Автоматика и телемеханика, № 3, 2016

Логическое управление

© 2016 г. Д.С. ЧИВИЛИХИН (chivdan@rain.ifmo.ru),
В.И. УЛЬЯНЦЕВ (ulyantsev@rain.ifmo.ru),
А.А. ШАЛЫТО, д-р техн. наук (shalyto@mail.ifmo.ru)
(Университет ИТМО, Санкт-Петербург)

МОДИФИЦИРОВАННЫЙ МУРАВЬИНЫЙ АЛГОРИТМ ДЛЯ ПОСТРОЕНИЯ КОНЕЧНЫХ АВТОМАТОВ ПО СЦЕНАРИЯМ РАБОТЫ И ТЕМПОРАЛЬНЫМ ФОРМУЛАМ[1]

Решается задача построения управляющих конечных автоматов по сценариям работы и темпоральным формулам. Предлагается новый алгоритм *pstMuACO*, совмещающий процедуру прореживания сценариев, точный алгоритм построения автоматов по сценариям работы *efsmSAT* на основе сведения к задаче выполнимости булевой формулы, и параллельный муравьиный алгоритм *pMuACO*. Эксперименты показали, что построение нескольких начальных решений для муравьиного алгоритма по сокращенным наборам сценариев существенно уменьшает суммарное время, необходимое для нахождения оптимального решения. Предложенный алгоритм может быть применен для автоматизированного построения надежных систем управления.

1. Введение

В определенных случаях к системам логического управления предъявляются повышенные требования надежности. Это относится, например, к системам, применяющимся в авиации, космонавтике и энергетике. В этих областях приложений ошибки в управляющей программе могут быть неприемлемыми. Высокого уровня надежности управляющих программ можно добиться с помощью верификации. Одним из подходов к верификации программ является метод проверки моделей (*model checking*) [1], в котором сначала строится модель проверяемой программы. Заметим, что в общем случае модель не эквивалентна исходной программе, так как строится либо вручную, либо эвристически. Свойства, которыми должна обладать модель проверяемой программы, записывают на языке темпоральной логики. Для проверки, удовлетворяет ли модель проверяемой программы этим свойствам, используются специальные программы – верификаторы, например *NuSMV* (`http://nusmv.fbk.eu/`) или *SPIN* (`http://spinroot.com`).

Одним из подходов к созданию систем логического управления является программирование с явным выделением состояний, или *автоматное программирование* [2, 3]. Суть этого подхода в том, что логика работы программ

[1] Работа выполнена при государственной финансовой поддержке ведущих университетов Российской Федерации (субсидия 074-U01), а также при поддержке Российского фонда фундаментальных исследований в рамках научного проекта № 14-01-0055114 а.

представляется в виде одного или нескольких взаимодействующих управляющих конечных автоматов. Одним из преимуществ автоматного программирования перед другими подходами является более высокий уровень автоматизации верификации автоматных программ при помощи метода проверки моделей [4]. Данное преимущество обусловлено тем, что точная модель автоматной программы может быть построена автоматически.

В некоторых случаях автоматные программы удается построить вручную, однако этот процесс является весьма трудоемким, а построенные человеком программы зачастую неоптимальны. Это неудивительно, так как в ряде случаев задачи построения конечных автоматов являются NP-трудными [5, 6]. Поэтому развиваются методы их автоматизированного построения, использующие метаэвристические алгоритмы оптимизации [7–9] (например, генетические [10] и муравьиные [11]), а также методы, основанные на сведении этих задач к другим NP-трудным задачам (задаче выполнимости булевой формулы SAT [12, 13] и задаче удовлетворения ограничениям CSP [14]).

Приведем пример, когда автоматную программу можно построить автоматически. Пусть заданы некоторые примеры поведения искомой автоматной программы, а также ее более общие свойства, записанные на языке темпоральной логики. Подход, предложенный в [10], позволяет с помощью генетического алгоритма автоматически построить управляющий конечный автомат, удовлетворяющий указанным примерам поведения и темпоральным свойствам.

Проблемой во всех методах автоматизированного построения конечных автоматов является то, что для построения даже небольших автоматов (до десяти состояний) может потребоваться до нескольких часов работы персонального компьютера. В [15] предложен параллельный алгоритм, основанный на муравьином алгоритме [11, 16]. Этот алгоритм позволяет за счет использования параллельных вычислений в несколько раз сократить время, необходимое для построения автоматов.

Данная статья является продолжением работы [15]. Решается задача построения управляющих конечных автоматов по сценариям работы и темпоральным формулам. Предлагается новый алгоритм *pstMuACO*, совмещающий параллельный муравьиный алгоритм *pMuACO* [15], точный алгоритм построения управляющих конечных автоматов по сценариям работы *efsmSAT* [13], основанный на решении задачи SAT, и процедуру прореживания сценариев. Как следует из [6], решаемая задача является как минимум NP-трудной, что обосновывает выбор класса алгоритмов ее решения. Проведенные эксперименты показали, что новый алгоритм позволяет строить автоматы существенно быстрее, чем известные алгоритмы.

2. Формулировка задачи построения управляющих конечных автоматов

Управляющим конечным автоматом [2] будем называть семерку $\langle X, E, Y, Z, y_0, \phi, \delta \rangle$, где X – множество булевых входных переменных, E – множество входных событий, Y – множество состояний, $y_0 \in Y$ – начальное состояние, Z – множество выходных воздействий, $\phi\colon Y \times E \times 2^X \to Y$ – функция переходов, а $\delta\colon Y \times E \times 2^X \to Z^*$ – функция выходов. Таким обра-

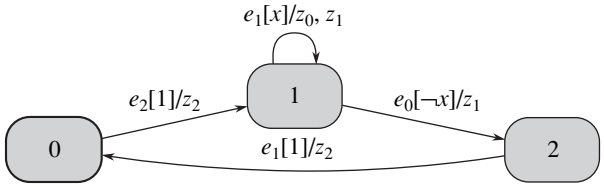

Рис. 1. Пример управляющего конечного автомата.

зом, в данной статье рассматриваются автоматы, каждый переход которых помечен входным событием, булевой формулой от входных переменных, и последовательностью выходных воздействий. Семантика работы автоматов такова. Пусть автомат находится в состоянии y. При поступлении очередного входного события $e \in E$ проверяется, существует ли переход из состояния y, помеченный событием e. Если такой переход существует и булева формула, которой он помечен, выполнима при текущих значениях входных переменных, то выполняется переход в новое состояние y', а на выход передается последовательность выходных воздействий, записанная на переходе.

Пример управляющего конечного автомата из трех состояний приведен на рис. 1. Каждый переход помечен входным событием из $E = \{e_0, e_1, e_2\}$, булевой формулой от единственной входной переменной x и последовательностью выходных воздействий из $Z = \{z_0, z_1, z_2\}$. Начальное состояние отмечено жирной рамкой.

Одним из типов исходных данных в задаче построения автоматов по спецификации являются примеры поведения, которое пользователь хочет наблюдать у искомой программы. В качестве таких примеров поведения могут выступать тестовые примеры, сценарии работы, негативные сценарии [17]. Второй тип исходных данных – темпоральные формулы, которым должна удовлетворять искомая программа. В данной статье в качестве примеров поведения рассматриваются сценарии работы, а темпоральные формулы задаются на языке логики линейного времени (Linear Time Logic, LTL).

Сценарий работы s_i – это последовательность троек $\left\{\langle e_i^j, \varphi_i^j, O_i^j\rangle\right\}_{j=0}^{l_i-1}$, называемых *элементами сценария*, где l_i – число элементов сценария s_i, $e_i^j \in E$ – входное событие, $\varphi_i^j \colon 2^X \to \{0,1\}$ – булева формула от входных переменных и $O_i^j \in Z^*$ – последовательность выходных воздействий. Считается, что автомат удовлетворяет сценарию работы $\langle e_i^j, \varphi_i^j, O_i^j\rangle$ в состоянии y, если в этом состоянии существует переход, помеченный событием e_i^j, последовательностью выходных воздействий O_i^j и формулой, равной φ_i^j как булева формула.

Опишем процедуру обработки сценария работы автоматом. Элементы сценария обрабатываются по очереди. При обработке элемента сценария проверяется, существует ли в автомате удовлетворяющий ему переход из текущего состояния. Если такой переход существует, то автомат переходит в новое состояние, на выход передается последовательность выходных воздействий, записанная на переходе. Таким образом, обработка сценария порождает путь

в автомате, представляющий собой последовательность посещенных состояний.

Автомат удовлетворяет сценарию работы, если он удовлетворяет всем элементам сценария в соответствующих состояниях этого пути. Например, автомат, изображенный на рис. 1, удовлетворяет сценарию $\langle e_2, 1, (z_2)\rangle\langle e_1, x, (z_0, z_1)\rangle\langle e_0, \neg x, (z_1)\rangle$, но не удовлетворяет сценарию $\langle e_2, 1, (z_2)\rangle\langle e_2, \neg x, (z_1)\rangle$.

LTL-формула включает в себя специфичные для конкретной решаемой задачи пропозициональные переменные, логические операторы $(\wedge, \vee, \neg)$, а также темпоральные операторы, такие как $\boldsymbol{G}$*lobally* (в любой момент времени), *ne*$\boldsymbol{X}t$ (в следующий момент времени), $\boldsymbol{F}$*uture* (когда-либо в будущем), $\boldsymbol{U}$*ntil* и $\boldsymbol{R}$*elease*. Формулы, рассматриваемые в данной статье, содержат следующие пропозициональные переменные:

— $\forall e \in E$: wasEvent(e) – совершен переход, помеченный входным событием e;

— $\forall z \in Z$: wasAction(z) – совершен переход, помеченный выходным воздействием z.

Автомат, изображенный на рис. 1, удовлетворяет *LTL*-формуле $\boldsymbol{G}(\neg\text{wasEvent}(e_0) \vee \boldsymbol{F}(\text{wasEvent}(e_2) \wedge \text{wasAction}(z_2)))$, которая утверждает, что если был выполнен переход, помеченный событием e_0, то в будущем будет выполнен переход, помеченный событием e_2 и выходным воздействием z_2. Формула $\boldsymbol{G}\,(\neg\text{wasEvent}(e_2) \vee \boldsymbol{X}\,(\text{wasEvent}(e_1)))$ неверна для автомата, так как после выполнения перехода по событию e_2 может быть выполнен переход не только по событию e_1, но и по событию e_0.

В данной статье решается задача построения управляющего конечного автомата с заданным числом состояний, удовлетворяющего заданному набору сценариев работы S и набору *LTL*-формул. Рассматриваемые алгоритмы построения автоматов осуществляют направленный перебор решений-кандидатов. Для оценки, насколько хорошо решение-кандидат удовлетворяет заданному набору сценариев и *LTL*-формул, используется соответствующая *функция приспособленности*.

3. Функция приспособленности

Для оценки соответствия автоматов заданному набору сценариев S и *LTL*-формулам используется функция приспособленности (ФП), предложенная в [17]. Эта ФП основана на редакционном расстоянии [18], верификаторе автоматных программ [17] и имеет вид

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

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

Первый компонент ФП F_{tests} оценивает соответствие автомата заданному набору сценариев работы S. При вычислении значения ФП автома-

та каждый сценарий s_i обрабатывается следующим образом. Автомату подаются на вход пары входных событий и булевых формул сценария s_i: $\langle e_i^0, \varphi_i^0\rangle, \langle e_i^1, \varphi_i^1\rangle, \ldots, \langle e_i^{l_i-1}, \varphi_i^{l_i-1}\rangle$. После обработки каждой такой пары автомат выдает некоторую последовательность выходных воздействий. В результате для i-го сценария получаем последовательность последовательностей $A_i = A_i^0, \ldots, A_i^{l_i-1}$. По полученной выходной последовательности A_i и эталонной последовательности $O_i = O_i^0, \ldots, O_i^{l_i-1}$, записанной в сценарии, вычисляется редакционное расстояние Левенштейна $\mathrm{ED}(O_i, A_i)$, изначальная версия которого для двоичных строк была предложена в [18]. Выражение для F_{tests} имеет вид:

$$F_{tests} = \frac{1}{|S|}\sum_{i=0}^{|S|-1}\left(1 - \frac{\mathrm{ED}\left(O_i, A_i\right)}{\max\left(\mathrm{len}\left(O_i\right), \mathrm{len}\left(A_i\right)\right)}\right),$$

где $|S|$ – число сценариев работы, $\mathrm{len}(p)$ – длина последовательности p, а $\mathrm{ED}\left(p_1, p_2\right)$ – редакционное расстояние Левенштейна между последовательностями p_1 и p_2.

Второй компонент ФП F_{LTL} оценивает соответствие автомата заданным темпоральным формулам. Верификатор, разработанный в [17], позволяет выделять 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 число достижимых из начального состояния переходов $t^i_{reachable}$. F_{LTL} вычисляется как среднее значение этой величины по всем LTL-формулам:

$$F_{LTL} = \frac{1}{k}\sum_{i=0}^{k-1}\frac{t^i_{checked}}{t_{reachable}},$$

где k – число LTL-формул. Отметим, что если автомат удовлетворяет формуле, то число проверенных переходов равно числу достижимых переходов, следовательно, максимальное значение F_{LTL} равно единице.

Наличие третьего компонента ФП определяется тем, что автоматы, содержащие небольшое число переходов, считаются более предпочтительными, чем автоматы с большим числом переходов.

4. Муравьиный алгоритм на основе графа мутаций

В данном разделе приводится краткое описание алгоритма *MuACO*, полную версию которого можно найти в [11]. *MuACO* (*Mutation-based Ant Colony Optimization*) – метаэвристический алгоритм поисковой оптимизации, предназначенный для построения конечных автоматов. Работа *MuACO* основана на использовании так называемого *графа мутаций*. Его вершины соответствуют конечным автоматам, а ребра – *мутациям* автоматов. Мутация – это небольшое изменение структуры автомата, вызванное применением *оператора мутации*. При решении задачи построения управляющих конечных автоматов по сценариям работы и темпоральным формулам в *MuACO* используются два оператора мутации. Первый из них выбирает случайный переход в автомате и изменяет состояние y, в которое этот переход ведет. Новое состояние

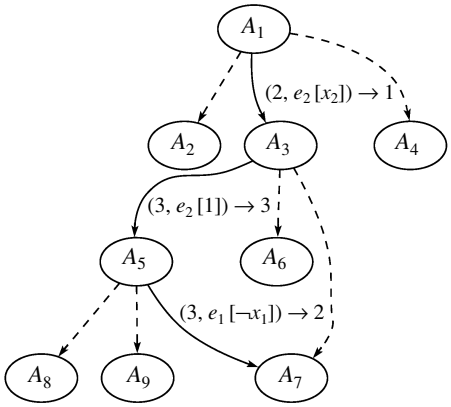

Рис. 2. Пример графа мутаций.

выбирается случайным образом среди всех состояний кроме y. Второй оператор последовательно рассматривает все состояния автомата и с определенной вероятностью добавляет новый или удаляет существующий переход в каждом из них.

Пример графа мутаций приведен на рис. 2. Запись на ребре $(3, e_1[\neg x_1]) \to 2$ означает, что соответствующая мутация изменила состояние, в которое ведет переход из состояния 3 по событию e_1 и формуле $\neg x_1$, на состояние 2.

На каждом ребре uv графа мутаций (u и v – вершины графа, соответствующие решениям-кандидатам) заданы величины эвристической информации η_{uv} и значения феромона τ_{uv}. Эвристическая информация на ребре uv вычисляется по формуле $\eta_{uv} = \max(\eta_{\min}, F(v) - F(u))$, где $\eta_{\min} = \text{const} = 10^{-3}$. Значения феромона τ_{uv} изначально равны $\tau_{\min} = \text{const} = 10^{-3}$ и изменяются в процессе работы алгоритма.

Будем рассматривать задачу максимизации ФП. Алгоритм начинает свою работу с единственного начального решения, которое может быть либо сгенерировано случайным образом, либо подано на вход. Это решение становится первой вершиной графа мутаций. Поиск решений производится колонией из N_{ants} муравьев. На каждой итерации колонии муравьи перемещаются по графу мутаций. В начале каждого шага муравей находится в определенной вершине графа u и выбирает, в какую вершину v выполнить переход. Для этого выполняется одно из следующих действий:

- С помощью метода рулетки [19] муравей выбирает следующую вершину v среди множества вершин N_u, инцидентных u. Вероятность p_{uv} выбрать вершину v вычисляется по классической в муравьиных алгоритмах формуле [20]:

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

  где $v \in N_u$, а $\alpha, \beta \in [0, 1]$ – величины, отражающие значимость значений феромона и эвристической информации соответственно;

- С помощью применения операторов мутации создается N_{mut} новых автоматов. Все новые автоматы добавляются в граф в качестве детей вершины v. Муравей переходит в вершину, соответствующую автомату с наибольшим значением ФП.

Каждый муравей хранит счетчик и значение ФП $f_{ant}^{\max}$ лучшего найденного им автомата. Если на текущем шаге муравей нашел решение, значение ФП которого больше $f_{ant}^{\max}$, то значение $f_{ant}^{\max}$ обновляется, а счетчик остается неизменным. В противном случае значение счетчика увеличивается на единицу. Когда значение счетчика становится равным n_{stag}, муравей принудительно останавливается.

Аналогичная процедура выполняется для колонии муравьев: хранится счетчик и максимальное значение ФП автомата, найденного муравьями. Если на текущей итерации колонии это значение ФП не увеличивается, то значение счетчика увеличивается на единицу. Если значение счетчика становится равным N_{stag}, считается, что алгоритм пришел в состояние стагнации. После этого алгоритм перезапускается.

В конце каждой итерации колонии производится обновление значений феромона на всех ребрах графа мутаций. Помимо эвристической информации и значений феромона, для каждого ребра хранится величина $F_{uv}^{\max}$, соответствующая наибольшему значению ФП автомата, найденного муравьем, посетившим это ребро.

Рассмотрим пути всех муравьев на данной итерации. Значением ФП муравья считается значение ФП лучшего автомата, найденного этим муравьем. Для каждого пути муравья выделяется отрезок от начала до вершины, соответствующей лучшему автомату на этом пути. Для всех ребер на этом отрезке обновляются значения $F_{uv}^{\max}$, после этого обновляются значения феромона:

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

где $\rho \in [0,1]$ – скорость испарения феромона.

5. Параллельный муравьиный алгоритм *pMuACO*

В данном разделе описывается параллельный муравьиный алгоритм, предложенный в [15]. Этот алгоритм предназначен для использования на многоядерных компьютерах с общей памятью, но может быть расширен для запуска на кластере. Будем называть этот алгоритм *pMuACO* (*parallel MuACO*).

Пусть параллельному алгоритму предоставлено m потоков для выполнения. В каждом потоке запускается алгоритм *MuACO*, причем для каждого алгоритма случайным образом генерируется отдельное начальное решение. Как было показано в [15], взаимодействие между отдельными потоками в алгоритме существенно повышает его эффективность. Реализовано два механизма взаимодействия, которые основаны на архиве K лучших решений всех потоков. Так, в каждый момент времени в ячейке K_i хранится лучшее решение, найденное i-м алгоритмом.

Первый механизм взаимодействия активизируется, когда i-й алгоритм *MuACO* приходит в состояние стагнации и перезапускается. При этом стар-

товое решение не генерируется случайным образом, а выбирается из архива лучших решений $K \setminus \{K_i\}$.

Второй механизм работает на каждой итерации колонии каждого алгоритма *MuACO* и использует генетический оператор *скрещивания* управляющих конечных автоматов. Оператор скрещивания принимает на вход два автомата (родители) и возвращает два автомата (потомки). Каждый из потомков содержит переходы, выбранные из обоих родителей. В [15] используется оператор скрещивания автоматов, предложенный в [10], учитывающий то, какие переходы родительских особей использовались при вычислении значения ФП.

Перед запуском очередной итерации i-го алгоритма из архива лучших решений случайным образом выбирается решение K_j, $j \neq i$. Оператор скрещивания применяется к K_i и K_j и возвращает два автомата A_1 и A_2. Далее, из получившихся автоматов выбирается тот, значение ФП которого больше – A_{best}. Автомат A_{best} добавляется в граф мутаций i-го алгоритма *MuACO* в качестве ребенка вершины, ассоциированной с K_i. В последующей итерации i-го алгоритма один муравей будет запущен из вершины, ассоциированной с A_{best}, а остальные муравьи, как и раньше, из вершины, ассоциированной с лучшим на тот момент решением i-го алгоритма K_i.

6. Предлагаемая модификация параллельного муравьиного алгоритма

В настоящей статье предлагается получать начальные решения для параллельного алгоритма *pMuACO* из сокращенных наборов сценариев работы. Данная модификация является расширением подхода, предложенного в [14], где для получения начального приближения используется точный метод построения управляющих конечных автоматов по сценариям работы. Точные методы основаны на сведении задачи построения управляющих конечных автоматов по сценариям работы к задачам выполнимости булевой формулы (SAT) и удовлетворения ограничений (CSP). Для решения задач SAT и CSP используются сторонние программные средства, такие как *cryptominisat* (`https://github.com/msoos/cryptominisat`) и *Choco* (`http://choco-solver.org/`). В [14] было показано, что совместное применение точного алгоритма и муравьиного алгоритма [11] существенно сокращает необходимое для построения автоматов время.

Очевидным развитием данного подхода является его применение для получения начального приближения в параллельных алгоритмах. Назовем этот алгоритм *psMuACO* (*parallel SAT MuACO*). Сначала с помощью точного алгоритма строится автомат, удовлетворяющий всем сценариям. Далее этот автомат используется в качестве начального решения для алгоритма *pMuACO*. Однако стоит учесть, что параллельные алгоритмы комбинаторной оптимизации достигают наибольшей эффективности, если поиск в разных потоках начинается с различных начальных решений. Это объясняется тем, что при различных начальных решениях существенно возрастает размер пространства поиска, исследуемого алгоритмом в единицу времени. При условии использования точных программных средств решения SAT и CSP алгоритмы на их основе обладают детерминированным поведением – по одному и тому же набору сценариев они всегда строят один и тот же автомат. Для исполь-

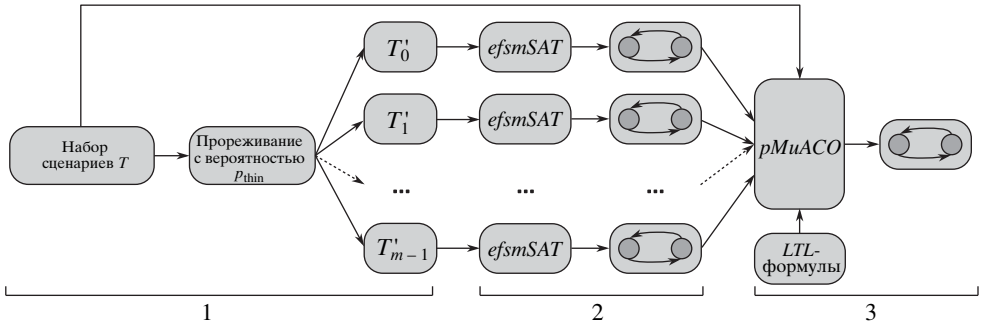

Рис. 3. Схема предлагаемого алгоритма *pstMuACO*.

зования этого подхода в совокупности с параллельным муравьиным алгоритмом необходимо с помощью алгоритмов на основе SAT или CSP получать несколько различных автоматов.

Предлагается решение этой проблемы путем построения начальных решений по сокращенным наборам сценариев, получаемым из исходного набора с помощью процедуры прореживания. Алгоритм состоит из трех этапов.

1. **Процедура прореживания сценариев**. Пусть необходимо по набору сценариев T построить n_{start} различных автоматов. Для этого сначала строится n_{start} сокращенных наборов сценариев $T'_0, \ldots, T'_{n_{start}-1}$. Каждый такой набор получается из исходного набора T путем прореживания – удаления каждого сценария с вероятностью p_{thin}.
2. **Построение начальных решений с помощью метода** *efsmSAT*. Каждый i-й автомат строится по сокращенному набору сценариев T'_i с помощью метода *efsmSAT* на основе сведения к задаче SAT [13]. Автоматы строятся независимо путем параллельного запуска программного средства *cryptominisat* решения задачи SAT. Данный подход не гарантирует получения различных автоматов, однако в экспериментах доля одинаковых автоматов оказывается близка к нулю.
3. **Построение автомата с помощью алгоритма** *pMuACO*. В случае когда $n_{start} = m$ i-й автомат, построенный по сокращенному набору сценариев T'_i, используется в качестве начального решения для i-го потока алгоритма. Если $n_{start} < m$, то начальное решение в каждом потоке выбирается случайным образом из множества сгенерированных начальных решений.

Число генерируемых с помощью SAT автоматов n_{start} и вероятность удаления сценария p_{thin} являются параметрами алгоритма. Схема предлагаемого алгоритма *pstMuACO* (*parallel SAT thinned out MuACO*) приведена на рис. 3.

7. Экспериментальное исследование

Опишем проведенные вычислительные эксперименты, процесс подготовки входных данных и выбор значений параметров алгоритмов. Для настройки значений параметров использовался сервер с 24-ядерным процессором AMD Opteron(TM) Processor 6234 @ 1,4 ГГц, а все вычислительные эксперименты

проводились на сервере с 64-ядерным процессором AMD Opteron(TM) Processor 6378 @ 2,4 ГГц.

7.1. Подготовка входных данных

Для проведения вычислительных экспериментов было подготовлено два набора входных данных: обучающий набор и тестовый набор. Каждый такой набор представляет собой множество примеров, каждый пример состоит из набора сценариев работы и набора *LTL*-формул. Опишем процедуру построения такого примера. Сначала случайным образом генерируется автомат со следующими параметрами: число состояний N, число входных событий $|E| = 2$, число входных переменных $|X| = 1$, число выходных воздействий $|Z| = 2$, последовательности выходных воздействий на переходах содержат от нуля до двух воздействий.

Далее, по автомату строится набор сценариев работы. Каждый сценарий соответствует случайному пути в автомате. Начало каждого сценария совпадает с начальным состоянием автомата. Длина сценария равна числу переходов, вошедших в путь. Всего генерируется $5N$ сценариев суммарной длиной $100N$.

Наконец, генерируются *LTL*-формулы, которым удовлетворяет автомат A. Алгоритму построения формул подается на вход автомат A, для которого их необходимо построить, и желаемое число формул n_{LTL} (здесь $n_{LTL} = 2$). Сначала с помощью программного средства *randltl* [21] генерируется $100n_{LTL}$ случайных LTL-формул, причем число вершин в дереве разбора формулы не превышает 15. Далее, с помощью верификатора [17] отбираются только те формулы, которым A удовлетворяет. Для дальнейшей проверки каждой такой формулы генерируется 50 случайных автоматов, для каждого из которых проверяется, верна ли для него формула. Формула добавляется в конечное множество формул, если она верна для не более чем половины этих автоматов. Процесс генерации формул прерывается, когда получено требуемое число формул.

7.2. Настройка значений параметров алгоритмов

В целях обеспечения корректности результатов вычислительных экспериментов значения параметров сравниваемых алгоритмов выбирались не вручную, а с помощью автоматической процедуры, называемой настройкой значений параметров (*parameter tuning*). Использовалось программное средство *irace* [22]. Суть процедуры настройки значений параметров в следующем. Пусть необходимо выбрать хорошие значения параметров алгоритма, предназначенного для решения задачи P. Программе-настройщику (*irace*) передается:

- описание параметров настраиваемого алгоритма: название, тип, область допустимых значений;
- набор примеров I задачи P;
- ограничение на время работы программы-настройщика.

Набор значений параметров алгоритма называется *конфигурацией*. Вначале *irace* генерирует множество случайных конфигураций. На каждой итерации неэффективные конфигурации отбрасываются с помощью следующего механизма. Настраиваемый алгоритм параметризуется конфигурацией и последовательно запускается для нескольких примеров из I, вычисляется значение целевой функции задачи P в каждом запуске. Конфигурация считается неэффективной и отбрасывается, если набор значений целевой функции для нее статистически хуже, чем для какой-либо другой конфигурации.

7.3. Изучение свойств предложенного модифицированного алгоритма

Целью первого блока экспериментов было изучение свойств предложенного алгоритма *pstMuACO*. Отдельная настройка параметров алгоритма не проводилась, использовались те же значения, что и в [15]: $N_{ants} = 4$, $N_{stag} = 28$, $n_{stag} = 45$, $N_{mut} = 44$, $\rho = 0{,}52$. Использовался набор из 100 примеров, каждый из которых был построен по автомату из десяти состояний. Алгоритму было отведено 16 потоков.

В первом эксперименте вероятность удаления сценария p_{thin} варьировалась в промежутке от нуля до 0,9 с шагом 0,1, число генерируемых с помощью SAT автоматов n_{start} варьировалось в пределах от нуля до 16. Графики зависимости медианного времени выполнения алгоритма от p_{thin} для различных значений n_{start} приведены на рис. 4.

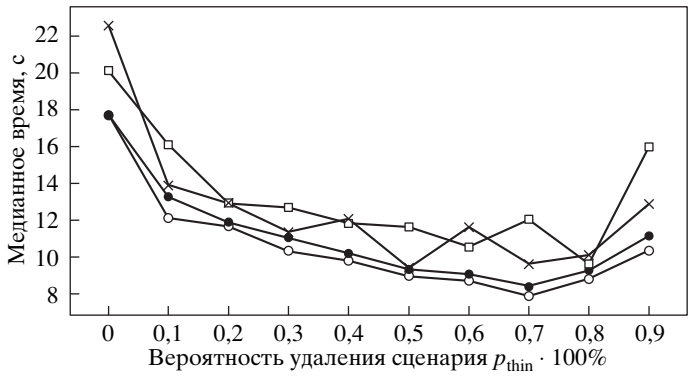

Рис. 4. Графики зависимости времени работы от вероятности удаления сценария: $n_{start} = 1$ (□), 2 (×), 6 (•), 16 (○).

Для всех значений n_{start} медианное время работы при увеличении p_{thin} сначала уменьшается, а затем, при приближении к максимальному значению p_{thin}, начинает увеличиваться. Результаты экспериментов показывают, что значения $p_{thin} = 0{,}7$ и $n_{start} = 16$ являются оптимальными для заданных значений параметров алгоритма.

7.4. Сравнение с предыдущими решениями

Целью второго блока экспериментов было сравнение предложенного в данной статье алгоритма *pstMuACO* с алгоритмом *psMuACO* и алгоритмом

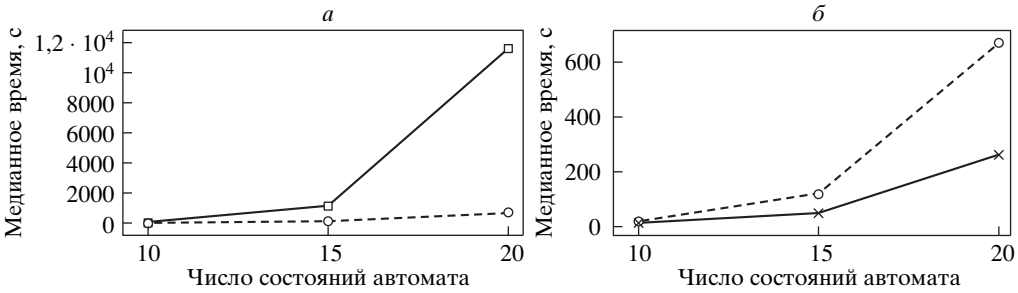

Рис. 5. Графики зависимости времени работы алгоритмов от числа состояний автомата: *pMuACO* (□), *psMuACO* (○), *pstMuACO* (×).

pMuACO, ранее предложенным в [15]. Значения параметров алгоритмов настраивались с помощью *irace*, на что для каждого алгоритма было отведено по 24 часа машинного времени. Обучающий набор для настройки состоял из 200 примеров, построенных по автоматам из 10–20 состояний. Полученные значения параметров алгоритмов приведены в табл. 1. Во всех алгоритмах значения параметров α и β были равны единице. Отметим, что полученные значения новых параметров $p_{thin} = 0{,}48$ и $n_{start} = 7$ отличаются от значений, признанных оптимальными в предыдущем эксперименте. Это является следствием того, что в настоящем эксперименте настраивался параллельный алгоритм *pstMuACO*, а в предыдущем были взяты некоторые значения параметров алгоритма *MuACO* и были подобраны оптимальные для них значения новых параметров.

Для тестирования было сгенерировано три набора из 50 примеров каждый, построенные по автоматам из 10, 15 и 20 состояний соответственно. Графики зависимости медианного времени работы алгоритмов от числа состояний приведены на рис. 5,*а* и рис. 5,*б*. Из графиков видно, что время работы алгоритма *psMuACO* существенно меньше времени работы алгоритма *pMuACO*. Алгоритм *pstMuACO*, в свою очередь, превосходит по быстродействию алгоритм *psMuACO*. Например, для автоматов из 20 состояний медиана времени работы алгоритма *pstMuACO* в 2,5 раза меньше медианы времени работы алгоритма *psMuACO* и в 17 раз меньше медианы времени работы алгорит-

Таблица 1. Значения параметров алгоритмов, полученные с помощью *irace*

Параметр	*pMuACO*	*psMuACO*	*pstMuACO*
Максимальное число шагов муравья без увеличения значения ФП n_{stag}	34	41	22
Максимальное число итераций колонии без увеличения значения ФП N_{stag}	33	35	17
Число мутаций N_{mut}	44	27	44
Число муравьев N_{ants}	4	4	17
Скорость испарения феромона ρ	0,2	0,6	0,21
Число генерируемых с помощью SAT решений n_{start}	–	–	7
Вероятность удаления сценария p_{thin}	–	–	0,48

Таблица 2. Медианное время работы алгоритма *pstMuACO* на данных увеличенной размерности, с.; пометка TL означает, что запуски не завершились за отведенное время

Число состояний	5	6	7	8	9	10
$\|X\| = 1, \|Z\| = \|E\| = 2$	3,2	4,2	5,5	5,4	6,5	13,3
$\|X\| = \|Z\| = \|E\| = 3$	47,0	38,6	74,9	176,1 (19 TL)	TL	TL

ма *pMuACO*. Отметим также, что для автоматов из 20 состояний в шести из 50 случаев алгоритмы *pMuACO* и *psMuACO* не завершили работу за 24 часа и были прерваны. Однако даже если бы они завершили свою работу за большее время, это не изменило бы медианное значение времени работы и графики на рис. 5,*а* и рис. 5,*б* остались бы неизменными.

Также было рассчитано среднее время работы для примеров, на которых все алгоритмы завершили свою работу менее чем за 24 часа. Например, для автоматов из 20 состояний этот показатель для алгоритма *pMuACO* составил 20588 с, для алгоритма *psMuACO* – 5421 с, а для алгоритма *pstMuACO* – 491 с. Таким образом, предложенный алгоритм *pstMuACO* в среднем более чем в 40 раз быстрее алгоритма *pMuACO* и более чем в 11 раз быстрее алгоритма *psMuACO*.

Для проверки статистической значимости различий во времени работы алгоритмов использовался тест Уилкоксона [23]. Тест отдельно проводился для каждого значения числа состояний и каждой пары алгоритмов. Нулевая гипотеза утверждала, что медиана разницы выборок равна нулю. Альтернативная гипотеза утверждала, что медиана разницы выборок меньше нуля. Предельный уровень значимости был равен 0,05. Для пары алгоритмов *pMuACO* и *psMuACO* были получены следующие p-value: $2{,}07 \cdot 10^{-6}$ (10 состояний), $7{,}14 \cdot 10^{-5}$ (15 состояний) и $1{,}25 \cdot 10^{-3}$ (20 состояний). Для пары алгоритмов *psMuACO* и *pstMuACO* были получены следующие p-value: 0,03 (10 состояний), $5{,}44 \cdot 10^{-7}$ (15 состояний) и $1{,}95 \cdot 10^{-5}$ (20 состояний). Эти результаты показывают, что для рассмотренных данных алгоритм *psMuACO* во всех случаях статистически быстрее алгоритма *pMuACO*, а алгоритм *pstMuACO* статистически быстрее алгоритма *psMuACO*.

Наконец, было проведено экспериментальное исследование на данных увеличенной размерности. Рассматривались автоматы, содержащие от пяти до десяти состояний, три входных события, три выходные переменные и три выходных воздействия. Последовательности выходных воздействий на переходах содержали от нуля до трех воздействий. Для каждого эксперимента было установлено ограничение по времени 300 с. Результаты экспериментов представлены в табл. 2. Исходя из приведенных данных можно сделать вывод о том, что увеличение размерности входных данных влечет существенное увеличение времени, необходимого для построения оптимального решения. Так, для N равного пяти, шести и семи медианное время работы алгоритма *pstMuACO* для данных увеличенной размерности примерно в 10 раз превышает время работы на данных небольшой размерности. Для больших значений N большинство запусков не завершилось за отведенное время.

8. Заключение

В статье предложен новый параллельный алгоритм построения управляющих конечных автоматов *pstMuACO*. Экспериментально показано, что новый алгоритм работает быстрее, чем ранее применявшиеся подходы. Так, для рассмотренных автоматов из 20 состояний новый алгоритм в среднем в 40 раз быстрее простого параллельного муравьиного алгоритма. Предложенный алгоритм может быть применен для автоматизированного построения надежных систем управления при условии небольшой размерности требуемых автоматов.

СПИСОК ЛИТЕРАТУРЫ

1. *Clarke E., Grumberg O., Peled D.* Model checking. Cambridge: MIT press, 1999.
2. *Поликарпова Н.И., Шалыто А.А.* Автоматное программирование. СПб.: Питер, 2009.
3. *Шалыто А.А.* Использование граф-схем и графов переходов при программной реализации алгоритмов логического управления. II // АиТ. 1996. № 7. С. 144–169.

   Shalyto A.A. Algorithmic Graph Schemes and Transition Graphs: Their Use in Software Realization of Logical Control Algorithms. II // Autom. Remote Control. 1996. V. 57. No. 7. P. 1027–1045.
4. *Вельдер С.Э., Лукин М.А., Шалыто А.А. и др.* Верификация автоматных программ. СПб.: Наука, 2011.
5. *Gold M.* Complexity of Automaton Identification from Given Data // Information and Control. 1978. V. 37. No. 3. P. 302–320.
6. *Ульянцев В.И.* Применение методов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов по сценариям работы: Бакалаврская работа [Электронный ресурс]. http://is.ifmo.ru/diploma-theses/2011/bachelor/ulyantsev/thesis.pdf
7. *Luke S.* Essentials of metaheuristics. Raleigh: Lulu, 2009.
8. *Карпенко А.П.* Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: уч. пос. М.: Изд-во МГТУ им. Н.Э. Баумана, 2014.
9. *Курейчик В.В., Курейчик В.М., Родзин С.И.* Теория эволюционных вычислений. М.: Физматлит, 2012.
10. *Tsarev F., Egorov K.* Finite State Machine Induction Using Genetic Algorithm Based on Testing and Model Checking // Proc. 13th ann. Conf. on Genetic and Evolutionary Computation Companion. N.Y. 2011. P. 759–762.
11. *Chivilikhin D., Ulyantsev V.* MuACO*sm*: A New Mutation-Based Ant Colony Optimization Algorithm for Learning Finite-State Machines // Proc. 15th ann. Conf. on Genetic and Evolutionary Computation. N.Y. 2013. P. 511–518.
12. *Heule M., Verwer S.* Exact DFA Identification Using SAT Solvers // Proc. 10th Int. Colloquium Conf. on Grammatical Inference: Theoretical Results and Applications. Berlin. 2010. P. 66–79.
13. *Ulyantsev V., Tsarev F.* Extended Finite-State Machine Induction Using SAT-solver // Proc. 10th Int. Conf. on Machine Learning and Applications. Los Alamitos. 2011. V. 2. P. 346–349.
14. *Chivilikhin D., Ulyantsev V., Shalyto A.* Combining Exact and Metaheuristic Techniques for Learning Extended Finite-State Machines from Test Scenarios and Temporal Properties // Proc. 13th Int. Conf. on Machine Learning and Applications. Detroit. 2014. P. 350–355.

15. *Chivilikhin D., Ulyantsev V., Shalyto A.* Extended Finite-State Machine Inference with Parallel Ant Colony Based Algorithms // Proc. 5th Int. Student Workshop on Bioinspired Optimization Methods and their Applications. Ljubljana. 2014. P. 117–126.
16. *Chivilikhin D., Ulyantsev V.* Inferring Automata-Based Programs from Specification with Mutation-Based Ant Colony Optimization // Proc. 16th Conf. on Genetic and Evolutionary Computation Companion. N.Y. 2014. P. 67–68.
17. *Егоров К.В.* Генерация управляющих автоматов на основе генетического программирования и верификации, 2013. Канд. дисс. НИУ ИТМО. [Электронный ресурс]. http://is.ifmo.ru/disser/egorov_disser.pdf
18. *Левенштейн В.И.* Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. АН СССР. 1965. №. 4. С. 845–848.
19. *Baker J.* Reducing Bias and Inefficiency in the Selection Algorithm // Proc. 2nd Int. Conf. on Genetic Algorithms and their application. Hillsdale. 1987. P. 14–21.
20. *Dorigo M., Maniezzo V., Colorni A.* Ant System: Optimization by a Colony of Cooperating Agents // IEEE Trans. Systems, Man and Cybernetics. 1996. V. 25. No. 1. P. 29–41.
21. *Duret-Lutz A.* Manipulating LTL Formulas using SPOT 1.0. Automated Technology for Verification and Analysis (под. ред. Hung D., Ogawa M.). Lect. Notes in Computer Science, 2013. V. 8172. P. 442–445.
22. *López-Ibáñez M., Dubois-Lacoste J., Stützle T., et al.* The `irace` package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, 2011.
23. *Wilcoxon F.* Individual Comparisons by Ranking Methods // Biometrics Bulletin. 1945. V. 1. No. 6. P. 80–83.

Статья представлена к публикации членом редколлегии О.П. Кузнецовым.

Поступила в редакцию 10.12.2014