ПОСТРОЕНИЕ КОНЕЧНЫХ АВТОМАТОВ НА ОСНОВЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ И ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Царев Ф.Н., Шалыто А.А.
Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, г. Санкт-Петербург
tsarev@rain.ifmo.ru, shalyto@mail.ifmo.ru

Аннотация: в последнее время для решения некоторых классов задач управления достаточно часто применяется автоматное программирование – парадигма программирования, при использовании которой программу предлагается строить в виде совокупности автоматизированных объектов управления, каждый из которых содержит систему управления (один или несколько взаимодействующих конечных автомата) и объект управления. Основная сложность при применении автоматного программирования состоит в построение конечных автоматов. В ряде задач автоматы удается построить эвристическими методами, однако часто такое построение требует больших затрат времени, а построенные автоматы не оптимальны. Полный перебор всех автоматов крайне трудоемок, поэтому для построения автоматов в задачах рассматриваемого класса целесообразно применять эволюционные алгоритмы, в частности, генетические алгоритмы и генетическое программирование. В работе приводятся результаты исследований в области построения конечных автоматов на основе генетических алгоритмов и генетического программирования.

Ключевые слова: автоматное программирование, генетические алгоритмы, генетическое программирование.

Введение

В последнее время для решения некоторых классов задач достаточно часто применяется автоматное программирование [1, 2] – парадигма программирования, при использовании которой программу предлагается строить в виде совокупности автоматизированных объектов управления, каждый из которых содержит систему управления (один или несколько взаимодействующих конечных автомата) и объект управления.

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

Управляющий автомат определяется конечным множеством управляющих состояний, функцией переходов и функцией действий. Взаимодействие между автоматами может осуществляться различными способами: за счет вложенности автоматов, с помощью обмена сообщениями, с помощью обмена номерами состояний и т. д.

При использовании автоматного программирования существенно упрощается (по сравнению с программами, написанными традиционными методами) верификация программ с использованием метода *Model checking* [3 – 6], так как построение модели Крипке по автоматной программе может быть автоматизировано [5]. Кроме этого, при использовании инструментальных средств для поддержки автоматного программирования таких, как, например, *UniMod* [7], более 60% исходного кода программы могут быть сгенерированы автоматически [1, 8].

Основная сложность при применении автоматного программирования состоит в построение конечных автоматов. В ряде задач автоматы удается построить эвристическими методами, однако часто такое построение требует больших затрат времени, а построенные автоматы не оптимальны. Примерами таких задач являются: управление командой беспилотных летательных аппаратов [9, 10] в соревнованиях с другой командой, итерированная дилемма узника [11], задача о «Флибах» [12, 13], задача «Умный муравей» [14]. Полный перебор всех автоматов

крайне трудоемок, а эвристическое построение не всегда дает приемлемые результаты. Поэтому для построения автоматов в задачах такого рода целесообразно применять эволюционные алгоритмы [15], в частности генетические алгоритмы [16 – 19] и генетическое программирование [20].

Генетические алгоритмы являются одним из современных и развивающихся направлений в искусственном интеллекте. С их помощью могут быть найдены решения многих задач в различных областях.

Генетическое программирование – разновидность генетических алгоритмов, в которой вместо низкоуровневого представления объектов в виде битовых строк используется высокоуровневое представление: деревья разбора программ, диаграммы переходов конечных автоматов и т. д. С помощью генетического программирования наиболее эффективно решаются задачи автоматического построения программ, клеточных автоматов и конечных автоматов.

Применение указанных методов повышает уровень автоматизации построения автоматных программ (с учетом сказанного выше, более 60% исходного кода программ могут быть построено автоматически), снижает влияние человеческого фактора на их качество и является одним из шагов к автоматическому построению программ.

Методы построения конечных автоматов с помощью генетических алгоритмов и генетического программирования можно подразделить на три типа:

- методы, использующие моделирование для вычисления функции приспособленности;
- методы, использующие обучающие примеры (тесты) для вычисления функции приспособленности;
- методы, использующие верификацию при вычислении функции приспособленности.

1. Методы, использующие моделирование для вычисления функции приспособленности

В настоящем разделе описываются методы построения конечных автоматов, использующие моделирование для вычисления функции приспособленности.

1.1. Методы, описанные в работах других авторов

Один из создателей эволюционного программирования Л. Фогель рассматривал интеллектуальное поведение индивида, как способность успешно предсказывать поведение среды, в которой он находится, и сообразно с этим действовать. В 60-х прошлого века годах Фогель поставил ряд экспериментов [13] по созданию искусственных систем, способных адаптироваться к первоначально не известной им среде.

В проведенных экспериментах Л. Фогель моделировал поведение простейшего живого существа, которое способно предсказывать изменения параметра среды, обладающего периодичностью. Это живое существо моделировалось конечным автоматом с действиями на переходах – автоматом Мили. В качестве среды выступала последовательность символов над двоичным алфавитом, например, (1111010010)*. На вход автомата в каждый момент времени подавалась битовое значение параметра окружающей среды. Автомат формировал значение выходной переменной – возможное значение рассматриваемого параметра в следующий момент времени.

Задача состояла в эволюционном построении автомата, способного как можно более точно в смысле какой-либо разумной функции приспособленности от входа и выхода (например, количества совпавших символов) предсказывать среду – угадывать следующий символ последовательности. Кроме того, Фогель накладывал ограничения на сложность порождаемого автомата, так как строить автоматы равные длине обозреваемой последовательности не представляет труда. Таким образом, предпочтение отдавалось автоматам, угадывающим как можно лучше, и в то же время имеющим как можно меньше состояний.

В начале эксперимента задавалась периодическая последовательность символов над двоичным алфавитом, например (101110011101)*. На начальной фазе выбирался префикс данной последовательности малой длины. После этого создавалась популяция автоматов с небольшим числом состояний (около пяти). Затем каждый из автоматов путем одной из пяти мутаций (выбираемой случайным образом равномерно) производил потомка. При этом были допустимы следующие мутации автомата:

- добавление состояния;
- удаление состояния (в случае, если число состояний больше единицы);
- замена стартового состояния (в случае, если число состояний больше единицы);
- замена перехода;
- замена действия на переходе.

Далее над потомком подобным образом производилось еще несколько мутаций (их число определялось случайным образом). Получившийся в результате мутаций автомат добавлялся в популяцию.

После добавления потомков в популяцию на всех ее особях вычислялась функция приспособленности. Половина наиболее приспособленных особей переносилась в популяцию следующего поколения, а менее приспособленные автоматы – отбрасывались. Таким образом, размер популяции оставался постоянным (стоит отметить, что в силу ограниченных возможностей компьютеров того времени, он был мал – всего несколько особей). Данный процесс продолжался до тех пор, пока не удавалось достигнуть желаемого результата – максимальное значение функции приспособленности не превосходило заданного порога. После этого к битовой последовательности, которая определяла среду, добавлялся очередной символ, и эволюционный процесс переходил в очередную фазу.

По мнению Фогеля, результаты экспериментов показали, что эволюционное алгоритмы могут быть успешно применены для построения «интеллектуальных» искусственных систем. При этом было отмечено, что построение вручную автоматов столь же результативных и простых, как те, что были построены эволюционным алгоритмом, является крайне сложной задачей.

Еще одной задачей, в которой построенные с помощью генетических алгоритмов автоматы обладают подобными свойствами, является задача «Умный муравей» [14, 21].

Приведем ее описание. Используется двумерный тор размером 32 на 32 клетки (рис. 1).

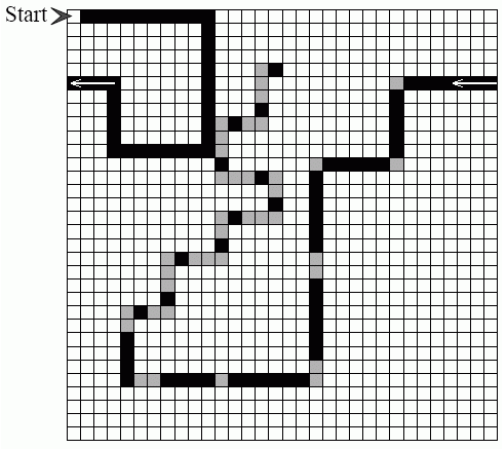

Рис. 1. Поле в задаче «Умный муравей»

На некоторых клетках поля расположены яблоки – черные клетки. Яблоки расположены вдоль некоторой ломаной линии, но не на всех ее клетках. Клетки ломаной, на которых яблок нет – серые. Белые клетки – не принадлежат ломаной и не содержат яблок. Всего на поле 89 яблок.

В клетке с пометкой «Start» находится муравей. Он занимает клетку поля и смотрит в одном из четырех направлений (север, запад, юг, восток). В начале игры муравей смотрит на восток. Он умеет определять находится ли яблоко непосредственно перед ним. За ход муравей совершает одно из четырех действий:

- идет вперед на одну клетку, съедая яблоко, если оно находится перед ним;
- поворачивается вправо;
- поворачивается влево;
- стоит на месте.

Съеденные муравьем яблоки не восполняются. Муравей жив на всем протяжении игры – еда не является необходимым ресурсом для его существования. Никаких других персонажей, кроме муравья, на поле нет. Ломаная *строго задана*. Муравей может ходить по любым клеткам поля. Игра длится 200 ходов, на каждом из которых муравей совершает одно из четырех описанных выше действий. В конце игры подсчитывается число яблок, съеденных муравьем. Это значение – результат игры.

Цель игры – создать муравья, который за 200 ходов съест как можно больше яблок. Муравьи, съевшие одинаковое число яблок, заканчивают игру с одинаковым результатом вне зависимости от числа ходов, затраченных каждым из них на процесс еды. Однако эта задача может иметь различные модификации, например, такую, в которой при одинаковом числе съеденных яблок, лучшим считается муравей, съевший яблоки за меньшее число ходов. Ниже будет показано, что поведение муравья может быть задано конечным автоматом. При этом может быть поставлена задача о построении автомата с минимальным числом состояний для муравья, съедающего все яблоки, или автомата для муравья, съедающего максимальное число яблок при заданном числе состояний.

Вручную сравнительно несложно построить автомат, описывающий следующую стратегию: «Вижу яблоко – иду вперед. Не вижу – поворачиваю. Сделал круг, но яблок нет – иду вперед». Он позволяет муравью съесть 81 яблоко за 200 ходов, а все 89 – только за 314 ходов.

В работе [14] с помощью генетических алгоритмов построен автомат из 13 состояний, который позволяет муравью съесть все яблоки за 200 ходов. В работе [14] приспособленность особи (*fitness*) определяется как число яблок, съеденное за 200 ходов. Кодирование автомата, задающего поведение муравья, в особь генетического алгоритма (битовую строку) в этой работе осуществлялось следующим образом: входному воздействию F сопоставлялась единица, а N – ноль. Каждое из четырех действий муравья кодировалось двоичными числами: 00 – N (стоять на месте), 01 – L (повернуть налево), 10 – R (повернуть направо), 11 – M (идти вперед). В работе [21] постороен автомат из 11 состояний, который позволяет съесть все яблоки за 193 хода.

В работе [22] рассматривается задача построения автоматов для игры «Соревнование за ресурсы» («Competition for Resources»). Игровое поле в этой игре имеет форму тора размером 10 на 10 клеток (тор можно представить в виде квадрата, у которого верхняя сторона склеена с нижней, а правая – с левой). Оно моделирует компьютерную сеть, в которой могут распространяться программные агенты. В рассматриваемой работе стратегия поведения второго агента была задана, а цель состояла в построении конечного автомата для управления первым агентом, который позволял бы достаточно часто выигрывать игру. Для решения этой задачи применялся генетический алгоритм, в котором конечный автомат был представлен в виде таблицы значений функции переходов и таблицы значений функции выходов.

1.2. Методы, разработанные в НИУ ИТМО

В рамках исследований, проводимых на кафедре «Технологии программирования» *НИУ ИТМО,* был разработан ряд методов генерации конечных автоматов с помощью генетических алгоритмов [23 – 27].

В работе [28] рассматривается задача «Умный муравей». Для ее решения предлагается применять генетическое программирование – в отличие от указанных выше работ в генетическом алгоритме автомат представляется не в виде битовой строки, а в виде графа переходов. Кроме этого, существенное отличие предлагаемого в этой работе алгоритма состоит в исполь-

зовании двух операторов скрещивания – традиционного и сохраняющего значимые части автоматов [29].

За счет применения такого алгоритма удается построить автомат, содержащий семь состояний и позволяющий муравью съесть все 89 яблок за число шагов. Не превышающее заданное. Граф переходов этого автомата приведен на рис. 2. При построении этого автомата функция приспособленности была вычислена порядка 130 миллионов раз, в то время как полный перебор в этом случае связан со значительно большим числом вычислений.

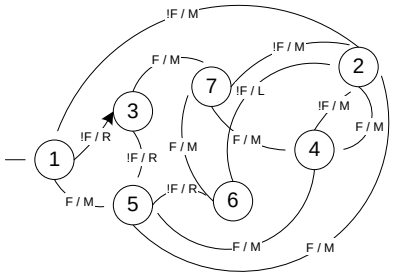

Рис. 2. Граф переходов автомата из семи состояний, решающего задачу «Умный муравей»

Кроме этого, в работе [28] описывается алгоритм перебора, с помощью которого было установлено, что автоматы с шестью и менее состояниями задачу «Умный муравей» не решают.

В работах [30, 31] предложен новый метод представления автоматов деревьями решений, который состоит в следующем: функции переходов и выходов автомата выражаются с помощью деревьев решений. Более формально: зададим для каждого состояния $q \in Q$ функцию $\sigma_q : X \to Q \times Y$, такую что $\sigma_q(x) = (\delta(q,x), \lambda(q,x))$ для $\forall x \in X$. Здесь Q – множество состояний автомата, X – множество входных воздействий, Y – множество выходных воздействий, $\delta : Q \times X \to Q$ – функция переходов, $\lambda : Q \times X \to Y$ – функция выхода. Каждая из этих функций может быть определена собственным деревом решений. Таким образом, автомат в целом может быть представлен упорядоченным набором деревьев решений и начальным состоянием.

В работе [32] предложен еще один метод – представления конечных автоматов с помощью сокращенных таблиц переходов. Этот метод позволяет решить проблему экспоненциального роста размера описания автомата, которая возникает при использовании полных таблиц переходов – число строк в таблице равно 2^n, где n – число предикатов. Опыт показывает, что в реальных задачах управляющие автоматы, построенные вручную, имеют гораздо меньше переходов, чем $|S| \cdot 2^n$ (здесь |S| – число состояний автомата). Причина этого состоит в том, что в большинстве задач предикаты имеют «локальную природу» по отношению к управляющим состояниям. В каждом состоянии *значимым* является лишь определенный, небольшой поднабор предикатов, остальные же состояния не влияют на значение управляющей функции. Именно это свойство позволяет существенно сократить размер описания состояний. Кроме этого, использование укавзанного свойства в процессе оптимизации позволяет получить результат, более похожий на автомат, построенный вручную, а, следовательно, и более понятный человеку.

В этой же работе описывается применение этого метода для построения автомата верхнего уровня для управления моделью беспилотного самолета.

2. Методы, использующие обучающие примеры (тесты) для вычисления функции приспособленности

2.1. Методы, описанные в работах других авторов

В работах [33, 34] производится сравнение алгоритмов машинного обучения для построения конечных автоматов-распознавателей. При этом рассматриваются два типа алгоритмов: эволюционные и основанные на построении префиксного дерева (бора) и дальнейшем объединении состояний на основе свидетельств (*Evidence-Driven State Merging*, далее *EDSM*).

Входными данными для алгоритма построения конечного автомата-распознавателя является набор слов, для каждого из которых известно, должно ли оно допускаться автоматом или не должно. Этот набор слов в дальнейшем будет называться множеством обучающих примеров. Выходными данными для рассматриваемого алгоритма является описание построенного конечного автомата, который удовлетворяет входным данным.

В качестве эволюционного алгоритма в рассматриваемых работах применяется так называемая (1+1)-эволюционная стратегия. На основании проведенных сравнительных экспериментов был сделан вывод, что разработанный эволюционный алгоритм превосходит алгоритм *EDSM* в том случае, когда необходимо построить автомат с относительно небольшим числом состояний, а обучающее множество примеров содержит небольшую долю строк заданной длины.

В работе [35] автоматы представлялись с помощью таблицы переходов. Разработанный в этой работе метод позволяет поддерживать в популяции автоматы с разным числом состояний. Функция приспособленности учитывает три компонента – число верно распознанных тестовых примеров, число состояний и переходов автомата, степень общности языка, соответствующего построенному автомату. Это позволяет уменьшить область поиска, и находить языки, соответствующие определенным критериям. В обучающий набор могут входить примеры слов, которые как принадлежат, так и не принадлежат языку.

Предложенный подход был проверен на практических задачах. Авторам удалось построить автомат из семи состояний, который распознает по множеству из ста примеров русские двухсложные слова.

В работе [36] также рассматривается задача построения конечных автоматов-распознавателей. В качестве метода представления конечных автоматов выбраны двоичные строки, в качестве операций мутации и скрещивания применяются однородное скрещивание и мутация. В качестве исходных данных для генетического алгоритма выступают пары, состоящие из входных слов и последовательностей пометок состояний, которые используются при обработке соответствующих слов.

По построению конечных преобразователей на основе обучающих примеров существует существенно меньше работ, чем по построению конечных распознавателей. Одной из работ по конечным автоматам-распознавателям является работа [37]. Такие автоматы при обработке символов входного слова при выполнении переходов записывают в выходной поток символы выходного алфавита, тем самым осуществляя преобразование слов одного формального языка в слова другого.

Так же, как и в рассмотренных выше работах [33, 34], посвященных построению конечных автоматов-распознавателей, в работе [37] применяется (1+1)-эволюционная стратегия. Конечный преобразователь представляется в виде набора двух таблиц – таблицы значений функции переходов и таблицы значений функции выходов. При этом предполагалось, что на каждом из переходов конечный преобразователь может вывести не более одного символа.

Входными данными для построения конечного автомата-распознавателя является набор обучающих примеров – пар слов, одно из которых является входным, а второе – соответствующим ему выходным.

В рассматриваемой работе рассматриваются три варианта функции приспособленности: на основе строгого сравнения, на основе вычисления расстояния Хэмминга [38], на основе вычисления редакционного расстояний (расстояния Левенштейна) [39].

2.2. Методы, разработанные в НИУ ИТМО

Методы построения конечных автоматов, использующие моделирование для вычисления функции приспособленности (описаны в разд. 1), обладают следующими основными недостатками:

- для решения новой задачи необходимо заново выполнять программную реализацию моделирования для вычисления функции приспособленности, что может быть весьма трудоемким;
- каждое вычисление функции приспособленности может требовать значительного времени, так как связано с моделированием поведения автомата в некоторой внешней среде.

Для устранения этих недостатков в работах [40, 41] был предложен метод построения автоматов с помощью генетического программирования на основе обучающих примеров (тестов). Исходными данными для построения конечного автомата управления системой со сложным поведением, в которой на одни и те же входные воздействия в зависимости от режима работы генерируются различные выходные воздействия, являются:

- список событий;
- список входных переменных;
- список выходных воздействий;
- набор тестов *Tests*, каждый из которых содержит последовательность *Input[i]* событий, поступающих на вход конечному автомату, и соответствующую ей эталонную последовательность *Answer[i]* выходных воздействий.

Важными особенностями применямого алгоритма генетического программирования являются: использование алгоритма расстановки выходных воздействий для сокращения пространства поиска, применение двух типов оператора скрещивания – традиционного и основанного на тестах, а также вычисление функции приспособленности на основе редакционного расстояния.

Алгоритм расстановки выходных воздействий и представление конечного автомата в виде хромосомы генетического алгоритма. Конечный автомат в алгоритме генетического программирования представляется в виде объекта, который содержит описания переходов для каждого состояния и номер начального состояния. Для каждого состояния хранится список переходов. Каждый переход описывается событием, при поступлении которого этот переход выполняется, и числом выходных воздействий, которые должны быть сгенерированы при выборе этого перехода.

Таким образом, в особи кодируется только «скелет» управляющего конечного автомата, а конкретные выходные воздействия, вырабатываемые на переходах, определяются с помощью алгоритма расстановки пометок. Выбор представления графа переходов автомата с помощью списков ребер обоснован тем, что, как правило, в автоматах управления системами со сложным поведением не в каждом состоянии определена реакция на каждое событие.

Опишем **алгоритм расстановки пометок на переходах**. Как было сказано выше, для каждого перехода в особи генетического алгоритма записано, сколько выходных воздействий должно вырабатываться при его выборе. Подадим на вход конечному автомату последовательность событий, соответствующую одному из тестов, и будем наблюдать за тем, какие переходы выполняет автомат. Зная эти переходы и информацию о том, сколько выходных воздействий должно быть сгенерировано на каждом переходе, можно определить, какие выходные воздействия должны вырабатываться на переходах, использовавшихся при обработке входной последовательности.

Для каждого перехода T и каждой последовательности выходных воздействий zs вычисляется величина $C[T][zs]$ – число раз, когда при обработке входной последовательности, соответствующей одному из тестов, на переходе T должны быть выработаны выходные воздействия, образующие последовательность zs. Далее, каждый переход помечается той последовательностью zs_0, для которой величина $C[T][zs_0]$ максимальна.

Функция приспособленности основана на редакционном расстоянии. Редакционным расстоянием между двумя последовательностями символов называется минимальное число опера-

ций замены символа, вставки символа и удаления символа, которые необходимо выполнить над первой последовательностью для того, чтобы она совпала со второй.

Для вычисления функции приспособленности выполняются следующие действия: на вход автомату подается каждая из последовательностей Input[i]. Обозначим последовательность выходных воздействий, которую сгенерировал автомат при входе Input[i], как Output[i]. После этого вычисляется величина

$$\mathrm{FF}_1 = \frac{\sum_{i=1}^{n}\left(1-\frac{\mathrm{ED}(\mathrm{Output}[i],\mathrm{Answer}[i])}{\max(|\mathrm{Output}[i]|,|\mathrm{Answer}[i]|)}\right)}{n},$$

где ED(A, B) – редакционное расстояние между строками A и B. Отметим, что значения функции FF_1 лежат в пределах от 0 до 1. При этом, чем «лучше» автомат соответствует тестам, тем больше значение функции приспособленности.

Функция приспособленности зависит не только от того, насколько «хорошо» автомат работает на тестах, но и от числа переходов, которые он содержит. Эта функция вычисляется следующим образом:

$$\mathrm{FF}_2 = \begin{cases} 0.5 \cdot T \cdot \mathrm{FF}_1 + \frac{1}{M} \cdot (\mathrm{M} - \mathrm{cnt}), \mathrm{FF}_1 < 1 \\ T + \frac{1}{M} \cdot (M - \mathrm{cnt}), \mathrm{FF}_1 = 1 \end{cases}$$

Здесь *cnt* – число переходов в автомате, T – «стоимость» прохождения всех тестов, M – произвольное целое число, большее максимального числа переходов в автомате. При проведении вычислительных экспериментов были выбраны следующие значения: $T = 20$, $M = 100$.

Эта функция приспособленности устроена таким образом, что при одинаковом значении функции FF_1, отражающей «прохождение» тестов автоматом, преимущество имеет автомат, содержащий меньшее число переходов. Кроме этого, автомат, который «идеально» проходит все тесты, оценивается выше, чем автомат, проходящий тесты не идеально.

Операция скрещивания. Скрещивание описаний автоматов производится следующим образом. Обозначим как *P1* и *P2* – «родительские» особи, а *S1* и *S2* – особи-«потомки». Для начальных состояний *S1.is* и *S2.is* автоматов *S1* и *S2* будет верно одно из двух соотношений:

- *S1.is* = *P1.is* и *S2.is* = *P2.is*;
- *S1.is* = *P2.is* и *S2.is* = *P1.is*.

Опишем, как устроены переходы автоматов *S1* и *S2*. Скрещивание описаний автоматов производится отдельно для каждого состояния. Обозначим список переходов из состояния номер i автомата *P1* как *P1.T*[i], а список переходов из состояния номер i автомата *P2* как *P2.T*[i]. Для выполнения «скрещивания переходов» с равной вероятностью может быть выбран один из двух методов.

При использовании *традиционного метода скрещивания* списки переходов *S1.T*[i] и *S2.T*[i] строятся следующим образом:

1. Строится общий список переходов, в который помещаются переходы, входящие как в *P1.T*[i], так и в *P2.T*[i].
2. К полученному списку применяется случайная перестановка.
3. Далее возможны два равновероятных варианта:
   - либо в *S1.T*[i] помещаются первые |*P1.T*[i]| переходов из полученного списка, а в *S2.T*[i] – оставшиеся переходы;
   - либо в *S1.T*[i] помещаются первые |*P2.T*[i]| переходов из полученного списка, а в S2.*T*[i] – оставшиеся переходы.

При использовании *метода скрещивания с учетом тестов* списки переходов *S1.T*[i] и *S2.T*[i] строятся следующим образом:

1. Составляется список всех используемых тестов, упорядоченный по возрастанию нормированного редакционного расстояние между «правильным ответом» Answer и

последовательностью Output выходных воздействий, генерируемой автоматом, – значению выражения $\frac{ED(Output[i], Answer[i])}{\max(|Output[i]|, |Answer[i]|)}$. В автоматах *P1* и *P2* помечаются те переходы, которые используются при обработке первых 10% тестов из полученного упорядоченного списка.

2. *Помеченные* переходы копируются в *S1.T*[*i*] и *S2.T*[*i*] напрямую.
3. Строится общий список переходов, в который помещаются *непомеченные* переходы, входящие как в *P1.T*[*i*], так и в *P2.T*[*i*].
4. К полученному списку *L* применяется случайная перестановка.
5. Список *S1.T*[*i*] дополняется первыми переходами из списка *L* до размера |*P1.T*[*i*]|, а список *S2.T*[*i*] дополняется оставшимися переходами.

В обоих случаях к получившимся в результате скрещивания автоматам *S1* и *S2* применяется операция удаления дублирующихся переходов.

Пример применения предлагаемого метода. Применение предлагаемого метода иллюстрируется на примере построения автомата управления часами с будильником [2]. Эти часы имеют три кнопки (помеченные буквами A, H, M), которые предназначены для изменения режима их работы и для настройки текущего времени или времени срабатывания будильника. Если будильник выключен, то кнопки H и M служат для установки текущего времени, а кнопка A включает его и переводит часы в режим «Настройка будильника», в котором кнопки H и M устанавливают не текущее время, а время срабатывания будильника. Повторное нажатие кнопки A включает будильник. После этого, если текущее время совпадает со временем срабатывания будильника, то включается звонок, который отключается либо нажатием кнопки A, либо самопроизвольно через минуту. Кроме этого, нажатие кнопки A приводит к выключению будильника.

Рассматриваемые часы с будильником являются системой со сложным поведением, так как в ответ на одни и те же входные события (нажатия кнопок) в зависимости от режима работы генерируются различные выходные воздействия. Поведение этих часов может быть описано с помощью приведенного в [2] конечного автомата, который содержит три состояния (рис. 3).

Построение конечного автомата управления часами с будильником проводилось при следующих параметрах алгоритма генетического программирования: размер поколения – 2000 особей; доля «элиты» – наиболее приспособленных особей, напрямую переходящих в следующее поколение – 10 %; число поколений до малой «мутации поколения» – 100 поколений; число поколений до большой «мутации поколения» – 150 поколений; размер автоматов в начальном поколении – четыре состояния.

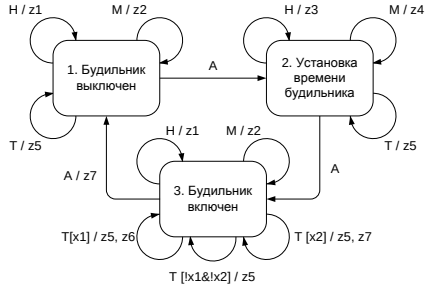

Рис. 3. Граф переходов автомата управления часами с будильником

Было проведено 1000 запусков алгоритма с указанными параметрами. Цель в каждом из них состояла в том, чтобы построить автомат, содержащий 14 переходов и соответствующий всем тестам (значение функции приспособленности, соответствующее такому автомату – 20.86). На каждом из запусков алгоритма генетического программирования был построен автомат, в котором из начального (нулевого) достижимы только три состояния из четырех. Если удалить недостижимое состояние, то этот граф переходов будет изоморфен графу пере-

ходов, построенному вручную. Для каждого из запусков запоминалось число вычислений функции приспособленности, которое равно числу просмотренных в процессе работы автоматов, в процессе построения автомата. Минимальное значение числа вычислений функции приспособленности составило 256063, максимальное – 9239523, среднее значение составляет 1443351.20 (стандартное отклонение – 1103401.82).

В работах [42 – 44] описанный метод был обобщен для случая не только дискретных, но и непрерывных выходных воздействий. В этом случае при определенном выборе функции приспособленности, расстановка выходных воздействий сводится к решению набора систем линейных уравнений. Эффективность предложенного метода была проиллюстрирована при построении автомата, обеспечивающего управление моделью беспилотного самолета при исполнении «мертвой петли». Моделирование беспилотного самолета проводилось с использование симулятора *FlightGear* (http://www.flightgear.org).

Для построения автомата были записаны три набора обучающих примеров (тестов), каждый из которых соответствовал выполнению «мертвой петли» под управлением пилота-человека. Каждый тест состоял из нескольких тысяч наборов входных и управляющих параметров, при записи опрос параметров производился 10 раз в секунду.

Время работы алгоритма составляло около 10 часов для одного набора тестов. При этом было обработано около двух тысяч поколений. Таким образом, создание и обработка одного поколения в среднем занимала около 20 с или 0,2 с на один автомат (размер поколения составлял 100 особей), что значительно меньше, чем в работе [32] , где это занимало около пять минут.

Было проведено около 50 запусков генетического алгоритма, в каждом из которых выбирался автомат с наибольшим значением функции приспособленности. Полеты моделей самолетов, управляемых выбранными автоматами, были просмотрены авторами. После этого автомат, используемый в полете, больше других похожем на «идеальную» «мертвую петлю», был назван лучшим. При этом отметим, что «идеальная» «мертвая петля» может отличаться от эталонной, которая выполняется вручную.

В процессе наблюдения за ходом выполнения «мертвом петли» под управлением лучшего автомата было установлено, что в зависимости от параметров среды и самолета при запуске автомата возможны три варианта выполнения «петли»:

- В большинстве случаев модель самолета, как и при управлении вручную, выполняет одну «мертвую петлю» и летит дальше.
- Иногда модель самолета может выполнить несколько «мертвых петель» с некоторым интервалом. Это происходит в случае, когда значения параметров модели самолета в конце выполнения «мертвой петли» схожи со значениями параметров в начале ее выполнения. Это приводит к тому, что если автомат после осуществления «петли» находится в том же состоянии, в котором был в начале, то поведение модели зацикливается. В ходе экспериментов авторы неоднократно наблюдали выполнение двух «мертвых петель» подряд. Выполнение большего числа «петель» не наблюдалось.
- Модель может вообще не выполнить «мертвую петлю», так как автомат не справляется с управлением, что, правда, бывает крайне редко.

Определение условий, при которых выполняется каждый из этих вариантов полета, требует дальнейших исследований.

Приведем ссылки на видеозаписи трех вариантов реализации «мертвой петли» под управлением лучшего автомата:

- Выполнение «мертвой петли», близкое к «идеальному» (http://www.youtube.com/watch?v=TzrLoJjjVTA);
- Выполнение «мертвой петли» с заваливанием на левый борт с последующим выравниванием (http://www.youtube.com/watch?v=C6WV7x2bqE8);
- Последовательное выполнение двух «мертвых петель» (http://www.youtube.com/watch?v=yFiG4yz67Ks).

3. Методы, использующие верификацию при вычислении функции приспособленности

3.1. Методы, описанные в работах других авторов

Работа [45] посвящена построению взаимодействующих конечных автоматов с использованием генетического программирования с функцией приспособленности, при вычислении которой используется верификация моделей [3]. Верификация моделей – это набор методов и алгоритмов, которые позволяют определять, соответствует ли программа некоторому множеству требований. При этом требования выражаются на языке темпоральной логики – записываются утверждения о поведении программы во времени.

В рассматриваемой работе решается задача построения системы взамодействующих конечных автоматов. Это означает, что конечные автоматы, входящие в систему, имеют общие глобальные переменные и каналы связи, по которым могут передаваться сообщения.

В эволюционном алгоритме каждый конечный автомат представляется в виде графов переходов, каждый из которых представляет собой набор вершин и дуг. Вершины соответствуют состояниям, а дуги – переходам. В качестве эволюционного алгоритма в рассматриваемой работе применяется так называемая (1+λ) эволюционная стратегия.

3.2. Методы, разработанные в НИУ ИТМО

В работах [46 – 49] предложен метод построения конечных автоматов на основе обучающих примеров и верификации. Для описания спецификации управляющего конечного автомата в этом методе применяется язык логики линейного времени *LTL* (*Linear Temporal Logic*). Алгоритм верификации основан на проверке пустоты языка, задаваемого пересечением рассматриваемого конечного автомата и автомата Бюхи, соответствующего отрицанию *LTL*-формулы, представляющей требование к автомату [3, 4]. Эта проверка осуществляется с помощью алгоритма двойного обхода в глубину.

Верификатор получает на вход модель автоматной программы и *LTL*-формулу [5]. После проверки модели верификатор либо сообщает, что формула выполняется, либо приводит контрпример – путь в модели, опровергающий утверждение [6].

Исходными данными, в дополнение к тем, которые использовались в методе, описанном в разд. 2.2, являются *LTL*-формулы, описывающие требования к управляющему конечному автомату.

При вычислении функции приспособленности учитывается как поведение автомата при обработке тестов, так и число выполняемых (верных) для автомата *LTL*-формул, составляющих спецификацию. При этом, чем больше число выполняемых формул и успешно пройденных тестов, тем больше значение функции приспособленности.

Для вычисления функции приспособленности конечный автомат, задаваемый рассматриваемой особью, запускается на всех тестах и проверяется на соответствие всем темпоральным формулам, составляющим спецификацию. Для учета указанных особенностей необходимо модифицировать введенную выше функцию приспособленности:

$$\mathrm{FF} = \mathrm{FF}_2 \cdot (1 + \frac{n_1}{n_2}) + \frac{1}{M} \cdot (M - \mathrm{cnt}).$$

Здесь n_2 – общее число темпоральных формул в спецификации, а n_1 – число формул, которые выполняются для рассматриваемого конечного автомата.

Операция мутации может выполняться двумя способами – традиционным и учитывающим результат верификации. При мутации, учитывающей результат верификации, происходит изменение одного из переходов, входящих в найденный контрпример. Это изменение может состоять либо в удалении перехода из этого пути, либо в изменении его конечного состояния, числа генерируемых выходных воздействий или события, инициирующего переход.

Экспериментальное исследование продложенного метода проводилось на задаче построения автомата управления дверьми лифта. Эта система содержит пять входных событий (*e11* – нажата кнопка «Открыть двери»; *e12* – нажата кнопка «Закрыть двери»; *e2* – открытие или

закрытие дверей успешно завершено; *e3* – препятствие мешает закрыть дверь; *e4* – дверь сломалась) и три выходные воздействия (*z1* – начать открытие дверей; *z2* – начать закрытие дверей; *z3* – позвонить в аварийную службу).

При построении управляющего автомата использовались девять тестов и 11 темпоральных свойств.

Целью экспериментального исследования было сравнение метода построения управляющих конечных автоматов на основе тестов с методом, использующим верификацию моделей. Для оценки трудоемкости сравниваемых методов было проведено по 1000 экспериментов, и для каждого эксперимента записывалось число вычислений функции приспособленности. Эксперименты показали, что при построении автоматов только на основе тестов, очень редко (всего в девяти случаях из 1000) результатом являлся автомат, который полностью удовлетворяет спецификации. Пример автомата, построенного только на основе тестов, приведен на рис. 5.

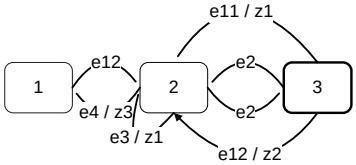

Рис. 5. Автомат управления дверьми лифта, построенный только на основе обучающих примеров

Это автомат обладает тем недостатком, что может отдать команду на закрытие дверей после того, как они сломаются или же начать открывать (закрывать) двери, когда они уже открыты (закрыты). Отметим, что некоторые из построенных в этом эксперименте автоматов обладали и другими недостатками.

При использовании верификации моделей построение автомата (рис. 6) занимало больше времени, но построенный автомат удовлетворял всем требованиям спецификации.

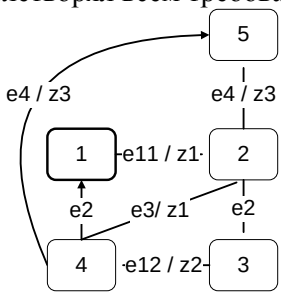

Рис. 6. Автомат управления дверьми лифта, построенный с использованием верификации

При построении конечного автомата управления дверьми лифта только на основе тестов, среднее значение вычислений функции приспособленности оказалось равным $7{,}479 \cdot 10^4$ (минимальное число вычислений – $2{,}184 \cdot 10^4$, максимальное – $2{,}999 \cdot 10^5$, среднеквадратичное отклонение – $2{,}54 \cdot 10^4$).

При использовании верификации моделей совместно с тестами, среднее значение числа вычислений функции приспособленности оказалось равным 7.246×10^5 (минимальное число вычислений – $7{,}054 \cdot 10^4$, максимальное – $5{,}492 \cdot 10^6$, среднеквадратичное отклонение – $7{,}729 \cdot 10^5$).

Таким образом, использование верификации хоть и замедляет процесс построения управляющего конечного автомата примерно в десять раз, но если принять во внимание то, что при построении только на основе тестов процент правильно построенных автоматов меньше 1%, то применение верификации в процессе работы генетического алгоритма оправдывает себя.

Заключение

В ходе исследований по применению генетических алгоритмов и генетического программирования для построения конечных автоматов были разработаны методы трех типов:

- методы, использующие моделирование для вычисления функции приспособленности;
- методы, использующие обучающие примеры (тесты) для вычисления функции приспособленности;
- методы, использующие верификацию при вычислении функции приспособленности.

Применение указанных методов повышает уровень автоматизации построения автоматных программ (с учетом сказанного выше, более 60% исходного кода программ могут быть построено автоматически), снижает влияние человеческого фактора на их качество и является одним из шагов к автоматическому построению программ.

Исследования в НИУ ИТМО проводились и проводятся в рамках Федеральной целевой программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2012 годы» и Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России на 2009–2013 годы».

Литература

1. *Шалыто А. А.* Автоматное программирование / Материалы конференции с международным участием «Технические и программные средства систем управления, контроля и измерения» (УКИ'10). ИПУ РАН. 2010, с. 156 – 167.
2. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирования. СПб: Питер, 2009.
3. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ. М.: МЦНМО, 2002.
4. *Gerth R., Peled D., Vardi M. Y., Wolper P.* Simple On-the-fly Automatic Verification of Linear Temporal Logic / Proc. of the 15th Workshop on Protocol Specification, Testing, and Verification, Warsaw, 1995, pp. 3 – 18.
5. Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода. Второй этап. СПбГУ ИТМО, 2007. http://is.ifmo.ru/verification/_2007_02_report-verification.pdf.
6. *Егоров К. В., Шалыто А. А.* Методика верификации автоматных программ // Информационно-управляющие системы. 2008, № 5, с. 15 – 21.
7. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. 2004. № 6. с. 12 – 17.
8. *Колыхматов И. И., Рыбак О. О., Шалыто А. А.* Моделирование устройства для продажи газированной воды на инструментальном средстве *UniMod*. Проектная документация. СПбГУ ИТМО. 2006. http://is.ifmo.ru/download/vending-machine-ru.pdf
9. Заочный тур всесибирской олимпиады 2005 по информатике. http://olimpic.nsu.ru/widesiberia/archive/wso6/2005/rus/1tour/problem/problem.html
10. *Паращенко Д. А., Царев Ф. Н., Шалыто А. А.* Технология моделирования одного класса мультиагентных систем на основе автоматного программирования на примере игры «Соревнование летающих тарелок». Проектная документация. СПбГУ ИТМО. 2006. http://is.ifmo.ru/unimod-projects/plates/
11. *Mitchell M.* An Introduction to Genetic Algorithms. MA: The MIT Press, 1996.
12. *Букатова И. Л.* Эволюционное моделирование и его приложения. М.: Наука, 1979.
13. *Fogel L., Owens A., Walsh M.* Artificial Intelligence through Simulated Evolution. NY: Wiley. 1966.
14. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System: Evolution as a Theme in Artificial Life /Proceedings of Second Conference on Artificial Life. MA: Addison-Wesley. 1992, pp.549 – 578. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html
15. *Рассел С., Норвиг П.* Искусственный интеллект. Современный подход. М.: Вильямс. 2006.
16. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.
17. *Курейчик В.М.* Генетические алгоритмы. Состояние, проблемы, перспективы // Известия РАН. Теория и системы управления. 1999. № 1, с. 144 – 160.

18. *Курейчик В.В., Курейчик В.М., Сороколетов П.В.* Анализ и обзор моделей эволюции // Известия РАН. Теория и системы управления. 2007. № 5, с. 114 – 126.
19. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volume III. CRC Press, 1999.
20. *Koza J. R.* Genetic programming: on the programming of computers by means of natural selection. MIT Press, 1992.
21. *Angeline P. J., Pollack J.* Evolutionary Module Acquisition // Proceedings of the Second Annual Conference on Evolutionary Programming. 1993. http://www.demo.cs.brandeis.edu/papers/ep93.pdf
22. *Spears W., Gordon D.* Evolving Finite-State Machine Strategies for Protecting Resources. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. Volume 1932/2009, pp. 5 – 28.
23. Технология генетического программирования для генерации автоматов управления системами со сложным поведением. Промежуточный отчет по этапу I «Выбор направлений исследований и базовых компонентов». http://is.ifmo.ru/genalg/_2007_01_report-genetic.pdf
24. Технология генетического программирования для генерации автоматов управления системами со сложным поведением. Промежуточный отчет по этапу II «Теоретические исследования поставленных перед НИР задач». http://is.ifmo.ru/genalg/_2007_02_report-genetic.pdf
25. Технология генетического программирования для генерации автоматов управления системами со сложным поведением. Промежуточный отчет по этапу III «Экспериментальные исследования поставленных перед НИР задач». http://is.ifmo.ru/genalg/_2007_03_report-genetic.pdf
26. Технология генетического программирования для генерации автоматов управления системами со сложным поведением. Промежуточный отчет по этапу IV «Обобщение и оценка результатов исследований». http://is.ifmo.ru/genalg/_2007_04_report-genetic.pdf
27. *Лобанов П. Г.* Использование генетических алгоритмов для генерации конечных автоматов. Диссертация на соискание ученой степени кандидата технических наук. СПбГУ ИТМО. 2008. http://is.ifmo.ru/disser/lobanov_disser.pdf
28. *Царев Ф. Н., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об «Умном муравье». / Тезисы научно-технической конференции «Научно-программное обеспечение в образовании и научных исследованиях». СПбГУ ПУ. 2008, с. 209 – 215.
29. *Царев Ф. Н., Шалыто А. А.* О построении автоматов с минимальным числом состояний для задачи об «Умном муравье» /Сборник докладов X международной конференции по мягким вычислениям и измерениям. СПбГЭТУ «ЛЭТИ». Т. 2, 2007, с. 88 – 91. http://is.ifmo.ru/download/ant_ga_min_number_of_state.pdf
30. *Данилов В. Р.* Метод представления автоматов деревьями решений для использования в генетическом программировании //Научно-технический вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование, с. 103 – 108.
31. *Данилов В. Р.* Технология генетического программирования для генерации автоматов управления системами со сложным поведением. СПбГУ ИТМО. 2007. Бакалаврская работа. http://is.ifmo.ru/papers/danilov_bachelor/
32. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Применение генетического программирования для генерации автоматов с большим числом входных переменных //Известия РАН. Теория и системы управления. 2010. № 2, с.100 – 117.
33. *Lucas S., Reynolds J.* Learning DFA: Evolution versus Evidence Driven State Merging // The 2003 Congress on Evolutionary Computation (CEC '03). Vol. 1, pp. 351 – 358.
34. *Lucas S., Reynolds J.* Learning Deterministic Finite Automata with a Smart State Labeling Algorithm // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2005. №7, pp. 1063 – 1074.
35. *Belz A., Eskikaya B.* A Genetic Algorithm for Finite State Automata Induction with Application to Phonotactics / Proceedings of the ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing, Saarbruecken, 1998, pp. 9 – 17.
36. *Spichakova M.* Genetic Inference of Finite State Machines. Masters thesis. Tallinn. 2007. http://s-ma-u-g.googlecode.com/files/thesis.pdf
37. *Lucas S.* Evolving Finite-State Transducers: Some Initial Explorations. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. Volume 2610/2003, pp. 241 – 57.
38. *Hamming R.* Error detecting and error correcting codes. Bell System Technical Journal 29 (2), pp. 147–160.

39. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академии Наук СССР 1963. № 4, с. 845 – 848.
40. *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.
41. *Царев Ф. Н.* Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования // Информационно-управляющие системы. 2010. № 5, с. 31 – 36.
42. *Александров А. В., Казаков С. В., Сергушичев А. А., Царев Ф. Н.* Генетическое программирование на основе обучающих примеров для построения конечных автоматов управления моделью беспилотного самолета /Сборник докладов международной конференции по мягким вычислениям и измерениям (SCM`2010). СПбГЭТУ «ЛЭТИ». Т. 1, с. 263 – 267.
43. *Alexandrov A., Sergushichev A., Kazakov S., Tsarev F.* Genetic Algorithm for Induction of Finite Automation with Continuous and Discrete Output Actions / Proceedings of the 2011 GECCO Conference Companion on Genetic and Evolutionary Computation. NY. : ACM. 2011, pp. 775 – 778.
44. *Александров А. В., Казаков С. В. Сергушичев А. А., Царев Ф. Н., Шалыто А. А.* Генерация конечных автоматов для управления моделью беспилотного самолета // Научно-технический вестник СПбГУ ИТМО. 2011. № 2, с. 3 – 11.
45. *Johnson C.* Genetic Programming with Fitness based on Model Checking /Lecture Notes in Computer Science. Springer. Berlin / Heidelberg. 2007. Volume 4445/2007, pp. 114 – 124.
46. *Егоров К. В., Царев Ф. Н.* Совместное применение генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением /Сборник трудов конференции «Информационные технологии и системы» (ИТИС`09). М.: Институт проблем передачи информации им. А. А. Харкевича РАН. 2009, с. 77 – 82.
47. *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 Санкт-Петербургского государственного университета информационных технологий, механики и оптики, № 5 (69), 2010, с. 81 – 86.
48. *Егоров К. В., Царев Ф. Н.* Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе верификации моделей и обучающих примеров / Материалы второй межвузовской научной конференции по проблемам информатики «Список-2011». СПб.: ВВМ. 2011, с. 343 – 350.
49. *Tsarev F., Egorov K.* Finite State Machine Induction using Genetic Programming Based on Testing and Model Checking / Proceedings of the 2011 GECCO Conference Companion on Genetic and Evolutionary Computation. NY. : ACM. 2011, pp. 759 – 762.