственной локальности на уровне общей гистограммы. У агентной модели изменять эти характеристики представляется более сложным в силу того, что она вычисляет их на основе уже имеющегося потока адресов. Также, можно отметить более точные результаты моделирования, полученные на тестовых примерах у модели эвристического приближения.

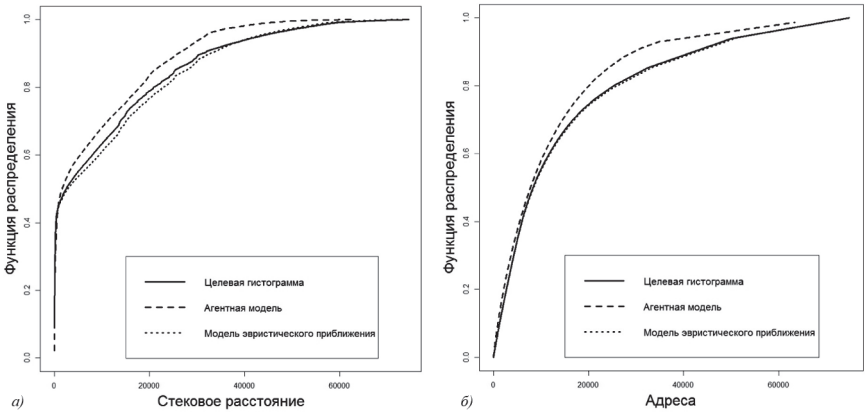


Рис. 4. Сравнение результатов моделирования алгоритмов:

а — кумулятивная функция гистограммы стековых расстояний; *б* — кумулятивная функция гистограммы популярности

Заключение

В статье представлены два алгоритма моделирования потока адресов для системы хранения данных с заданными характеристиками временной локальности и популярности, и проведено их сравнение. Алгоритм эвристического приближения показывает большую точность, однако при этом обладает и большей вычислительной сложностью. Также следует отметить, что если модель эвристического приближения позволяет напрямую задавать параметры временной локальности и распределение популярности адресов, то в агентной модели они вычисляются на основе уже имеющегося потока адресов, и изменение этих параметров представляет определенную сложность.

Литература

1. EMC VNX FAST VP, A Detailed Review. http://www.emc.com/collateral/software/white-papers/h8058-fast-vp-unified-storage-wp.pdf

2. *R. L. Mattson, J. Gecsei, D. R. Slutz*, and *I. L. Traiger*. Evaluation techniques for storage hierarchies. IBM Syst. J. 9(2). 1970. Pp. 78–117.
3. *J. R. Spirn.* Program Behavior: Models and Measurements. Elsevier North Holland Inc., 1977.
4. *G. Almasi, C. Cascaval*, and *D. A. Padua*. Calculating stack distances efficiently // Workshop Memory Syst. Perf. Jun 2002. Pp. 37–43
5. *Dror G*. Feitelson Workload Modeling for Computer Systems Performance Evaluation. http://www.cs.huji.ac.il/~feit/wlmod/wlmod.pdf

АЛГОРИТМЫ РАСЧЕТА RAID 6

К. И. Тюшев

студент кафедры системного программирования;
Kirill.Tyushev8@gmail.com

А. И. Калмук

студент кафедры системного программирования;
alexkalmuk@gmail.com

Научный руководитель:

Короткевич А. И.

alexey.korotkev14ich@gmail.com

Санкт-Петербургский государственный университет

Аннотация: В данной статье мы представим алгоритм расчета RAID 6, расскажем про две его реализации, а также сравним их с промышленной реализацией, используемой в СХД AVRORA.

Введение

Системы хранения данных (СХД) являются специализированными средствами для хранения, поддержки целостности и передачи информации. Для этих целей используются различные спецификации по количеству дисков, размещению данных и контрольных сумм. СХД AVRORA использует наиболее распространённую спецификацию — RAID 6, которая дополнительно хранит два диска с контрольными суммами (синдромами), что позволяет восстанавливать до двух дисков с данными. Алгоритм, используемый в СХД AVRORA, оптимизирован с использованием возможностей современных процессоров, и для своего выполнения требует поддержки команд SSSE3. Нашей целью было реализовать алгоритм расчета RAID 6, который превосходил бы существующий по производительности, но при этом использовал только наиболее распространенные команды процессора.

Алгоритм

Мы использовали метод расчёта RAID 6, основанный на подсчёте синдрома чётности и полиномиального синдрома Рида—Соломона.

Синдром чётности — это XOR данных по всем дискам.

Код Рида-Соломона имеет вид:

$$Q(g_0, \ldots, g_n) = x^n * g_n + x^{n-1} * g_{n-1} + \ldots + g_0,$$

где g_i — блоки данных фиксированного размера, интерпретируемые как элементы векторного пространства, а элементы $\{x^i\}_{i=0}^n$ лежат в некотором поле.

В алгоритме расчета RAID 6, описанном в [3], используется представление поля Галуа $GF(2^8)$ как кольца вычетов по модулю неприводимого многочлена, т. е. однобайтовых слов, с конструктивно описанным умножением. В данной работе мы воспользуемся представлением поля Галуа $GF(2^8)$ в матричной алгебре [4].

Реализация

Так как нашей задачей является максимально ускорить вычисления, то было решено писать код на языке ассемблера, что позволило более эффективно использовать регистры, чем после трансляции из языка Си, и проводить дополнительные оптимизации.

Было исследовано две реализации — с использованием команды PSHUFB и только XOR, соответственно. При использовании PSHUFB блок данных хранится на одном регистре. Поэтому мы можем проводить любые промежуточные вычисления без обращения к памяти, но нам нужно много раз проходить по всем дискам. Во второй реализации блок данных хранится на 8 регистрах. Мы меньше раз проходим по дискам, но для проведения некоторых вычислений без обращения к памяти не хватает регистров.

Почти во всех функциях библиотеки расчёта RAID 6 нужно пересчитывать контрольные суммы. Также расчёт контрольных сумм занимает основное время работы, так как нужно проходить по всем дискам. Поэтому важно как можно быстрее проводить умножение на X, чтобы быстро считать синдром Рида — Соломона по схеме Горнера.

Матрица X имеет достаточно простой вид. Поэтому умножение на X в матричном алгоритме можно представить несколькими инструкциями процессора. При использовании PSHUFB это один сдвиг, один shuffle, один xor и один mov. А во второй реализации это циклический сдвиг регистров и три инструкции xor. Но циклический сдвиг регистров можно не делать, развернув цикл на 8 итераций. Таким образом, в одной из реализаций умножение на X это три инструкции xor, при размере блока 128 байт.

В отличие от СХД AVRORA, одна из новых реализаций не использует команды SSSE3, что позволяет использовать её на большем числе процессоров.

Что сделано

Реализованы библиотеки функций для восстановления одного и двух дисков данных, восстановления диска синдрома вместе с диском данных, расчёта контрольных сумм, пересчёта синдромов при изменении данных на одном диске.

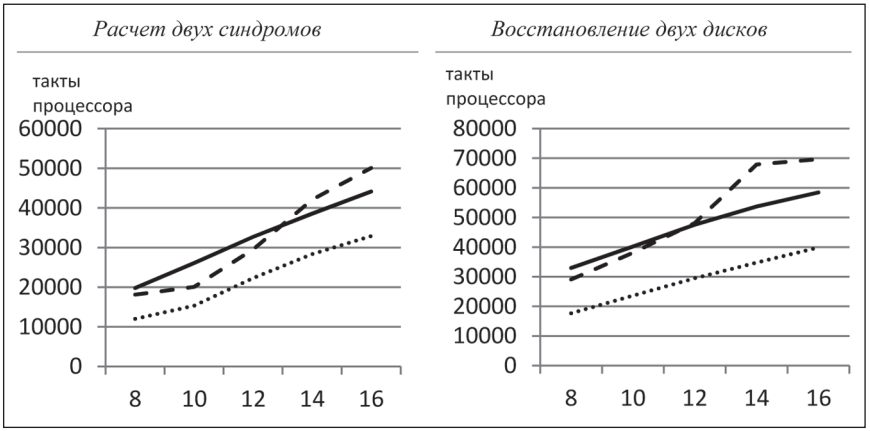
Для ускорения вычислений в обеих реализациях написан генератор кода умножения на произвольную матрицу.

Код оптимизирован под использование 128-битных регистров XMM и инструкций из технологии процессора SSE. Обращения к памяти происходят группами данных, кратными размеру линии кэша процессора.

При реализации алгоритма не использовались условные переходы из-за того, что они мешают одновременному исполнению инструкций, идущих друг за другом, так как заранее не известно, какая инструкция будет исполняться следующей.

Если в инструкции процессора используется регистр, значение которого вычисляется в предыдущей инструкции, то это также мешает одновременному исполнению инструкций, идущих друг за другом, так как приходится ждать окончания выполнения первой инструкции. Ввиду этого в реализациях инструкции перемешивались так, чтобы не допустить описанной выше ситуации.

Далее представлены графики времени работы двух новых реализаций и промышленной реализации СХД AVRORA в тестовом окружении:



Из графиков видно, что одна из реализаций матричного алгоритма работает быстрее, чем существующий промышленный код. А вторая хоть и уступает первой по производительности, но почти не уступает СХД AVRORA. Таким образом, имеет смысл дальнейшая оптимизация обеих реализаций.

Дальнейшие планы

При использовании регистров большего размера, за один проход по дискам можно обработать больше данных на дисках. В матричном алгоритме такая оптимизация увеличивает производительность RAID 6. Поэтому в дальнейшем планируется добавить реализации, которые используют инструкции и регистры по технологии процессора AVX, в которой размер регистров увеличен вдвое по сравнению с SSE.

Вероятность отказа трёх дисков растёт с увеличением числа дисковых массивов. Поэтому иногда возникает необходимость восстановления трех дисков с данными. Это возможно, если хранить три контрольные суммы. Ввиду этого планируется обеспечить в будущем такую возможность на основе уже реализованных нами алгоритмов.

Литература

1. System V Application Binary Interface AMD64 Architecture Processor Supplement (Draft Version 0.99.5).
2. *Утешев А. Ю.* Математика отказоустойчивых дисковых массивов. http://pmpu.ru/vf4/codes/raid
3. *H. Peter Anvin*. The mathematics of RAID-6. 2006–2011. http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf
4. *Утешев А. Ю.* Поля Галуа. http://pmpu.ru/vf4/gruppe/galois
5. *Федоров А. Р.* Задача восстановления утерянных дисков в массиве с использованием арифметики конечных колец.

СОДЕРЖАНИЕ

Научное издание

СПИСОК-2012

Материалы
всероссийской научной конференции
по проблемам информатики

25–27 апр. 2012 г.,
Санкт-Петербург

Компьютерная верстка: *О. В. Шакиров*

Издательство «ВВМ»
190000, Санкт-Петербург,
ул. Декабристов, 6, лит. А, пом. 10н
E-mail: vvmpub@yandex.ru

Подписано к печати 27.09.12. Формат $60 \times 90\ ^1/_{16}$. Бумага офсетная.
Гарнитура Таймс. Печать цифровая. Печ. л. 32,5. Тираж 300 экз.
Заказ 5567

Отпечатано в Отделе оперативной полиграфии
химического факультета СПбГУ
198504, Санкт-Петербург, Старый Петергоф, Университетский пр., 26
Тел.: (812) 428-4043, 428-6919