Статья опубликована в сборнике докладов X Международной конференции по мягким вычислениям и измерениям. Том 2. СПбГЭТУ «ЛЭТИ». 2007, с. 84–87.

РАЗРАБОТКА БИБЛИОТЕКИ ДЛЯ ГЕНЕРАЦИИ УПРАВЛЯЮЩИХ АВТОМАТОВ МЕТОДОМ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Н. И. Поликарпова, В. Н. Точилин, А. А. Шалыто
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Abstract. Control devices, network protocols and many other software and hardware systems have *complex behavior*. It is convenient to describe an entity with complex behavior as a pair of a *control automaton* and a *control object*. However, building a control automaton heuristically is often hard (or even impossible). We propose to use *genetic programming* to solve this problem. In this work we present a method for building control automaton basing on given control object and *fitness* function. We describe automata representation that is suitable for genetic optimization and adopt *crossover* and *mutation* mechanism. We also suggest a possible solution to the problem of exponential growth of automaton description with increase of its input size. Concrete implementations of the proposed data structures and algorithms are united into the *AutoGen* framework – a library for generating control automata.

Введение

В последнее время для описания сущностей со сложным поведением в программировании предлагается использовать *автоматный подход* [1]. В соответствии с этим подходом сущность представляется в виде *автоматизированного объекта* – совокупности *управляющего автомата* и *объекта управления* (ОУ).

Объект управления характеризуется множеством *вычислительных состояний*, а также двумя наборами функций: множеством *предикатов*, отображающих вычислительные состояния в логические значения (истина или ложь), и множеством *действий*, позволяющих изменять вычислительное состояние.

Управляющий автомат определяется конечным множеством *управляющих состояний*, *функцией переходов* и *функцией выходов*. Входными и выходными переменными автомата являются, соответственно, значения предикатов и номера действий ОУ.

При использовании автоматного подхода поведение ОУ при его проектировании можно сделать несложным: его предикаты и действия могут быть реализованы как простые, короткие функции, которые практически не содержат ветвлений. При этом вся «логика» оказывается сосредоточенной в управляющем автомате.

Описание логики в форме диаграммы переходов конечного автомата хорошо структурировано, и разработан ряд методов его декомпозиции [1, 2]. Несмотря на это, опыт показывает, что построение управляющего автомата обычно требует от разработчика гораздо больше усилий и порождает больше ошибок, чем реализация ОУ. Для некоторых задач эвристическое построение автомата или невозможно, или затруднительно. Кроме того, даже для простых задач автомат, построенный вручную, часто является неоптимальным.

Эту проблему авторы предлагают решать методами *генетического программирования* (ГП) [3]. При этом автоматы генерируются автоматически, что существенно упрощает построение автоматных программ и позволяет повысить уровень автоматизации их проектирования.

Основная идея ГП состоит в построении программ путем применения генетических алгоритмов к некоторой *модели вычислений*. При этом для каждой модели достаточно *один раз* адаптировать генетический алгоритм. В дальнейшем разработчику программы остается задать *оценочную функцию*, которая определяет численное значение *пригодности* (*fitness*) для каждого возможного результата вычисления с использованием выбранной модели.

В качестве моделей вычислений в ГП чаще всего используют «низкоуровневые» модели (команды процессора, специализированные деревья, графы), которые имеют ограниченный набор элементарных операций (таких как, например, запись и чтение ячеек памяти, арифметические операции, вызовы подпрограмм и т.д.). Достоинство низкоуровневых моделей состоит в универсальности: с их помощью можно построить любую программу целиком и единообразно, вне зависимости от специфики решаемой задачи. Однако, такие модели обладают и серьезными недостатками. Во-первых, построенная автоматически программа из-за отсутствия высокоуровневой структуры редко бывает понятна человеку. Во-вторых, из-за того, что пространство допустимых программ в этом случае очень велико, генетическая оптимизация может потребовать большого времени.

Устранить указанные недостатки можно путем перехода к высокоуровневым моделям вычислений. Для этого в настоящей работе предлагается объединить автоматный и генетический подходы. В качестве оптимизируемой модели вычислений предлагается использовать «автоматизированный объект». При этом реализация объекта управления производится вручную, а генетический алгоритм применяется для автоматического построения управляющего автомата. Автомат – «высокоуровневый» вычислитель, так как его элементарные операции (предикаты и действия) специфичны для конкретной задачи. Поэтому предлагаемый подход лишен указанных недостатков.

1. Состояние вопроса

Эволюционной оптимизации моделей вычислений в виде конечных автоматов были посвящены многие исследования. При этом большинство авторов занималось оптимизацией автоматов-*распознавателей* (*parsers*) и *преобразователей* (*transducers*).

Собственно ГП было впервые упомянуто в книге [3]. Первоначально предлагалось представлять программу в виде дерева. В работе [4] впервые оптимизировалась модель в форме графа, который можно интерпретировать как диаграмму переходов конечного автомата.

В работе [5] отмечается необходимость расширения ГП для обеспечения возможности использования сложных структур данных. В ряде последующих работ рассмотрено применение различных структур данных. Отметим, что подход, предлагаемый в настоящей работе, позволяет работать с произвольными структурами данных, используя в составе автоматизированного объекта ОУ любой сложности.

Практически во всех исследованиях по этой теме автомат в каждый момент времени обрабатывает только одну входную переменную (исключения составляют лишь несколько работ, в которых допускается фиксированное небольшое число входов). Теоретически, любое количество параллельных входов сводится к одному, однако при этом размер входного алфавита растет экспоненциально.

Также в рассмотренных работах автомат на каждом шаге может выполнить не более одного действия из заданного множества. В таком случае любая комбинация действий, которые могут быть выполнены одновременно, также должна считаться элементарным действием, что приводит к экспоненциальному росту количества действий.

Из всех перечисленных работ наиболее близкие к рассматриваемой теме результаты получены в статьях [6, 7], в которых описано создание управляющей программы робота. Однако и эти работы не лишены упомянутых выше недостатков, относящихся к входным и выходным воздействиям.

2. Постановка задачи

Задача построения управляющего автомата состоит в том, чтобы найти

автомат заданного вида, такой, что за фиксированное число шагов работы под управлением этого автомата заданный ОУ перейдет в вычислительное состояние с максимальной пригодностью.

В связи с использованием ГП для генерации управляющих автоматов возникают две основные задачи: выбор представления автомата в виде *особи* и адаптация генетических операторов (*мутации* и *скрещивания*) для выбранного представления.

3. Способы представления автомата и адаптация генетических операторов

В классической интерпретации генетического алгоритма особь представляется в виде набора *хромосом*. Управляющий автомат можно представить как набор состояний, в каждом из которых его поведение определяется сужением функций переходов и выходов. Таким образом, удобно сопоставить каждому состоянию хромосому.

Далее необходимо выбрать удобное представление для хромосом состояний.

Естественный способ записи таких хромосом – это табличное представление функций переходов и выходов. Если ОУ имеет n предикатов и m действий, то таблица содержит 2^n строк (по одной для каждой возможной комбинации значений предикатов) и $m+1$ столбцов, в первом из которых записано значение функции переходов (номера целевых состояний), а совокупность остальных столбцов соответствует множеству действий, которые необходимо выполнить при переходах. Таблицу такого вида будем называть *полной таблицей состояния*.

При мутации состояния с некоторой вероятностью может измениться каждый элемент таблицы. При скрещивании полных таблиц предлагается по очереди выполнять традиционное одноточечное скрещивание соответствующих столбцов этих таблиц.

Основная проблема, возникающая при использовании полных таблиц состояний – экспоненциальный рост размерности хромосомы с увеличением числа предикатов ОУ.

Опыт показывает, что в реальных задачах управляющие автоматы, построенные вручную, имеют не так много переходов. Причина этого, по мнению авторов, состоит в том, что в большинстве задач предикаты имеют «локальную природу» по отношению к управляющим состояниям. В каждом состоянии *значимым* является лишь определенный, небольшой набор предикатов, остальные же предикаты не влияют на значения функций переходов и выходов. Это свойство позволяет существенно сократить размер описания каждого состояния.

Один из подходов к сокращению размера хромосомы каждого состояния – ограничить некоторой константой r количество значимых в состоянии предикатов. В этом случае к описанию хромосомы добавляется битовый вектор, описывающий множество значимых предикатов. Полученное представление хромосомы назовем *сокращенной таблицей состояния*.

Число строк сокращенной таблицы 2^r, однако, константа r обычно невелика. Как показывает опыт, для большинства автоматизированных объектов она не превышает нескольких единиц.

По сравнению с представлением в виде полной таблицы, в этом случае добавилась возможность мутации множества значимых предикатов. При этом каждый из них с некоторой вероятностью заменяется другим предикатом, не принадлежащим указанному множеству. Мутация сокращенной таблицы происходит так же, как и мутация полной таблицы.

Скрещивание сокращенных таблиц – наиболее сложный из используемых алгоритмов. Поскольку родительские хромосомы, представленные сокращенными таблицами, могут иметь разные множества значимых предикатов, сначала необходимо выбрать, какие из этих предикатов будут значимы для хромосом детей.

После этого заполняются таблицы обоих детей. В предлагаемой реализации оператора скрещивания на значения каждой строки таблицы ребенка влияют значения нескольких строк родительских таблиц. При этом конкретное значение, помещаемое в ячейку таблицы ребенка, определяется «голосованием» всех влияющих на нее ячеек родительских таблиц.

Предлагаемые алгоритмы были реализованы авторами в библиотеке *AutoGen*, которая была апробирована на решении ряда задач, в том числе, и такой сложной, как «Электрические джунгли» (http://is.ifmo.ru/elejungle/). Эта библиотека будет опубликована на сайте http://is.ifmo.ru в разделе «Генетические алгоритмы». Библиотека и подход в целом показали свою эффективность.

Заключение

В работе предложены методы использования ГП для построения управляющих автоматов, в которых отсутствуют ограничения на число входов и выходов.

Авторы предполагают, что имеются широкие возможности для дальнейшего развития генетического подхода к построению управляющих автоматов.

Во-первых, необходимо изучить и реализовать другие решения проблемы экспоненциального роста размерности хромосомы состояния.

Во-вторых, необходимо рассмотреть проблему «долгого» вычисления оценочной функции. Каждое такое вычисление – это эмуляция определенного числа шагов работы заданного ОУ совместно с оцениваемым автоматом. Такая эмуляция требует больших временных затрат по сравнению с другими этапами генетической оптимизации.

В-третьих, можно рассмотреть различные более общие постановки задачи построения систем со сложным поведением.

И, наконец, необходимо рассмотреть подходы к представлению результатов генетической оптимизации в виде, более понятном человеку. Например, для сложных задач необходимо генерировать не один автомат, а систему взаимодействующих автоматов.

Литература

1. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука. 1998.
2. *Harel D.*, *Polity M.* Modeling Reactive Systems with Statecharts. The Statemate Approach. New York: McGraw-Hill. 1998.
3. *Koza J.* Genetic Programming: On the programming of Computers by Means of Natural Selection. Cambridge: MIT Press, 1992.
4. *Teller A., Veloso M.* PADO: A New Learning Architecture for Object Recognition / Symbolic Visual Learning. New York: Oxford University Press, 1996, pp.81–116.
5. *Koza J. R.* Future Work and Practical Applications of Genetic Programming. Handbook of Evolutionary Computation. Bristol: IOP Publishing Ltd, 1997.
6. *Petrovic P.* Simulated evolution of distributed FSA behaviour-based arbitration / The Eighth Scandinavian Conference on Artificial Intelligence (SCAI'03). 2003.
7. *Petrovic P.* Evolving automatons for distributed behavior arbitration. Technical Report. Norwegian University of Science and Technology. 2005.