УДК 004.4’252, 004.023, 004.855.5

МУРАВЬИНЫЙ АЛГОРИТМ ДЛЯ ПОСТРОЕНИЯ АВТОМАТНЫХ ПРОГРАММ ПО СПЕЦИФИКАЦИИ

Д.С. Чивилихин
Университет ИТМО
Россия, 197101, Санкт-Петербург, пр. Кронверкский, 49
E-mail: chivdan@rain.ifmo.ru

В.И. Ульянцев
Университет ИТМО
Россия, 197101, Санкт-Петербург, пр. Кронверкский, 49
E-mail: ulyantsev@rain.ifmo.ru

А.А. Шалыто
Университет ИТМО
Россия, 197101, Санкт-Петербург, пр. Кронверкский, 49
E-mail: shalyto@mail.ifmo.ru

Ключевые слова: конечный автомат, муравьиный алгоритм, верификация программ, надежность

Аннотация: В некоторых прикладных областях предъявляются высокие требования к надежности программного обеспечения. Традиционный в области разработки программ процесс тестирования не может гарантировать корректность программы, поэтому прибегают к верификации. Верификация позволяет проверять некоторые свойства программы во всех возможных ее состояниях, однако сам процесс верификации сложен. В методе проверки моделей (model checking) вручную строится модель программы, а требования к ней записываются свойства на языке темпоральной логики. Выполнение или невыполнение этих требований к модели может быть относительно просто проверено. Основной проблемой этого метода является разрыв между программой и ее моделью. Парадигма автоматного программированию позволяет преодолеть эту проблему. В автоматном программировании логика работы программ описывается управляющими конечными автоматами, модели которых могут быть построены автоматически. В работе приводятся результаты исследований по применению муравьиных алгоритмов для решения задач построения управляющих конечных автоматов.

1. Введение

Проектирование программного обеспечения является сложной задачей, особенно когда область применения требует высокого уровня надежности программ. В таких

областях, как финансы, а также авиационная и космическая промышленность, цена ошибок очень высока, так как их возникновение может привести к большим потерям ресурсов или даже человеческих жизней. Программы в таких случаях не могут подвергаться одной лишь процедуре тестирования, поскольку она может только указать на наличие ошибок, но не может гарантировать их отсутствия.

Метод проверки моделей (model checking [7]) применяется для проверки некоторых свойств программного обеспечения во всех возможных его состояниях. При этом сначала вручную строится модель рассматриваемой программной системы, а требований к модели записываются на языке темпоральной логики. Темпоральные свойства проверяется для модели, но не для исходной программной системы, что в общем случае указывает на разрыв между программным обеспечением и его моделью. Парадигма автоматного программирования [1] позволяет преодолеть это ограничение. Программы, разработанные с помощью этой парадигмы могут быть автоматически преобразованы в используемые в методе проверки моделей структуры [11].Таким образом, в данном случае нет никакого разрыва между этими программами и их моделями.

В автоматном программировании предлагается разрабатывать программные системы в виде набора *автоматизированных объектов управления*. Автоматизированный объект управления состоит из объекта управления и системы управления, представленной в виде одного или нескольких взаимодействующих управляющих конечных автоматов.

Объект управления характеризуется набором команд, реализующих его методы. Автоматизированный объект управления может иметь несколько связанных с ним поставщиков событий, которые оповещают его о том, что происходит в системе. В частности, сам объект управления может являться поставщиком событий. После получения события от некоторого поставщика, автомат выполняет переход в новое состояние, посылая объекту управления последовательность выходных воздействий (команд).

Еще одно важное понятие в автоматном программировании – система со сложным поведением. Система с простым поведением всегда реагирует одинаково на одни и те же события. Система со сложным поведением, напротив, может реагировать поразному на одни и те же события в зависимости от истории взаимодействия. Подавляющее большинство реальных программных систем являются системами со сложным поведением. Парадигма автоматного программирования заключается в разработке систем со сложным поведением в виде автоматизированных объектов управления [1].

Ключевым преимуществом автоматного программирования является возможность создавать программы автоматически по спецификации, которая может содержать тестовые примеры и темпоральные свойства. Для этого, в частности, применяются алгоритмы поисковой оптимизации. Так, в [2] для этого применялись эволюционные алгоритмы.

В данной работе описывается метод построения управляющих конечных автоматов, разработанный в [5] и основанный на муравьином алгоритме [8] нового типа. Ранее новый алгоритм сравнивался с известными методами на задачах построения автоматов по тестовым примерам, а также на основе моделирования [5,6]. Здесь этот алгоритм применяется к задаче построения автоматов по спецификации, включающей темпоральные формулы. Экспериментально показывается, что новый алгоритм обладает более высокой производительностью по сравнению с ранее применявшими-

ся генетическими алгоритмами.

2. Построение автоматных программ по спецификации

Управляющим конечным автоматом называется семерка $(S, s_0, Z, \Sigma, \Delta, \delta, \lambda)$, где S – множество *состояний*, $s_0 \in S$ – *начальное состояние*, Z – множество булевых *входных переменных*, Σ – множество *входных событий*, Δ – *множество выходных воздействий*, $\delta\colon S \times \Sigma \times 2^Z \to S$ – *функция переходов*, а $\lambda\colon S \times \Sigma \times 2^Z \to \Delta^*$ – *функция выходов*.

Проще говоря, управляющий конечный автомат это такой автомат, каждый переход которого помечен событием, булевой формулой от входных переменных, называемой *охранным условием*, и последовательностью выходных воздействий. Пример управляющего автомата с двумя состояниями приведен на рис. 1.

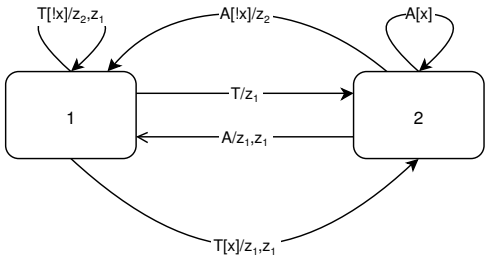


Рис. 1. Пример управляющего конечного автомата

2.1. Спецификация автоматных программ

Спецификация автоматной программы может включать в себя тестовые примеры, положительные и негативные сценарии, а также *LTL*-формулы. В дальнейшем для удобства мы будем называть управляющие конечные автоматы управляющими автоматами, конечными автоматами и просто автоматами.

Тестовый пример $T_i = \{In[i], Ans[i]\}$ – это пара из входной последовательности $In[i]$ и выходной последовательности $Ans[i]$. Элементами входной последовательности $In[i]$ являются пары $\langle e, f\rangle$, где $e \in \Sigma$ – входное событие, а f – булева формула от входных переменных. Выходные последовательности $Out[i]$ составлены из элементов множества выходных воздействий Δ.

Говорят, что автомат удовлетворяет тестовому примеру, если он переводит поданную ему на вход последовательность $In[i]$ в выходную последовательность $Ans[i]$. Иначе, автомат не удовлетворяет тестовому примеру.

Положительные сценарии схожи с тестовыми примерами, но несут больше информации. *Положительный сценарий* $Sc[i]$ это последовательность троек $\langle e, f, O\rangle$ называемых *элементами сценария*, где $e \in \Sigma$ – событие, f – булева формула от входных переменных и $O \in \Delta^*$ – последовательность выходных воздействий (возможно, пустая).

Говорят, что автомат удовлетворяет элементу сценария $\langle e, f, O\rangle$ в состоянии s, если в в этом состоянии у автомата существует переход, помеченный событием e,

последовательностью выходных воздействий O и охранным условием, равным f как булева формула. Автомат удовлетворяет положительному сценарию, если он удовлетворяет всем последовательно обработанным элементам сценария.

Негативным сценарием называется последовательность событий, которая не должна выполняться в искомом автомате. Автомат удовлетворяет негативному сценарию $N_i = \{e_1, e_2, \ldots, e_{n-1}, e_n\}$, если он принимает все префиксы сценария, кроме самого сценария. А именно, если после обработки последовательности событий $e_1, e_2, \ldots, e_{n-1}$ автомат находится в состоянии s, то в этом состоянии не должно существовать перехода по событию e_n.

Следуя [11], для представления темпоральных свойств автомата в работе используется логика линейного времени (Linear Temporal Logic, *LTL*). Язык *LTL* состоит из пропозициональных переменных *Prop*, логических операторов «**and**», «**or**», «**not**» и набора темпоральных операторов, таких как **G** (глобально в будущем), **X** (в следующий момент времени), **F** (когда-либо в будущем), **U** (до некоторого момента в будущем) и **R**. Для проверки того, удовлетворяет ли автомат *LTL*-формуле применяется верификатор автоматных программ, разработанный в [11]. Указанный верификатор принимает на вход автомат и *LTL*-формулу и либо выдает на выход контрпример, либо сообщение о том, что автомат удовлетворяет формуле.

Сформулируем задачу построения управляющего конечного автомата по спецификации. Требуется построить автомат с не более, чем N_{states} состояниями, удовлетворяющий набору тестовых примеров $\{T[i]\}$ (или набору сценариев $\{Sc_i\}$), набору негативных сценариев $\{N_i\}$, и множеству *LTL*-формул. Известно, что уже задача построения автомата только по сценариям является, как минимум, *NP*-трудной. Этим обуславливается необходимость применения эвристических алгоритмов при решении поставленной задачи.

2.2. Представление особей

Обычно методы построения автоматов по обучающим примерам напрямую не используют автоматы в качестве особей алгоритмов. Вместо этого применяются так называемые *каркасы автоматов*. Идея каркаса была впервые предложена в [10] при построении автоматов-распознавателей, а затем расширена в [11] для построения конечных преобразователей и управляющих конечных автоматов. Каркасом автомата в называется автомат, в котором переходы помечены не последовательностями выходных воздействий, а числом необходимых выходных воздействий. Пример каркаса автомата, изображенного на рис. 1, приведен на рис. 2.

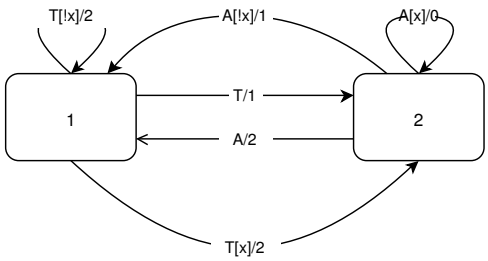


Рис. 2. Пример каркаса автомата

Для получения автомата по каркасу используется алгоритм расстановки выход-

ных воздействий, предложенный в [3]. Алгоритм получает на вход набор тестов (сценариев) и каркас автомата и позволяет назначить переходам каркаса последовательности выходных воздействий так, чтобы полученный автомат максимально точно описывал тесты (сценарии). Алгоритм поочередно обрабатывает каждый тестовый пример (сценарий) и строит для каждого перехода таблицу частот выходных последовательностей, которые на нем встречаются. Далее для каждого перехода выбирается наиболее часто встречающаяся последовательность выходных воздействий. Этот подход позволяет алгоритму поисковой оптимизации подбирать для каждого перехода автомата не саму последовательность выходных воздействий, а лишь ее длину, тем самым существенно сокращая размер пространства поиска.

Несмотря на то, что в описываемом алгоритме используются каркасы автоматов вместо самих автоматов, мы будем называть каркасы автоматами. Причиной этого является, во-первых, то, что алгоритм может строить не только каркасы, но и сами автоматы, а во-вторых то, что каркасы по сути тоже являются автоматами с измененным множеством выходных воздействий.

2.3. Функция приспособленности

Для построения управляющих автоматов используется функция приспособленности (ФП), предложенная в [11]. Эта функция учитывает соответствие автомата заданному набору тестов или положительных сценариев, множеству негативных сценариев и набору *LTL*-формул.

Первый компонент функции приспособленности F_{tests} оценивает насколько хорошо автомат соответствует заданному набору тестов или положительных сценариев. Каждый тестовый пример обрабатывается следующим образом. Входная последовательность i-го теста $In[i]$ подается на вход автомату, полученная выходная последовательность $Out[i]$ сравнивается с эталонной последовательностью $Ans[i]$ из тестового примера. Общее выражение для F_{tests} имеет вид:

$$F_{\text{tests}} = \frac{1}{|T|}\sum_{i=1}^{|T|}\left(1 - \frac{\text{ED}\left(Out[i], Ans[i]\right)}{\max\left(\text{len}\left(Out[i]\right), \text{len}\left(Ans[i]\right)\right)}\right),$$

где $|T|$ – число тестовых примеров, $\text{len}(s)$ – длина последовательности s, а $\text{ED}\left(s_1, s_2\right)$ – редакционное расстояние [9] между последовательностями s_1 и s_2. В случае использование сценариев вместо тестов, F_{tests} вычисляется аналогичным образом.

Второй компонент функции приспособленности F_{negSc} оценивает соответствие автомата негативным сценариям $\{N_i\}_i$ и определяется как число негативных сценариев, которым автомат не удовлетворяет, деленное на общее число негативных сценариев.

Третий компонент F_{LTL} оценивает соответствие автомата темпоральным свойствам. Верификатор, разработанный в [11], позволяет выделить переходы автомата, которые точно не входят в контрпример. Эти переходы называются *проверенными*. Вклад i-й *LTL*-формулы рассчитывается как число проверенных переходов t^i_{checked}, деленное на общее число переходов $t^i_{\text{reachable}}$, достижимых из начального состояния. F_{LTL} вычисляется как среднее значение этой величины по всем *LTL*-формулам:

$$F_{\text{LTL}} = \frac{1}{m}\sum_{i=1}^{m}\frac{t^i_{\text{checked}}}{t_{\text{reachable}}},$$

где m – число LTL-формул.

Итоговое выражение для функции приспособленности учитывает число переходов автомата и имеет вид:

$$F = f_{\text{tests}} - F_{\text{negSc}} + F_{\text{LTL}} + \frac{M - n_{\text{transitions}}}{100 \cdot M},$$

где $n_{\text{transitions}}$ – число всех переходов автомата, а M – число, гарантированно превосходящее $n_{\text{transitions}}$. В экспериментах использовалось значение $M = 100$.

3. Муравьиные алгоритмы

Муравьиные алгоритмы [8] – семейство алгоритмов поисковой оптимизации, основанных на поведении колоний муравьев в процессе поиска пищи. Эти алгоритмы относятся к методам роевого интеллекта [4], которые применяют принципы самоорганизации коллективов живых существ для решения комбинаторных задач. Коллектив обычно состоит из относительно простых особей-агентов. Процессы самоорганизации приводят к тому, что интеллект коллектива существенно превосходит интеллект отдельных агентов.

Для решения комбинаторной задачи с помощью муравьиного алгоритма обычно выполняют ее сведение к задаче поиска кратчайшего пути в некотором графе, называемом *графом конструирования*. Вершины этого графа, который обычно является полным, обозначают компоненты решения комбинаторной задачи.

Каждому ребру (u, v) графа (u и v – вершины графа) ставится в соответствие число τ_{uv}, называемое *значением феромона*. Также на ребре может быть задано число η_{uv}, называемое *эвристической информацией*. Различие между этими двумя величинами состоит в том, что эвристическая информация задается изначально, исходя из условий задачи, и не меняется во время выполнения алгоритма, в то время как значения феромона изменяются агентами-муравьями в процессе построения решений. Общая схема любого муравьиного алгоритма состоит из трех последовательных этапов, которые повторяются, пока не будет найдено оптимальное решение или не выполнится какой-либо критерий останова. Примером такого критерия останова является исчерпание выделенных алгоритму вычислительных ресурсов.

На первом этапе, называемом `ConstructAntSolutions`, колония муравьев строит решения. В начале этого этапа каждый муравей помещается в некоторую вершину графа. Размещение муравьев по начальным вершинам обычно зависит от конкретной задачи. Каждый муравей добавляет компоненты в свое текущее решение, переходя по вершинам графа до тех пор, пока он не построит полное решение. Выбор следующей вершины осуществляется с помощью некоторого вероятностного алгоритма.

На этапе `UpdatePheromones` выполняется обновление значений феромона на ребрах графа. Значение феромона на ребре графа может увеличиться, если ребро вошло в путь некоторого муравья, либо уменьшиться вследствие *испарения феромона* – пропорционального уменьшения значений феромона на всех ребрах графа.

На третьем, необязательном этапе `DaemonActions` выполняются операции, которые не могут быть выполнены отдельными муравьями. Например, могут производиться локальные оптимизации найденных на данной итерации решений.

4. Алгоритм построения автоматов

4.1. Операторы мутации

Под мутацией конечного автомата понимается небольшое изменение его структуры – таблиц, задающих функции переходов и выходов. В данной работе используются следующие три оператора мутации конечных автоматов.

- **Изменение состояния, в которое ведет переход**. Для случайно выбранного перехода в автомате состояние s, в которое он ведет, заменяется другим состоянием, выбранным случайным образом из множества $S \setminus \{s\}$.

- **Изменение выходного воздействия на переходе**. Воздействие a на случайно выбранном переходе заменяется на другое воздействие, выбранное случайным образом из множества $\Delta \setminus \{a\}$. Этот оператор не применяется в случае построения автоматов по сценариям.

- **Добавление или удаление переходов**. Наличие некоторых переходов в состоянии автомата может сделать его противоречащими некоторым LTL-формулам или негативным сценариям. Поэтому существует необходимость удалять и, соответственно, добавлять переходы. Оператор мутации сканирует состояния автомата и с вероятностью p_{mutTran} изменяет набор переходов в выбранном состоянии. Во всех экспериментах использовалось значение $p_{\text{mutTran}} = 0.05$. Если было принято решение изменить переходы, то случайным образом решается, добавить или удалить переход. При этом переход добавляется лишь в том случае, когда в этом состоянии нет перехода по какому-либо событию. Если это так, то в состояние добавляется новый переход, пометки которого выбираются случайным образом из соответствующих множеств состояний и действий. В случае удаления перехода, из текущего состояния удаляется случайно выбранный переход.

4.2. Представление пространства поиска

Основой разработанного метода построения конечных автоматов является представление пространства поиска, т.е. множества всех автоматов с заданными параметрами, в виде ориентированного графа G. Каждая вершина этого графа, который мы будем называть *графом мутаций*, ассоциирована с конечным автоматом, а ребра соответствуют мутациям автоматов. Небольшой пример графа мутаций приведен на рис. 3. Значение эвристической информации на каждом ребре (u, v) графа вычисляется по формуле:

$$\eta_{uv} = \max\left(\eta_{\min}, F(v) - F(u)\right),$$

где $\eta_{\min}$ – небольшая положительная константа, например, 10^{-3}.

4.3. Построение решений муравьями

Процедуру построения решений муравьями можно разделить на два этапа. На первом этапе выбираются вершины графа, из которых муравьи начнут построение

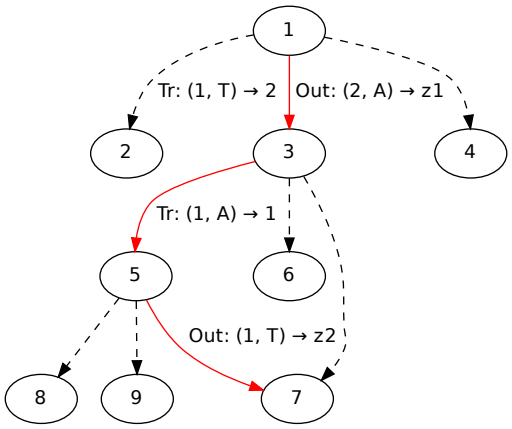


Рис. 3. Пример графа мутаций. Мутации на ребрах подписаны в следующей нотации – «Tr : (состояние, событие) $\to$ новое состояние» (изменение состояния, в которое ведет переход) и «Out : (состояние, событие) $\to$ новое выходное воздействие» (изменение выходного воздействия на переходе)

путей. В качестве единственной стартовой вершины для всех муравьев выбирается вершина, ассоциированная с лучшим найденным решением.

На втором этапе совершается одна итерация муравьиной колонии, в ходе которой каждый муравей перемещается по графу, начиная с соответствующей стартовой вершины. Пусть муравей находится в вершине u, ассоциированной с автоматом A. Если у вершины u существуют инцидентные ей ребра, то муравей выполняет одно из следующих действий.

1. **Построение новых решений**. С вероятностью p_{new} муравей пытается создать новые ребра и вершины графа G путем выполнения фиксированного числа N_{mut} мутаций автомата A. После выполнения муравьем всех N_{mut} мутаций он выбирает лучшую из построенных вершин и переходит в нее. Процесс обработки одной мутации автомата A таков:

   - выполнить мутацию автомата A, получить автомат A_{mut};
   - найти в графе G вершину t, ассоциированную с автоматом A_{mut}. Если такой вершины не существует, создать ее и добавить в граф;
   - добавить в граф ребро (u, t).

   Отметим, что переход осуществляется даже в том случае, когда значение функции приспособленности у лучшего из построенных с помощью мутаций автоматов меньше значения функции приспособленности автомата A. Это свойство алгоритма позволяет ему эффективно выходить из локальных оптимумов.

2. **Вероятностный выбор**. С вероятностью $1 - p_{\text{new}}$ муравей выбирает следующую вершину из множества N_u ребер, инцидентных вершине u. Вершина v выбирается с вероятностью, определяемой классической в муравьиных алго-

ритмах формулой [8]:

$$p_{uv} = \frac{\tau_{uv}^{\alpha} \cdot \eta_{uv}^{\beta}}{\sum_{w \in N_u} \tau_{uw}^{\alpha} \cdot \eta_{uw}^{\beta}},$$

где $\alpha, \beta \in [0, 1]$.

Если у вершины u нет инцидентных ей ребер, то муравей всегда выполняет построение новых решений. Все муравьи в колонии запускаются «параллельно» – каждый из них делает по одному шагу до тех пор, пока все муравьи не остановятся. Каждый муравей может выполнить максимум n_{stag} шагов, на каждом из которых происходит построение новых решений либо вероятностный выбор, без увеличения своего значения ФП. После этого муравей будет остановлен. Аналогично, колония муравьев может выполнить максимум N_{stag} итераций без увеличения максимального значения ФП. После этого алгоритм перезапускается. Сплошные ребра графа мутаций на рис. 3 обозначают путь муравья, а штрихованные ребра – все остальные мутации, которые были выполнены муравьем.

4.4. Обновление значений феромона

Значение феромона, которое муравей откладывает на ребрах своего пути, равно максимальному значению ФП всех автоматов, посещенных этим муравьем. Для каждого ребра (u, v) графа G хранится число τ_{uv}^{best} – наибольшее из значений феромона, когда-либо отложенных на этом ребре. Последовательно рассматриваются все пути муравьев на текущей итерации алгоритма. Для каждого пути муравья выделяется отрезок от начала пути до вершины, содержащей автомат с наибольшим на пути значением ФП. Для всех ребер из этого отрезка обновляются значения τ_{uv}^{best}, а затем значения феромона на всех ребрах графа G обновляются по формуле:

$$\tau_{uv} = max(\tau_{\min}, (1 - \rho)\tau_{uv} + \tau_{uv}^{\text{best}}),$$

где $\rho \in [0, 1]$ – скорость испарения феромона, а $\tau_{\min} = 10^{-3}$ – минимальное разрешенное значение феромона. Введение нижней границы $\tau_{\min}$ исключает чрезмерное испарение феромона с ребер графа.

5. Эксперименты

5.1. Настройка параметров сравниваемых алгоритмов

Для выполнения справедливого сравнения генетического и муравьиного алгоритмов была проведена процедура автоматического выбора значений их параметров, называемая еще настройкой параметров. Алгоритм настройки парамеров получает на вход алгоритм, требующий настройки, t_{tune} – время, отведенное для настройки, описание параметров алгоритма и экземпляр задачи, на которой будет проводиться настройка.

Обоим алгоритмам было отведено не более $t_{\text{tune}} = 12$ часов на настройку. Описание параметра алгоритма включает его имя, тип (например, `int` или `float`) и допустимый диапазон его значений. Естественно, что система настройки выполняет запуски настраиваемого алгоритма для оценки его производительности при заданном

наборе параметров. Каждый запуск алгоритма ограничивался $n_{\max}^{\text{run}}$ вычислениями ФП.

Сначала производится десять запусков алгоритма со случайными значениями параметров. Это делается для того, чтобы получить примерную оценку времени работы алгоритма t_{run}. Используя это приближение, можно рассчитать примерное число запусков, которые могут быть выполнены во время процесса настройки как $n_{\text{runs}} = \frac{t_{\text{tune}}}{t_{\text{run}}}$. Предположим, что каждый из n_{params} параметров будет иметь ровно n_{levels} значений. Для оценки качества конкретного набора параметров производится n_{repeats} запусков алгоритма и вычисляется средняя производительность алгоритма по всем запускам. Число значений каждого параметра n_{levels} может быть рассчитано по формуле:

$$n_{\text{levels}} = \exp\left(\frac{\ln \frac{n_{\text{runs}}}{n_{\text{repeats}}}}{n_{\text{params}}}\right).$$

Зная минимальное и максимальное значение каждого параметра, а также число его значений, можно провести настройку значений параметров. Наборы параметров сравниваются в соответствии с долей успешных запусков алгоритма, которую они обеспечивают. Запуск считается успешным, если в его результате было найдено оптимальное решение задачи.

5.2. Построение автомата управления дверьми лифта

В рассматриваемой задаче объект управления представляет собой модель дверей лифта [11]. Система характеризуется пятью входными событиями: e_{11} (нажата кнопка «Открыть двери»), e_{12} (нажата кнопка «Закрыть двери»), e_2 (двери успешно открыты), e_3 (препятствие помешало дверям закрыться), e_4 (двери заклинило). Существует три возможных выходных воздействия: z_1 (начать открывать двери), z_2 (начать закрывать двери), z_3 (послать аварийный сигнал). Спецификация, являющаяся входными данные для генетического и муравьиного алгоритмов, состоит из девяти тестовых примеров (девяти положительных сценариев), 30 негативных сценариев и 11-ти *LTL*-формул. Поиск осуществлялся среди автоматов с не более, чем $N_{\text{states}} = 6$ состояниями. Искомый автомат с пятью достижимыми состояниями изображен на рис. 4.

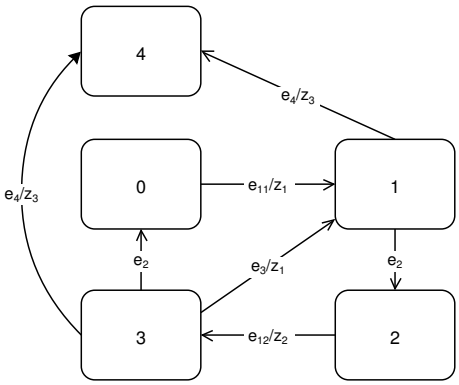


Рис. 4. Автомат управления дверьми лифта

Для осуществления сравнения с результатами, полученными в [11], было проведено два эксперимента. В первом эксперименте в качестве входных данных исполь-

зовались только тестовые примеры и *LTL*-формулы. Во втором эксперименте были также добавлены негативные сценарии работы, а вместо тестовых примеров использовались положительные сценарии работы.

Для настройки параметров алгоритмов был выбран второй вариант эксперимента. Каждый запуск каждого алгоритма в процессе настройки был ограничен $n_{\text{run}}^{\max} = 10000$ вычислениями ФП.

В каждом эксперименте было проведено по 1000 запусков каждого алгоритма, каждый запуск продолжался до достижения лучшим решением значения ФП, равного 2.0075. Для сравнения алгоритмов измерялось среднее, медианное, минимальное, максимальное число вычислений функции приспособленности, а также стандартное отклонение. Статистическая значимость полученных результатов была подтверждена с помощью t-теста, который дал p-value меньше, чем 2.2×10^{-16} для обоих экспериментов. Это указывает на то, что вероятность совпадения средних значений числа вычислений ФП для генетического и муравьиного алгоритмов мала. Результаты экспериментов приведены в таблице 1, гистограммы полученных распределений числа вычислений ФП изображены на рис. 5a для первого эксперимента и на рис. 5b для второго эксперимента.

Полученные результаты позволяют утверждать, что муравьиный алгоритм решает рассматриваемую задачу в два-три раза быстрее генетического алгоритма. Более того, муравьиный алгоритм не только решает задачу в среднем быстрее, но и с существенно меньшим стандартным отклонением. При этом использование сценариев вместо тестовых примеров в несколько раз увеличивает производительность обоих алгоритмов.

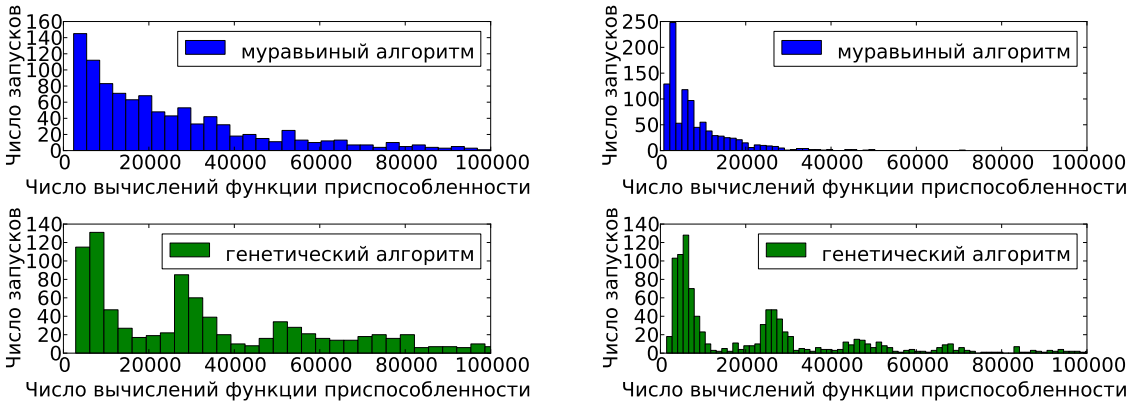
Таблица 1. Построение автомата управления дверьми лифта: число вычислений ФП для достижения оптимального решения

	Тесты + *LTL*-формулы		**Сценарии + *LTL*-формулы**	
Алгоритм	**Генетический**	**Муравьиный**	**Генетический**	**Муравьиный**
Среднее	52405	25976	25336	8370
Стандартное отклонение	61440	23389	28503	7991
Медиана	31507	18680	11915	6026
Минимум	2875	2392	1461	800
Максимум	498365	154802	195319	71480

6. Заключение

В статье был описан метод построения управляющих конечных автоматов на основе муравьиного алгоритма. Был предложен способ применения этого метода к задаче построения автоматов по спецификации, включающей обучающие примеры и темпоральные формулы. Проведенные экспериментальные исследования позволяют сделать вывод о том, что новый алгоритм в два-три раза превосходит по производительности ранее применявшийся метод, основанный на генетическом алгоритме.

Работа выполнена при государственной финансовой поддержке ведущих университетов Российской Федерации (субсидия 074-U01) и при поддержке РФФИ в рамках



(a) Тесты и *LTL*-формулы (b) Сценарии и *LTL*-формулы

Рис. 5. Распределение запусков алгоритмов по числу вычислений ФП

научного проекта 14-07-31337 мол_а.

Список литературы

1. Поликарпова Н.И., Шалыто А.А. Автоматное программирование. СПб.: Питер, 2009. 176 с.
2. Царев Ф.Н., Шалыто А.А. Построение конечных автоматов на основе генетических алгоритмов и генетического программирования // Труды Третьей российской конференции с международным участием «Технические и программные средства систем управления, контроля и измерения» УКИ ’12. М.: ИПУ РАН, 2012. С. 2017-2031.
3. Царев Ф.Н. Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования // Информационно-управляющие системы. 2010. № 5. С. 31-36.
4. Bonabeau E., Dorigo M., Theraulaz G. Swarm Intelligence: From Natural to Artificial Systems. New York, NY: Oxford University Press, Inc., 1999. 307 p.
5. Chivilikhin D., Ulyantsev V. MuACOsm: a new mutation-based ant colony optimization algorithm for learning finite-state machines // Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference GECCO ’13. New York, NY, USA: ACM, 2013. P. 511-518.
6. Chivilikhin D., Ulyantsev V., Tsarev F. Test-based extended finite-state machines induction with evolutionary algorithms and ant colony optimization // Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion GECCO ’12. New York, NY, USA: ACM, 2012. P. 603-606.
7. Clarke E.M., Grumberg O., Peled D.A. Model checking. Cambridge, MA: MIT press, 1999. 330 p.
8. Dorigo M., Stützle T. Ant Colony Optimization. Cambridge, MA: MIT Press, 2004. 319 p.
9. Левенштейн В.И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады Академии Наук СССР. 1965. № 4. С. 845-848.
10. Lucas S.M., Reynolds T.J. Learning deterministic finite automata with a smart state labelling evolutionary algorithm // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2005. Vol. 27. P. 1063-1074.
11. Tsarev F, Egorov K. Finite State Machine Induction Using Genetic Algorithm Based on Testing and Model Checking // Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation GECCO ’11. New York, NY, USA: ACM, 2011. P. 759-762.