Министерство образования и науки Российской Федерации
Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

На правах рукописи

Чивилихин Даниил Сергеевич

Генерация конечных автоматов на основе муравьиных алгоритмов

Специальность 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
докт. техн. наук, профессор
Шалыто А. А.

Санкт-Петербург — 2015

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность и степень разработанности проблемы. В последнее время для реализации управления объектами со сложным поведением все чаще применяется *автоматное программирование* [1—4]. Это парадигма программирования, в которой предлагается реализовывать программы в виде *автоматизированных объектов управления*, каждый из которых состоит из *системы управления*, представленной одним или несколькими *управляющими конечными автоматами*, и *объекта управления*. Основными достоинствами автоматного программирования являются наглядность описания поведения программ, возможность автоматической генерации кода и высокий уровень автоматизации верификации автоматных программ с помощью метода *проверки моделей* (*model checking*) [5, 26]. Идеи автоматного программирования используются, например, в средах разработки *MATLAB/Stateflow*, *IBM Rational Rhapsody*, а также в стандартах разработки приложений для промышленной автоматики, например *IEC 61131* [112] и *IEC 61499* [113].

В некоторых случаях конечные автоматы для прикладных задач удается построить эвристически, однако известны примеры, когда построенные вручную автоматы неоптимальны, или их вовсе не удается построить [6, 7, 27]. Это указывает на актуальность повышения степени автоматизации процесса разработки автоматных программ.

В последнее время проводится все больше исследований в области *поисковой инженерии программного обеспечения* (*search-based software engineering*) [28]. Это новая, стремительно развивающаяся область исследований, по которой на сентябрь 2015 года насчитывается всего лишь 1389 публикаций [114]. При этом до 1992 года было опубликовано менее десяти работ, а в 2014 году вышло более 200 научных статей. В рамках этого подхода для автоматизированного решения вычислительно сложных задач, возникающих при разработке программного обеспечения [29—32], применяются *метаэвристические* алгоритмы *поисковой оптимизации*, позволяющие в большинстве

случаев находить близкие к оптимальным решения. Примерами метаэвристик являются эволюционные [33] и генетические алгоритмы [34], эволюционные стратегии [35] и муравьиные алгоритмы [36—40]. Они применяются, поскольку реализуют неполный направленный перебор решений в *пространстве поиска* – множестве всех возможных решений задачи. Обычно поиск начинается со случайного решения, которое постепенно улучшается путем внесения в него небольших случайных изменений. Для оценки степени соответствия решения условиям задачи применяется так называемая *функция присособленности* (ФП).

Большинство существующих методов генерации конечных автоматов основано либо на использовании примеров поведения (сценариев или тестов) [41—47], либо на моделировании, которое позволяет тестировать автоматы [6, 7, 27, 8, 48, 49, 9]. Применение этих подходов не дает никаких гарантий относительно поведения сгенерированных автоматов на данных, не использованных для их генерации. Поэтому в последнее время получили развитие методы, в которых наряду со сценариями или тестами используется проверка *темпоральных свойств* [10—12, 50—52], позволяющих задать более общие закономерности поведения. Эти свойства задаются с помощью формул на языках темпоральной логики, например *Linear Temporal Logic* (*LTL*). Такие методы гарантируют, что поведение автомата на любых входных данных будет соответствовать заданным примерам поведения и темпоральным формулам, но не более того. При этом известно, что задача генерации конечного автомата, удовлетворяющего *LTL*-формуле, является 2EXPTIME-полной [53], а число состояний сгенерированного автомата в худшем случае дважды экспоненциально зависит от длины формулы. Эти оценки также можно распространить на случай сценариев, так как их можно записать в виде *LTL*-формул.

Известные методы генерации конечных автоматов с учетом темпоральных формул чаще всего основаны на генетических алгоритмах [10—12, 50, 52] и так называемой *AE*-парадигме [54—57]. Методы первой группы на основе

генетических алгоритмов [10—12, 52] позволяют генерировать автоматы по примерам поведения с учетом заданных *LTL*-формул. Недостатком этих методов является то, что для нахождения решения может потребоваться очень много времени.

Вторая группа методов основана на *AE*-парадигме [54—57], в которой программа с входным сигналом x и выходным сигналом y, удовлетворяющая темпоральной формуле $\varphi(x, y)$, конструируется как побочный продукт доказательства теоремы $(\forall x)(\exists y)\varphi(x, y)$. Название парадигмы происходит от английских терминов для кванторов $\forall$ (*for All*) и $\exists$ (*Exists*). Методы, основанные на этой парадигме [54—57], являются *точными* в том смысле, что позволяют найти решение, если оно существует, или доказать, что решения нет. В силу высокой вычислительной сложности эти методы зачастую работают дольше генетических алгоритмов. Еще одним их недостатком является большое число состояний генерируемых автоматов. Поэтому исследования, направленные на развитие методов генерации конечных автоматов, лишенных указанных недостатков, являются **актуальными**.

Дополнительной сложностью при применении метаэвристических алгоритмов для генерации конечных автоматов является «плохой» *ландшафт функции приспособленности*. В простейшем случае двухпараметрического пространства поиска, ландшафт ФП – поверхность, задающая график ФП. Например, диссертантом были изучены свойства ландшафта ФП в задаче о построении автомата [143], управляющего агентом в некоторой игре на тороидальном поле [27]. Эта задача является одной из классических модельных задач для изучения свойств метаэвристических алгоритмов. Было обнаружено, что решения, незначительно отличающиеся от оптимального, обладают большим разбросом значений ФП. Фактически, это означает, что метаэвристический алгоритм может найти оптимальное решение лишь случайно. Это наблюдение указывает на то, что метаэвристический алгоритм генерации конечных автоматов должен производить активный поиск в окрестности субоп-

тимальных решений. Одним из вариантов реализации такого подхода является использование долговременной общей памяти по аналогии с *муравьиными алгоритмами.*

Муравьиные алгоритмы [36—40] – это семейство метаэвристик для решения задач оптимизации на графах, инспирированных наблюдениями за поведением муравьев в процессе поиска пищи. Отличительной особенностью муравьиных алгоритмов является использование отдельными муравьями-агентами *общей памяти* об исследованной за время работы алгоритма области пространства поиска. Отметим, что ранее муравьиные алгоритмы для генерации автоматов не использовались. Настоящая работа направлена на *разработку более эффективных по сравнению с существующими методов генерации конечных автоматов с учетом темпоральных формул* на основе *муравьиных алгоритмов.*

В соответствии с паспортом специальности 05.13.11 – «Математическое обеспечение вычислительных машин, комплексов и компьютерных сетей», настоящая диссертация относится к области исследований «1. Модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования».

Целью работы является развитие существующих подходов к генерации конечных автоматов по сценариям работы и темпоральным формулам за счет применения муравьиных алгоритмов.

Задачи диссертационной работы состоят в следующем.

1. Разработать метод генерации конечных автоматов по сценариям работы и темпоральным формулам на основе муравьиного алгоритма.
2. Разработать методы генерации конечных автоматов по сценариям работы и темпоральным формулам на основе параллельных муравьиных алгоритмов.
3. Разработать инструментальное средство, реализующее предложенные методы.

4. Внедрить результаты работы при генерации автоматной логики для базисных функциональных блоков стандарта *IEC 61499* и в учебный процесс.

Научная новизна. В работе получены следующие новые научные результаты, которые выносятся на защиту.

1. Метод генерации конечных автоматов по сценариям работы и темпоральным формулам, основанный на муравьином алгоритме, отличающемся от существующих тем, что в нем используется предложенный автором граф мутаций. Показано, что предложенный метод работает быстрее известных методов на основе генетических алгоритмов и AE-парадигмы.
2. Метод генерации конечных автоматов по сценариям работы и темпоральным формулам, совмещающий параллельный муравьиный алгоритм, точный метод генерации конечных автоматов по сценариям работы на основе решения задачи выполнимости булевой формулы и процедуру прореживания сценариев. Показано, что метод работает быстрее параллельного генетического алгоритма и параллельного муравьиного алгоритма.

Методология и методы исследования. Методологическая основа диссертации – итеративная генерация автоматов с последующим отбором и ранжированием по степени соответствия входным данным. В работе используются методы теории автоматов, эволюционных вычислений, темпоральной логики, дискретной математики и математической статистики.

Достоверность научных положений и выводов, полученных в диссертации, подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, результатами экспериментов и их статистическим анализом.

Теоретическое значение работы состоит в том, что показана применимость муравьиных алгоритмов для решения задач, в которых пространство поиска не имеет естественного графового представления.

Практическое значение работы состоит в том, что разработанные методы генерации конечных автоматов реализованы в виде инструментального программного средства, позволяющего генерировать автоматы существенно быстрее известных методов.

Внедрение результатов работы.

Результаты диссертации были внедрены в Технологическом университете Лулео (Швеция) при генерации автоматной логики для базисных функциональных блоков стандарта *IEC 61499* и использованы в учебном процессе в Университете ИТМО на кафедре «Компьютерные технологии» в рамках курсов «Теория автоматов и программирование» и «Генетическое программирование».

Апробация результатов работы. Основные результаты работы докладывались на 14 конференциях: III-я и VI-я Всероссийская конференция по проблемам информатики СПИСОК (2012, 2013, Матмех СПбГУ), 14th, 15th and 16th Genetic and Evolutionary Computation Conference (2012, Филадельфия, 2013, Амстердам, 2014, Ванкувер), 8th International Conference on Swarm Intelligence (2012, Брюссель); VII-я Международная научно-практическая конференция «Интегрированные модели и мягкие вычисления в искусственном интеллекте» (2013, Коломна), 7th IFAC Conference on Manufacturing Modelling, Management, and Control (2013, Санкт-Петербург), 12th and 13th International Conference on Machine Learning and Applications (2013, Майами, 2014, Детройт), 1st BRICS countries Congress on Computational Intelligence (2013, Ресифи), 6th International Student Workshop on Bioinspired Optimization Methods and their Applications (2014, Любляна), XII-е Всероссийское совещание по проблемам управления (2014, Москва), 13th IEEE International Conference on Industrial Informatics (2015, Кембридж) и 13th

IEEE International Symposium on Parallel and Distributed Processing with Applications (2015, Хельсинки).

Личный вклад автора. Решение задач диссертации, разработанные методы, алгоритмы и их реализация принадлежат лично автору.

Публикации. Основные результаты по теме диссертации изложены в 18 публикациях [129—138, 144—147, 139—142], три из которых изданы в российских журналах, рекомендованных ВАК [129—131], 11 – в изданиях, индексируемых в международных базах цитирования *Web of Science* [133, 134, 136, 138—140, 142] и *Scopus* [132—142]. Доля диссертанта в работах, выполненных в соавторстве, указана в списке публикаций.

Свидетельства о регистрации программы для ЭВМ. Автором по теме диссертации получено два свидетельства о регистрации программы для ЭВМ.

1. № 2013661249 от 3 декабря 2013 года «Программное средство, реализующее муравьиный алгоритм для построения конечных автоматов», авторы Чивилихин Д.С., Ульянцев В.И.
2. № 2015610291 от 12 января 2015 года «Библиотека параллельных муравьиных алгоритмов для построения управляющих конечных автоматов», авторы Чивилихин Д.С., Ульянцев В.И.

Участие в научно-исследовательских работах. Результаты диссертации были получены при выполнении научно-исследовательских работ по следующим темам: «Разработка методов автоматизированного построения надежного программного обеспечения на основе автоматного подхода по обучающим примерам и темпоральным свойствам» (Грант РФФИ № 14-07-31337 мол_а) и «Разработка муравьиных алгоритмов для построения управляющих конечных автоматов» (Грант РФФИ № 14-01-00551 А). Автор является победителем конкурса грантов Санкт-Петербурга для студентов, аспирантов, молодых ученых, молодых кандидатов наук 2013 г., тема проекта — «Разработка методов генерации управляющих конечных автоматов на основе мура-

вьиных алгоритмов», а также стипендиатом Президента РФ для аспирантов, обучающихся по приоритетным направлениям модернизации и технологического развития экономики России на 2015/16 учебный год.

Объем и структура работы. Диссертация состоит из введения, пяти глав, заключения и двух приложений. Объем диссертации составляет 144 страницы, с 33 рисунками, 6 таблицами и 9 листингами. Список литературы содержит 147 наименований.

В **первой главе** приводится обзор работ, посвященных автоматному программированию, метаэвристическим алгоритмам оптимизации, и методам генерации конечных автоматов. Приводится формальная постановка задачи генерации конечных автоматов по сценариям работы и темпоральным формулам. На основе результатов обзора формулируются задачи, решаемые в диссертации.

Вторая глава посвящена предлагаемому методу *MuACO* (*Mutation-based Ant Colony Optimization*) для генерации конечных автоматов по сценариям работы и темпоральным формулам, который основан на муравьином алгоритме с предложенным графом мутаций. Дается описание комбинированного метода *SAT+MuACO*, в котором начальное решение для *MuACO* генерируется с помощью точного метода *efsmSAT* на основе решения задачи выполнимости. Далее приводится описание экспериментов по сравнению предлагаемых методов с генетическим алгоритмом, а также с методами на основе AE-парадигмы. Излагается использованный способ генерации случайных тестовых данных, процедура настройки значений параметров алгоритмов и протокол проведения экспериментов. Результаты экспериментальных запусков представлены в виде графиков медианных значений числа вычислений функции приспособленности и ящичных диаграмм распределений времени работы. Приводятся результаты статистического анализа полученных результатов.

В **третьей главе** описываются предлагаемые методы генерации конечных автоматов по сценариям работы и темпоральным формулам на основе параллельных муравьиных алгоритмов. В первом методе *pMuACO* (*parallel MuACO*) в каждом потоке запускается метод *MuACO*, а взаимодействие между отдельными потоками реализовано с помощью архива лучших решений и операции кроссовера. Второй метод *psMuACO* (*parallel SAT MuACO*) отличается от *pMuACO* тем, что в качестве начального решения во всех потоках используется автомат, сгенерированный точным методом *efsmSAT* генерации автоматов только по сценариям работы, основанным на решении задачи выполнимости. Третий метод *pstMuACO* (*parallel SAT thin-out MuACO*) совмещает процедуру прореживания сценариев, алгоритм *efsmSAT* и метод *pMuACO*. Наконец, приводится описание экспериментов по сравнению эффективности предложенных методов *pMuACO*, *psMuACO* и *pstMuACO*.

В **четвертой главе** описывается разработанное программное средство, реализующее предложенные в диссертации методы.

В **пятой главе** описывается внедрение предложенных в диссертации методов при генерации диаграмм управления выполнением базисных функциональных блоков стандарта *IEC 61499*.

В **заключении** сформулированы результаты, полученные в диссертации.

ГЛАВА 1. АВТОМАТНОЕ ПРОГРАММИРОВАНИЕ, КОНЕЧНЫЕ АВТОМАТЫ, МЕТОДЫ ГЕНЕРАЦИИ КОНЕЧНЫХ АВТОМАТОВ

В данной главе приводятся основные сведения об автоматном программировании, различных типах конечных автоматов, методах генерации конечных автоматов. В завершение главы (раздел 1.7) представлены и обоснованы задачи, решаемые в диссертационной работе.

1.1. Автоматное программирование

Автоматное программирование [1—4] предполагает особый способ разработки программ, в которых есть *сущности со сложным поведением*. Говорят, что сущность обладает *простым поведением*, если она всегда одинаково реагирует на одинаковые входные воздействия. Если же сущность обладает сложным поведением, то ее реакции на одинаковые входные воздействия могут различаться и зависеть не только от самих входных воздействий, но и от предыстории.

Термин «состояние» является базовым в автоматном программировании. Состояние инкапсулирует всю информацию о прошлом системы, необходимую для формирования реакции на входные воздействия. Таким образом, при использовании термина «состояние» для описания систем со сложным поведением знание предыстории не требуется.

Неформально можно выделить *управляющие состояния* и *вычислительные состояния*. Число управляющих состояний обычно невелико, и каждое из них несет определенный смысл. Число вычислительных состояний, напротив, может быть очень велико или бесконечно. Смысла вычислительные состояний обычно не несут и отличаются друг от друга лишь количественно. Далее под состоянием всегда будем понимать управляющее состояние.

Входное воздействие – это вектор, состоящий из *событий* и значений *входных переменных*. Совокупность правил, по которым, в зависимости от

входных воздействий, происходит смена состояний, называется *функцией переходов*. В автоматном программировании под *конечным автоматом* понимается совокупность конечного *множества состояний*, *функции переходов*, а также *функции выходов*, которая определяет правила формирования *выходных воздействий*.

Сущность со сложным поведением в автоматном программировании предлагается представлять в виде так называемого *автоматизированного объекта управления*, состоящего из *управляющей* и *управляемой* частей. Управляющая часть реализует логику – выбор выходных воздействий в зависимости от состояния и входного воздействия. Управляющая часть называется *системой управления*, а автомат, реализующий ее, называется *управляющим конечным автоматом*. Управляемая часть называется *объектом управления* и связана с управлящей частью *прямой связью* – реализует выполнение действий, выбранных управляющей частью. Входные воздействия в простейшем случае поступают только из *внешней среды*. Также управляющая и управляемая части могут быть связаны *обратной связью*: входные события могут поступать не только из внешней среды, но и от объекта управления. В диссертации рассматриваются такие автоматизированные объекты управления, в которых управляющая часть реализована с использованием одного управляющего конечного автомата. Схема автоматизированного объекта управления приведена на рисунке 1, обратная связь изображена пунктирной линией.

Парадигма автоматного программирования состоит в представлении сущностей со сложным поведением в виде автоматизированных объектов управления.

Управляющим конечным автоматом [1] будем называть семерку $\langle X, E, Y, y_0, Z, \phi, \delta \rangle$, где:

— X – множество булевых входных переменных;

— E – множество входных событий;

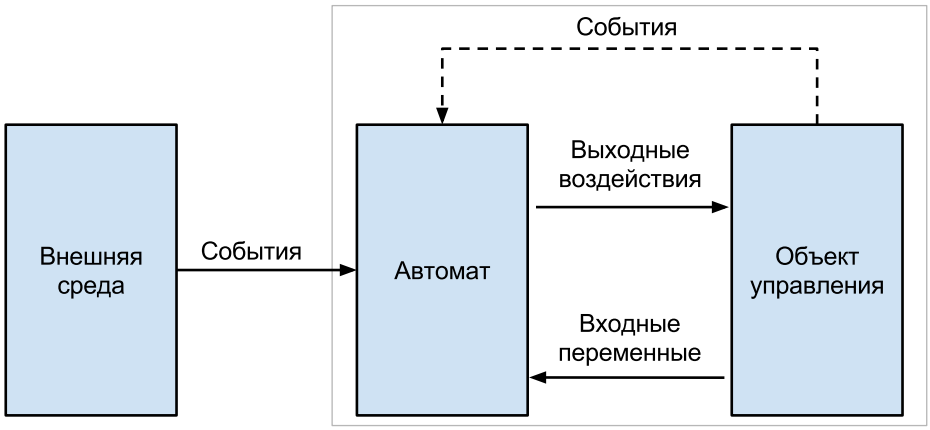

Рисунок 1 – Схема автоматизированного объекта управления

— 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;
— $\delta\colon Y \times E \times 2^X \to Z^*$ – функция выходов.

Отличительной особенностью управляющих конечных автоматов является использование сложных логических условий на переходах [115].

Графически конечные автоматы обычно изображаются в виде графов переходов. Пример графа переходов управляющего конечного автомата из трех состояний приведен на рисунке 2. 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. Далее будем считать автомат и граф его переходов взаимозаменяемыми понятиями.

Одним из достоинств автоматного программирования является высокий уровень автоматизации верификации [5, 12] автоматных программ с помощью *метода проверки моделей* (*model checking*) [26, 13]. Верификация на основе этого метода позволяет проверить, удовлетворяет ли программа некоторым *темпоральным свойствам*. Темпоральные свойства описывают временные

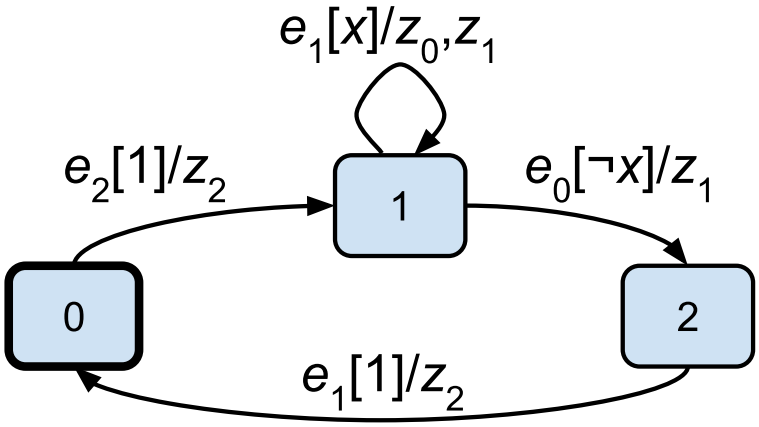

Рисунок 2 – Пример графа управляющего конечного автомата

связи между последовательными состояниями программы и задаются в виде формул на *языках темпоральной логики*, например, *LTL*.

В методе проверки моделей процесс верификации включает построение формальной модели программы и проверяемых свойств, а также проверку выполнения свойств для модели. Если свойство не выполняется, то выводится контрпример, сначала на уровне модели, а затем на уровне программы.

При верификации ядра автоматной программы (управляющего конечного автомата) этапы построения формальной модели программы и проверяемых свойств, а также преобразования контрпримера из терминов модели в термины программы могут быть выполнены автоматически [5]. Это привело к развитию методов, позволяющих по примерам поведения (сценариям, тестам) и темпоральным формулам автоматически сгенерировать удовлетворяющий им автомат [11, 12].

Ярким примером применения автоматного подхода в промышленности является международный стандарт *IEC 61499* [58], определяющий открытую архитектуру для разработки событийных систем распределенного и централизованного управления и автоматизации. Элементарным компонентом стандарта *IEC 61499* является *функциональный блок*. Функциональные блоки характеризуются *интерфейсом*, который определяет использующиеся входные/выходные события и входные/выходные переменные. *Ба-*

зисный функциональный блок задается с помощью событийной модели, называемой *диаграммой управления выполнением* (*Execution Control Chart*), которая представляет собой конечный автомат Мура специального вида. *Составной* функциональный блок задается сетью ФБ, среди которых могут быть как базисные, так и составные. Благодаря их конечно-автоматной основе, программы, реализованные с помощью функциональных блоков, также поддаются автоматизированной верификации [59, 60].

1.2. Метаэвристические алгоритмы

При решении сложных комбинаторных задач встает вопрос быстродействия алгоритмов. Так, для NP-трудных задач точные алгоритмы, гарантированно находящие оптимальное решение задачи, будут работать, в худшем случае, за экспоненциальное от размера входных данных время. Поэтому при решении таких задач часто приходится пользоваться *эвристическими алгоритмами*, позволяющими в большинстве случаев находить близкие к оптимальным решения. Основной сложностью при применении эвристических алгоритмов является проблема выхода из локальных оптимумов.

Поэтому были изобретены *метаэвристические алгоритмы* или *метаэвристики* [14, 61] – эвристики более высокого уровня, направляющие работу проблемно-ориентированных эвристик в более выгодные области пространства поиска. Существуют десятки самых различных метаэвристических алгоритмов. Многие метаэвристики являются *биоинспирированными* – основанными на процессах, происходящих в природе [15], например, *эволюционные алгоритмы* [33, 15, 16] и *муравьиные алгоритмы* [36—39]. Примерами других популярных метаэвристических алгоритмов являются:

— метод имитации отжига (*simulated annealing*) [62];
— поиск с ограничениями (*tabu search*) [63].
— метод оптимизации роем частиц (*particle swarm optimization*) [64];
— пчелиный алгорим (*bees algorithm*) [65, 17];

— светлячковый алгоритм (*firefly algorithm*) [66];

— алгоритм искусственной пчелиной колонии (*artificial bee colony*) [67];

— искусственные иммунные системы (*artifical immune systems*) [68].

В диссертации применяются именно муравьиные алгоритмы, поскольку они используют долговременную общую память об исследованной области пространства поиска. Как показано в разделе 1.5, это их свойство может быть полезным при решении задач генерации конечных автоматов.

1.2.1. Эволюционные алгоритмы

Эволюционные алгоритмы (ЭА) [33, 16] – это метаэвристики, принципы работы которых инспирированы законами эволюции живых существ. В основе эволюционных алгоритмов лежит принцип естественного отбора. Алгоритм оперирует *популяцией* или *поколением* – набором решений-кандидатов некоторой задачи оптимизации. Особи первого поколения обычно генерируются случайным образом. На каждой итерации алгоритма происходит формирование нового поколения на основе текущего. При этом к особям применяются *операторы мутации* и/или *скрещивания*.

Особи, которые получат право скрещиваться или перейти в новое поколение, определяются с помощью *оператора селекции*. Оператор мутации применяется к одной особи и изменяет одну или несколько частей представления особи – *генов*. Оператор скрещивания обычно применяется к двум особям и выдает на выход также две особи. Принцип его работы состоит в том, что родительские особи обмениваются генами таким образом, что каждая дочерняя особь содержит часть генов каждой из родительских особей. Степень соответствия особей-решений условиям задачи определяется с помощью так называемой *функции приспособленности* (ФП).

Выделяют два простейших эволюционных алгоритма: (μ, λ)-ЭА и $(\mu+\lambda)$-ЭА [33]. В обоих алгоритмах популяция состоит из μ особей, оператор скре-

щивания в них не применяется. В (μ, λ)-ЭА в новую популяцию переходят μ лучших из λ вновь сгенерированных особей.

В ($\mu+\lambda$)-ЭА в новую популяцию переходят μ особей из λ вновь сгенерированных и μ особей текущей популяции. Отдельно стоит выделить (1+1)-ЭА как наиболее простой эволюционный алгоритм. В этом алгоритме популяция состоит из единственной особи. На каждой итерации к текущей особи применяется оператор мутации. Текущая особь заменяется на новую, если значение ФП новой особи не меньше значения ФП текущей особи (в случае задачи максимизации ФП).

Основной отличительной особенностью *генетических алгоритмов* [34, 18] является то, что в них наряду с операторами мутации и селекции применяется оператор скрещивания (кроссовера). Опишем схему классического генетического алгоритма. На каждой итерации с помощью оператора селекции определяются особи, которые получают возможность скрещиваться. Выбранные особи подвергаются операции кроссовера, дочерние особи добавляются в новую популяцию. Когда новая популяция уже сформирована, к каждой особи с некоторой вероятностью применяется оператор мутации. Существуют также другие, более сложные модели генетических алгоритмов, например, островная модель. В ней популяция разделена на несколько частей-островов, каждая из которых развивается независимо от других. Раз в несколько итераций происходит миграция особей между островами.

В настоящее время под эволюционными алгоритмами или, точнее, под *эволюционными вычислениями* понимают большой раздел вычислительного интеллекта, включающий в себя, например, генетические алгоритмы [34, 18], генетическое программирование [69], эволюционные стратегии [35] и дифференциальную эволюцию [70]. Перечисленные методы находят свое применение в автоматической генерации программ [71], робототехнике [72], выращивании аппаратных средств [73], решении задач оптимизации транспортных потоков [74] и многих других областях. Так, например, специалисты *NASA*

в работе [73] применили эволюционный алгоритм для автоматического получения формы антенны для космического аппарата миссии *ST5*.

Отдельно стоит упомянуть о проблеме восприятия эвристических алгоритмов в целом и, в частности, эволюционных алгоритмов представителями индустрии. Эти алгоритмы в большинстве своем являются рандомизированными, и многие компании отказываются от их использования именно по этой причине. Однако уже сейчас существует несколько компаний, предоставляющих решения различных задач оптимизации и планирования производства, основанные на эволюционных алгоритмах:

— *Natural Selection, Inc.* [116];
— *Red Cedar Technology* [117];
— *NeuroSolutions* [118].

1.2.2. Муравьиные алгоритмы

Муравьиные алгоритмы – семейство алгоритмов оптимизации, работа которых основана на поведении колонии муравьев в процессе поиска источников пищи. Первый муравьиный алгоритм *Ant System* был разработан M. Dorigo и применялся для решения задачи о коммивояжере [36]. Муравьиные алгоритмы успешно применяются для решения таких сложных комбинаторных задач, как, например, задача о рюкзаке, задача об упаковке в контейнеры, квадратичная задача о назначениях [37].

1.2.2.1. Предпосылки к созданию муравьиных алгоритмов

Первой работой, заложившей основу создания муравьиных алгоритмов, принято считать работу *Grasse* 1959 года об изучении поведения термитов, в которой для его объяснения была предложена теория *стигмергии* (*stigmergy*). Это слово образовано от греческих *stigma* (знак, отметка) и *ergon* (работа, действие). Буквально, стигмергия – непрямое взаимодействие особей в процессе выполнения работы в некоторой среде путем ее модификации

(оставление знаков, отметок). Как оказалось, данная теория применима не только к термитам, но и к некоторым видам муравьев.

Непосредственной предпосылкой создания муравьиных алгоритмов стала работа [75], в которой изучалось поведение аргентинских муравьев *Iridomyrmex humilis* в процессе поиска источников пищи. Были проведены два эксперимента с так называемыми «двойными мостами».

В первом эксперименте между колонией муравьев и источником пищи располагались два разветвляющихся пути одинаковой длины. Эксперимент состоял в следующем: муравьям разрешали искать источник пищи до тех пор, пока найденный ими путь не станет устойчивым. Эксперимент повторялся несколько раз. В результате выяснилось, что муравьи сходятся к выбору одного из путей почти равновероятно. Во втором эксперименте длину одного из путей увеличили в два раза. В этом случае, в подавляющем большинстве экспериментов муравьи выбирали более короткий путь.

Такое интеллектуальное поведение, казалось бы, простых муравьев-рабочих объясняется теорией стигмергии. Муравьи откладывают на своем пути особое химическое вещество – феромон. Высокая концентрация феромона в определенной области на пути муравья привлекает его, увеличивая вероятность того, что муравей проложит свой путь через эту область. Феромон, естественно, со временем испаряется, его концентрация уменьшается. Чем короче путь муравья до источника пищи и обратно, тем меньше времени потребуется для его преодоления, тем больше концентрация феромона на его пути, тем больше муравьев последуют этому пути.

В той же работе была предложена вероятностная модель поведения муравьев в экспериментах с двойными мостами, были записаны дифференциальные уравнения, описывающие эволюцию этой модели. Модель не учитывала испарение феромона, так как в экспериментах было установлено, что время, требующееся для установления устойчивого пути муравьев примерно равно времени жизни феромона. В данной модели муравьи откладывали фе-

ромон как на пути к источнику пищи, так и на обратном пути. Позже было показано, что муравьи, откладывающие феромон только на обратном пути, неспособны найти кратчайший путь до источника пищи.

1.2.2.2. Решение задач комбинаторной оптимизации с помощью муравьиных алгоритмов

Опишем классический способ решения комбинаторных задач с помощью муравьиных алгоритмов [37]. Заметим, что муравьиные алгоритмы – это алгоритмы глобальной оптимизации: если алгоритму будет предоставлено достаточно времени (в пределе, бесконечно много времени), он найдет оптимальное решение [37].

Пусть некая комбинаторная задача представлена в виде тройки $\left(\hat{S}, \hat{f}, \Omega\right)$, где $\hat{S}$ – *множество решений-кандидатов*, $\hat{f}$ – *целевая функция* и Ω – *множество ограничений*, определяющее *допустимые решения*. Задача состоит в нахождении глобально оптимального допустимого решения s^*. Описанная задача следующим образом сводится к задаче, которую можно решить с помощью муравьиного алгоритма. Пусть $C = \{c_1, \ldots, c_K\}$ – конечное *множество компонентов* решения, а $X = \{x = \langle c_{i_1}, c_{i_2}, \ldots, c_{i_h}, \ldots \rangle, |x| \leqslant n < +\infty\}$ – *множество состояний задачи*.

Множество допустимых состояний $\tilde{X} \subset X$ определяется как множество состояний, из которых может быть составлено решение-кандидат, удовлетворяющее ограничениям Ω. *Множество допустимых решений* определяется как $\tilde{S} = \tilde{X} \cap S$, а *множество оптимальных допустимых решений* S^* является подмножеством множества $\tilde{S}$.

Далее вводится так называемый *граф конструирования* (*construction graph*) $G_C = (C, L)$, вершинами которого являются компоненты задачи C. Граф G_C полон – каждая пара вершин из C соединена ребром из L. Таким образом, исходная задача сводится к поиску в графе G_C такого пути, который

обладает максимальным значением функции $\hat{f}$, удовлетворяет ограничениям Ω и соответствует допустимому решению.

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 случае статической задачи), 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 решение или не выполнится какой-либо критерий останова. Примером такого критерия 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 правила.

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. Конкретные реализации трех описанных этапов могут различаться

для различных муравьиных алгоритмов. Примерами реализаций муравьиных алгоритмов для задачи коммивояжера являются *Ant Colony System* [38], *MAX MIN Ant System* [39], *Rank-based Ant System* [40] и *Elitist Ant System* [76].

1.2.2.3. Современные приложения муравьиных алгоритмов

На настоящий момент известно множество успешных примеров применения муравьиных алгоритмов для решения NP-трудных задач. В [77, 19] предлагаются модификации муравьиных алгоритмов для решения траспортных задач, например, задачи выбора маршрута транспорта (*vehicle routing problem*). В [78] разработан муравьиный алгоритм для минимизации энергопотребления при мультивещании в беспроводных динамических сетях. В [79] предложена модификация муравьиного алгоритма для решения задачи упорядочивания автомобилей на линии сборки. В работе [80] предлагается модифицированная модель муравьиного алгоритма, применимая для решения оптимизационных задач на очень больших графах. Преимущество этого подхода демонстрируется на примере решения задачи проверки моделей. В [20] предложен гибридный эволюционно-муравьиный алгоритм для решения задачи разбиения.

Некоторые компании, например, *Eurobios* [119] и *AntOptima* [120], предоставляют программные решения задач оптимизации, логистики, составления расписаний и анализа данных на основе муравьиных алгоритмов.

В заключение отметим, что, как показал обзор, муравьиные алгоритмы ранее не применялись для генерации конечных автоматов.

1.2.2.4. Генерация конечных автоматов с помощью классических муравьиных алгоритмов

Из проведенного обзора можно сделать вывод о том, что муравьиные алгоритмы могут быть весьма эффективны применительно к задачам оптимизации на графах. С формальной точки зрения, подход к решению комбинаторных задач с помощью муравьиных алгоритмов, описанный в разделе 1.2.2.2,

позволяет применить муравьиный алгоритм к любой комбинаторной задаче. Однако эффективность любого муравьиного алгоритма будет сильно зависеть от того, как определена *эвристическая информация* на ребрах графа конструирования.

В качестве примера приведем алгоритм решения задачи генерации конечных автоматов, рассмотренный диссертантом в [136]. Алгоритм позволяет генерировать конечные автоматы специального вида, каждый переход которых помечен одним входным событием и единственным выходным воздействием. Такой автомат является, например, решением известной задачи об «Умном муравье» [7, 27].

Предполагается, что задано число состояний искомого автомата N, множество возможных входных событий E и множество возможных выходных воздействий Z. Также на множестве всех автоматов с указанными параметрами (N, E, Z) задана функция приспособленности f. Необходимо найти конечный автомат, значение функции приспособленности которого не меньше наперед заданного числа $f_{\min}$.

Классическое сведение этой задачи к виду, приемлемому для применения классического муравьиного алгоритма (например, *Ant System* [36]), выглядит следующим образом. Множество решений-кандидатов $\hat{S}$ представляет собой множество всех конечных автоматов с параметрами (N, E, Z). Здесь целевая функция $\hat{f}$ – функция приспособленности автомата f. Множество компонентов C – множество всех возможных переходов автоматов с параметрами (N, E, Z):

$$C = \{t = \langle i, j, e, z\rangle \,|i, j \in Y, e \in E, z \in Z\}\,,$$

где Y – множество состояний автомата, $i \in Y$ – состояние, из которого выполняется переход, $j \in Y$ – состояние, в которое выполняется переход, $e \in E$ – событие, по которому выполняется переход, а $z \in Z$ – выходное воздействие, записанное на переходе.

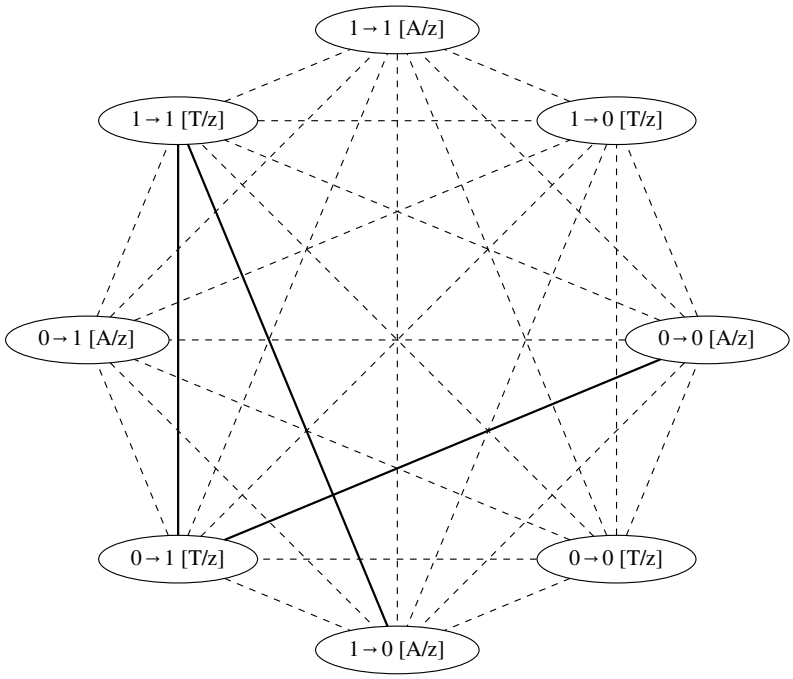

Рисунок 3 – Пример графа конструирования классического муравьиного алгоритма для генерации конечных автоматов

Пример графа конструирования представлен на рисунке 3. В приведенном примере граф был построен для автоматов с двумя состояниями, множество событий E состоит и двух элементов A и T, а множество выходных воздействий Z состоит из единственного элемента z.

Ограничения Ω описывают то, что конечный автомат должен быть *детерминированным* – из каждого состояния по каждому входному событию в автомате должен существовать ровно один переход. Для реализации ограничений k-й муравей использует свою внутреннюю память для хранения множества посещенных им компонентов L_t^k.

Пусть k-й муравей располагается в вершине $u \in C$. Обозначим множество вершин, инцидентных u, как N_u. Перед выбором следующей вершины муравей сканирует множества N_u и L_t^k, формируя множество компонентов $\hat{N}_u$. Это множество содержит только такие компоненты $t \in N_u$, что для любого состояния $y \in Y$ и входного события $e \in E$, множество посещенных компонентов L_t^k не содержит переходов с тем же стартовым состоянием и входным событием.

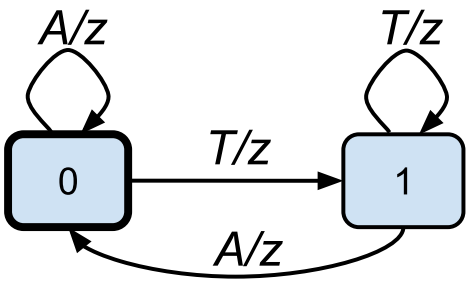

Рисунок 4 – Пример автомата, сгенерированного классическим муравьиным алгоритмом

Следующая вершина v выбирается k-м муравьем из множества $\hat{N}_u$ с вероятностью, вычисляемой по формуле:

$$p_v = \frac{\tau_{uv}^{\alpha}}{\sum\limits_{w \in \hat{N}_u} \tau_{uw}^{\alpha}},$$

где $a \in (0, +\infty)$.

Эта формула не учитывает эвристическую информацию, поскольку не было найдено значимого способа ее определения. Сплошные ребра графа рисунке 3 обозначают возможный путь муравья и соответствуют автомату, изображенному на рисунке 4.

Использовался муравьиный алгоритм *Elitist Ant System* [37], в котором лучшее найденное решение s_{best} откладывает феромон наряду с решениями, найденными на текущей итерации. Значения феромона обновляются по формуле:

$$\tau_{uv} = (1 - \rho)\tau_{uv} + \sum_{k=1}^{N_{\text{ants}}} \Delta\tau_{uv}^{k} + w_{\text{elit}} \cdot \Delta\tau_{uv}^{\text{best}},$$

где параметр $w_{\text{elit}} \in [0, 1]$ определяет влияние элитарного решения, $\rho \in [0, 1]$ – скорость испарения феромона,

$$\Delta\tau_{uv}^{k} = \begin{cases} f(s_k), (u, v) \in L_t^k; \\ 0, \text{иначе}; \end{cases}$$

и

$$\Delta\tau_{uv}^{\text{best}} = \begin{cases} f(s^{\text{best}}), (u, v) \in L_t^{\text{best}}; \\ 0, \text{иначе}. \end{cases}$$

Здесь s_k – решение, найденное k-м муравьем на текущей итерации, а L_t^{best} – множество ребер графа конструирования, посещенных лучшим муравьем.

Как показано в работе [136], такой алгоритм работает очень долго даже для такой простой задачи, как задача об «Умном муравье» [7, 27].

1.3. Поисковая инженерия программного обеспечения

Особый интерес представляет новая область исследований, в которой предлагается применять алгоритмы поисковой оптимизации для решения задач инженерии программного обеспечения. Эта область получила название *поисковая инженерия программного обеспечения* [28, 81] (*search-based software engineering*). Алгоритмы поисковой оптимизации, в основном, генетические алгоритмы и генетическое программирование, применяются здесь, например, для:

— тестирования программ [29];
— решения задачи следующего выпуска (*next release problem*) [30];
— поддержки программ [31];
— управления проектами [32];
— оптимизации кода [82, 83].

Например, в работе [82] методы поисковой инжерении программного обеспечения были применены для оптимизации кода *bowtie2* [84] – популярного в области биоинформатики программного средства для выравнивания последовательностей (*sequence alignment*). Код данного проекта составляет 50000 строк на языке *C++*. Применив генетическое программирование, авторам удалось достичь увеличения скорости работы программы в 70 раз, при этом несколько улучшив качество ее работы. Авторам [83] удалось с помощью генетического программирования добиться стократного увеличения эффективности ключевой функции, использующейся в другом программном средстве для выравнивания последовательностей *BarraCUDA* [85].

Значимость этого направления в науке также отражает динамика числа публикаций: до 1992 года было опубликовано менее десяти работ, а в 2014 году вышло более 200 научных статей. На сентябрь 2015 года по данной тематике насчитывается всего лишь 1389 публикаций [114].

1.4. Методы генерации конечных автоматов

В настоящем разделе описываются основные существующие методы генерации различных типов конечных автоматов, основанные на эволюционных алгоритмах, генетических алгоритмах и генетическом программировании, алгоритмах решения задач выполнимости булевой формулы и удовлетворения ограничениям, AE-парадигме. Отметим, что большинство существующих методов не подходит для генерации автоматов со сложными условиями на переходах [115].

1.4.1. Методы генерации конечных автоматов без учета темпоральных формул

1.4.1.1. Методы на основе эволюционных и генетических алгоритмов

Одной из первых работ по генерации конечных автоматов принято считать статью [86], в которой было предложено моделировать эволюционные процессы для создания предсказателей в форме конечных автоматов. Под конечным автоматом подразумевалась программа, в каждый момент находящаяся в некотором состоянии. Прочитав входной символ, программа переходит в новое состояние и генерирует выходную строку. Задачей автомата в этой работе было предсказание символов строки (например, 320121300), а функция приспособленности оценивала, насколько хорошо автомат предсказывает данные. Лучшие автоматы получали возможность порождать потомков с использованием операций мутации числа состояний, функций переходов и выходов автомата, а также стартового состояния. Описанная процедура стала основой методов, нашедших свое применение в решении задач предска-

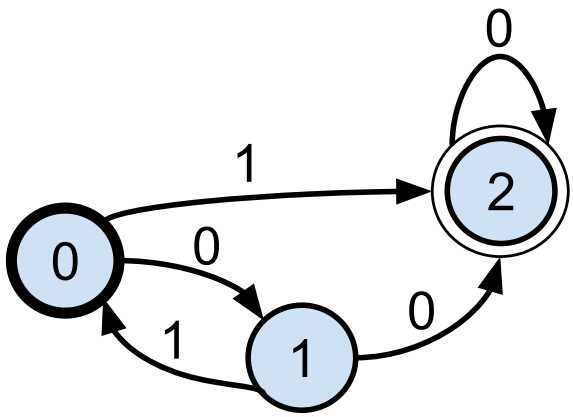

Рисунок 5 – Пример детерминированного конечного автомата

зания данных, идентификации систем, управления, распознавания образов. Книга этого автора [87] стала первой книгой по эволюционным вычислениям. Отметим, что в 1969 эта книга вышла на русском языке [21].

Известны применения эволюционных алгоритмов для генерации таких типов автоматов, как детерминированные конечные автоматы (ДКА) и автоматы-преобразователи (*finite-state transducers*). Переходы ДКА помечены только входным символом, а состояния могут быть допускающими или недопускающими. Если обработка входного слова заканчивается в допускающем состоянии, то слово принадлежит языку, заданному ДКА. Пример ДКА приведен на рисунке 5, допускающее состояние отмечено двойной рамкой.

Так, в работе [42] (1 + 1)-ЭА использовался для построения ДКА по заданному множеству помеченных строк. Для представления автоматов использовались полные таблицы переходов. Особенностью этой работы является то, что пометки состояний автомата не хранились и не строились эволюционным алгоритмом. Вместо этого был предложен алгоритм, позволяющий оптимальным образом расставить эти пометки исходя из обучающих примеров. Эволюционный алгоритм сравнивался с лучшим на тот момент алгоритмом слияния состояний [43]. Было показано, что ЭА эффективнее алгоритма слияния состояний когда число состояний автомата невелико.

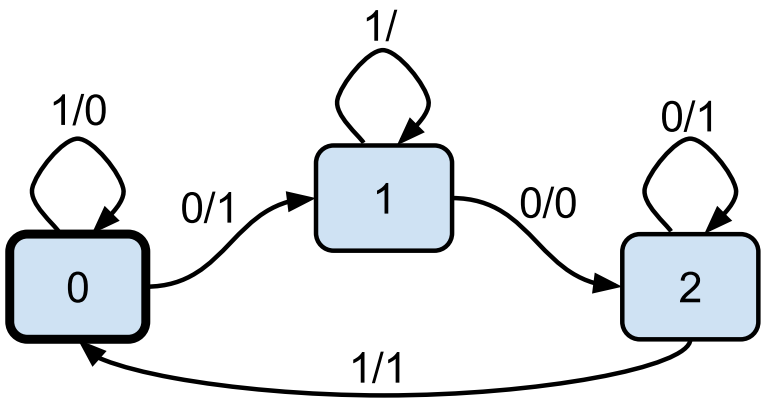

Рисунок 6 – Пример конечного автомата-преобразователя

Автор [88] использует алгоритм расстановки пометок, предложенный в [42], однако в качестве метода поисковой оптимизации предлагает новый гибридный алгоритм, совмещающий постепенное обучение (*incremental learning*) и эволюционный алгоритм. Предлагается перед началом работы отсортировать строки обучающего набора по возрастанию длины. Обучающий набор разделяется на заданное число блоков. Основная идея состоит в том, что на начальных итерациях автоматы обучаются на первом блоке, а остальные блоки добавляются в активный обучающий набор по мере достижения идеальной приспособленности на уже включенных блоках. Этот подход позволил получать чуть менее качественные автоматы, чем в работе [42], но в несколько раз быстрее.

В [41] эволюционный алгоритм использовался для построения автоматов-преобразователей по тестовым примерам. Пример автомата-преобразователя приведен на рисунке 6. Его переходы помечены входным символом (перед косой чертой) и выходным символом (после косой черты). Тестовые примеры представляли собой набор пар строк, например `101010` $\rightarrow$ `01011`. Искомый автомат должен был удовлетворять всем таким примерам – например, при получении на вход строки `101010` должен выдавать на выходе строку `01011`. Исследовалось применение функций приспособленности на основе полного совпадений строк, расстояния Хэмминга и расстояния

Левенштейна [89]. Было показано, что функция приспособленности на основе расстояния Левенштейна обеспечивает наибольшую эффективность эволюционного алгоритма.

В [6] генетические алгоритмы применялись для построения конечных автоматов, решающих задачу о флибах. Эта задача состоит в построении автомата-преобразователя, предсказывающего состояние некой среды, заданное битовой маской определенной длины. Идеальный автомат, получив на вход любой бит этой маски, должен выдать на выход следующий бит. Позже, в [22] был предложен алгоритм, позволяющий построить флиб с идеальной точностью предсказания, содержащий минимально возможное число состояний.

Генетические алгоритмы также применялись при построении автоматов-преобразователей для задачи об «Умном муравье» [7, 27]. В [7] был разработан специальный оператор скрещивания, учитывающий поведение автомата при выполнении первых сорока ходов. В англоязычной литературе также встречаются примеры использования генетических алгоритмов для решения этой задачи [48, 90].

Авторы [49] применяют генетический алгоритм для построения автоматов в задаче о завоевании ресурсов (*Competition for Resources*). Функция приспособленности в этой задаче основана на моделировании поведения агента в некоторой игре. Рассматривалась задача построения конечно-автоматной системы управления агентом для игры против псевдослучайной стратегии. Одним из интересных результатов этого исследования является то, что конечно-автоматной модели в чистом виде оказалось недостаточно для успешной игры агента. Для повышения эффективности было предложено добавить к модели несколько ячеек памяти, которые использовались для предотвращения нежелательного циклического поведения агента.

В работах [9, 91] генетические алгоритмы применялись для решения значительно более сложной задачи построения управляющих конечных ав-

томатов для управления беспилотным летательным аппаратом. В работе [9] используется функция приспособленности, основанная на моделировании, в связи с чем построение автомата требует значительных вычислительных затрат. В работе [91] предлагается строить автоматы для автопилота по тестовым примерам поведения, которые записываются экспертом. Этот подход в значительной мере ускорил построение автоматов.

Также отметим, что когда число входных переменных автомата велико, становится особо важным способ представления автоматов в виде особей эволюционных алгоритмов. Простейшим способом является метод полных таблиц переходов, в котором для каждой комбинации значений входных переменных хранится переход. В случае большого числа входных переменных использование полных таблиц приводит к уменьшению эффективности эволюционных алгоритмов. Поэтому в [9] был разработан метод сокращенных таблиц переходов, в котором хранятся переходы только для выбранного множества значимых входных переменных. Альтернативными способами представления автоматов являются деревья решений [23] и линейные бинарные графы [24].

1.4.1.2. Методы на основе алгоритмов слияния состояний

Рассмотрим задачу генерации детерминированного конечного автомата по множеству двоичных слов, у каждого из которых есть пометка – входит оно в язык («+») или нет («−»). Задача состоит в построении минимального по числу состояний детерминированного конечного автомата, допускающего все слова из заданного множества с пометкой «+» и не допускающего все слова с пометкой «−». В работе [92] было доказано, что эта задача входит в класс NP-трудных.

Все алгоритмы слияния состояний, например [43, 93], сначала строят префиксное дерево, в которое входят все слова заданного множества слов. Очевидно, что если опустить требование минимальности, то такое префикс-

ное дерево и является решением задачи. Далее будем называть вершины префиксного дерева состояниями. Алгоритмы слияния состояний эвристически вычисляют некий количественный показатель того, насколько «выгодно» слить две вершины дерева в одну.

Одним из основных недостатков этой группы алгоритмов является то, что построенные автоматы ими автоматы зачастую далеко не минимальны. Также, как было показано в работе [94], построенные с помощью слияния состояний автоматы имеют гораздо худшую обобщающую способность по сравнению с автоматами, построенными эволюционным алгоритмом.

1.4.1.3. Точные методы на основе сведения к задачам *SAT* и *CSP*

Описанные выше методы и алгоритмы не являются точными – не дается никаких гарантий того, что найденное решение будет оптимальным. Однако для некоторых задач генерации конечных автоматов по примерам поведения разработаны точные методы, которые позволяют находить оптимальные решения этих задач или доказывать, что оптимального решения не существует. Эти методы основаны на сведении задач построения автоматов к задаче выполнимости булевой формулы (SAT) и задаче удовлетворения ограничениям (CSP).

Так, в [44] предложено сведение задачи генерации детерминированных конечных автоматов к задаче SAT. В работе [45], выполненной на кафедре «Компьютерные технологии», это сведение было дополнено предикатами нарушения симметрии, что привело к существенному увеличению эффективности метода. Также было предложено точное решение задачи генерации ДКА по зашумленным входным данным, которая была поставлена на соревновании, проведенном в 2004 году на конференции *Genetic and Evolutionary Computation Conference* [121].

В работе [46], также выполненной на кафедре «Компьютерные технологии», был предложен алгоритм построения управляющих конечных автома-

тов по сценариям работы, основанный на сведении этой задачи к задаче *SAT*. Позже, в статье [132] был предложен алгоритм, основанный на сведении к задаче удовлетворения ограничениям (*CSP*). В работе [47] был предложен эффективный метод построения детерминированных конечных автоматов, комбинирующий алгоритм слияния состояний и алгоритм на основе сведения к задаче выполнимости булевой формулы.

В основе этих методов лежит построение по сценариям префиксного дерева сценариев и дальнейшая раскраска этого дерева в заданное число цветов, которое соответствует числу состояний автомата. Для определения того, в какие цвета раскрасить вершины дерева, применяется сведение к задачам *SAT* и *CSP*. Для решения экземпляров этой задачи применяются программные средства решения задач *SAT* и *CSP*, например, *lingeling* [122] и *Choco* [123].

Как показывается в указанных статьях, скорость работы этих методов на несколько порядков превышает скорость работы методов, основанных на эвристическом слиянии состояний. Однако стоит отметить, что эти методы являются узкоспециализированными и применимы лишь для построения конечных автоматов по жестко заданным входным данным. Также не известны работы, в которых подобные методы применялись бы для построения конечных автоматов с учетом заданных темпоральных свойств.

1.4.2. Методы генерации конечных автоматов с учетом темпоральных формул

1.4.2.1. Методы на основе генетических и эволюционных алгоритмов

Вероятно, одним из первых примеров совместного применения генетического программирования и метода проверки моделей для генерации конечных автоматов является [50]. В этой работе отмечается, что при применении генетического программирования функция приспособленности обычно основана на тестировании – проверке корректности работы решения-кандидата

для некоторого конечного набора входных данных (назовем этот набор *обучающим*). Такой подход не может гарантировать правильного поведения системы на данных, не вошедших в обучающий набор. Поэтому предлагается в качестве функции приспособленности использовать результат верификации, а именно долю заданных темпоральных свойств, которым удовлетворяет решение-кандидат. Темпоральные свойства задаются на языке *CTL* (*Computation Tree Logic*) [95], а для проверки моделей используется программное средство *SMV* [124].

В статье программы генерируются в виде взаимодействующих конечных автоматов [96]. Для генерации автоматов автор применяет $(1 + \lambda)$-ЭА. Изначально автомат состоит из одного состояния. Предложен оператор мутации, который с высокой вероятностью добавляет состояния и/или переходы. Скрещивание не используется.

Представлены результаты экспериментов по генерации автоматов управления кофейным аппаратом. В первом эксперименте удалось построить автоматы из трех-пяти состояний, удовлетворяющие восьми *CTL*-формулам. Для более сложного примера 17 формул автомат построить не удалось.

У этой работы есть несколько недостатков. Во-первых, оператор мутации никак не использует результаты верификации. Во-вторых, используемая функция приспособленности имеет слишком мало возможных значений. Например, если требуется построить автомат, удовлетворяющий всего одной *CTL*-формуле, то предложенный метод, скорее всего, не найдет решения, так как в этом случае ФП может иметь всего два значения – ноль или единица. Такая ФП не дает поисковому алгоритму никакой информации, поэтому решение может быть найдено лишь случайно.

Авторы [10—12, 52] устраняют некоторые недостатки предыдущей работы. Указанные работы посвящены генерации *управляющих конечных автоматов* по тестовым примерам (или сценариям работы) и темпоральным формулам на языке *LTL*. В качестве алгоритма поисковой оптимизации ис-

пользуется генетический алгоритм. Предлагаются специализированные операторы мутации и скрещивания, учитывающие поведение автоматов на тестовых примерах и результаты верификации темпоральных формул. Также предложен алгоритм верификации автоматных программ, в котором преобразование управляющего конечного автомата в автомат Бюхи осуществляется не явно, а «на лету» в процессе верификации. Опишем предложенный в указанных работах подход более подробно.

Решение представляется в виде *каркаса* конечного автомата – автомата, в котором вместо конкретных выходных воздействий на переходах указано лишь число выходных воздействий. Выходные воздействия для каждого перехода вычисляются оптимальным образом перед каждым вычислением ФП. Для этого используется предложенный авторами *алгоритм расстановки пометок на переходах*.

Начальная популяция генетического алгоритма заполняется случайно сгенерированными автоматами. При формировании следующего поколения используется элитизм – несколько лучших особей переходят в следующее поколение без изменений. Далее, пока новое поколение не заполнено, из старого поколения случайным образом выбираются две особи, к которым применяются операции мутации и скрещивания. Результирующие измененные особи добавляются в новое поколение.

Предложенный оператор мутации с определенной вероятностью изменяет каждый переход автомата. Учет результатов верификации производится следующим образом. Пусть при вычислении значения ФП автомата верификатор обнаружил контрпример к одной из *LTL*-формул. Контрпример в терминах автомата Бюхи автоматически преобразуется в термины исходного автомата – список переходов. Изменение переходов, входящих в контрпример, происходит с большей вероятностью, чем изменение остальных переходов.

Также предложен оператор скрещивания конечных автоматов, учитывающий их поведение на тестах/сценариях. Идея заключается в следующем.

Для каждой из родительских особей выбирается 10 % тестов, которые наилучшим образом описываются этой особью. Переходы, которые выполняются при обработке этих тестов, помечаются. Помеченные переходы вместе переходят в дочерние особи.

В указанных работах предлагается усовершенствованная по сравнению с [50] функция приспособленности, учитывающая результаты верификации. Верификатор, разработанный в [12], позволяет выделять переходы автомата, которые точно не входят в контрпример. Указанный верификатор выполняет двойной обход в глубину. Когда при первом обходе состояние автомата покидается, ведущие из него переходы помечаются как *проверенные*, так как они точно не лежат на пути, опровергающем *LTL*-формулу [12]. В качестве вклада результатов верификации в значение ФП используется отношение числа проверенных переходов к числу переходов, достижимых из начального состояния.

Если в течении некоторого числа поколений не происходит увеличения значения ФП, то применяются «малая» и «большая» мутации поколения. При «малой» мутации все особи, кроме 10% лучших, подвергаются мутации. При «большой» мутации поколения к каждой особи либо применяется оператор мутации, либо она заменяется на новую случайно сгенерированную особь.

Отметим, что метод требует значительного времени даже при генерации автоматов только по сценариям. Эксперименты по генерации конечных автоматов на основе сценариев работы и темпоральных формул проводились в [12] только для двух простых примеров, в которых решения содержали всего три и пять состояний.

1.4.3. Методы на основе *AE*-парадигмы

Теоретические основы этой группы методов были заложены еще А. Черчем, когда в [97] он сформулировал так называемую задачу синтеза. В 1989 году был предложен подход к синтезу реактивных систем, удовлетворяющих за-

данным *LTL*-формулам [54]. Суть этого подхода, названного *AE*-парадигмой, заключается в том, что программа с входным сигналом x и выходным сигналом y, удовлетворяющая темпоральной формуле $\varphi(x, y)$, конструируется как побочный продукт доказательства теоремы $(\forall x)(\exists y)\varphi(x, y)$. Название парадигмы происходит от английских терминов для кванторов $\forall$ (*for All*) и $\exists$ (*Exists*).

Позже, в диссертации [53] было доказано, что данная задача полна в сложностом классе 2EXPTIME. В теории задача синтеза по *LTL*-формулам была решена, однако ввиду полученной оценки сложности практических реализаций этого подхода долгое время не существовало.

После многочисленных оптимизаций различных этапов подхода из [54] появились применимые на практике методы генерации конечных автоматов по *LTL*-формулам. Эти методы являются *точными* в том смысле, что позволяют найти решение, если оно существует, или доказать, что решения не существует. Однако как показал обзор, ни один из этих методов не позволяет искать решение в виде автомата с фиксированным числом состояний. Далее в настоящем разделе рассматриваются программные средства, реализующие методы генерации автоматных моделей по *LTL*-формулам.

1.4.3.1. Программное средство *lily*

Данное программное средство является реализацией алгоритма, предложенного в статье [55], котораяй является непосредственным продолжением работы [54]: предложены оптимизации всех этапов алгоритма.

Программное средство *lily* [55] принимает на вход набор *LTL*-формул. Решение строится в виде автомата Мура, представленного в виде модуля на языке *Verilog*, поддерживаются входные и выходные логические переменные. Строятся полные автоматы – такие, в которых в каждом состоянии в явном виде записан переход по каждой комбинации значений входных переменных.

Изменение значений выходных переменных происходит при переходе в новое состояние.

1.4.3.2. Программные средства *unbeast* и *G4LTL_ST*

В [98] был предложен подход, названный *bounded synthesis*, в котором для уменьшения размера пространства поиска в задаче ситнеза вводится ограничение на число состояний искомой модели. Поиск начинается при малом значении ограничения, которое инкрементально увеличивается пока не будет найдено удовлетворяющее заданным формулам решение. Этот подход был реализован с использованием программного средства для решения задачи выполнимости формул в теориях.

В работе [56] предлагается развитие подхода *bounded synthesis* [98], в котором по *LTL*-формуле строится так называемый *universal co-Büchi automaton*, который затем используется для построения некоторой игры между автоматом и средой. Цель автомата в этой игре заключается в том, чтобы для каждого входного воздействия, выданного средой, сгенерировать правильное выходное воздействие. Цель среды – подобрать входное воздействие, для которого автомат ошибется. Выигрышная стратегия в такой игре соответствует автоматной модели, удовлетворяющей заданным формулам.

Решение строится в виде полного автомата Мили с входными и выходными логическими переменными. Программное средство *unbeast* не позволяет вывести результирующий автомат в явном виде. Вместо этого представляется интерактивный интерфейс, позволяющий тестировать сгенерированное решение. Также предоставляется возможность доказательства того, что решения не существует.

Другая реализация подхода на основе игр из [98] представлена в работе [57], посвященной генерации модулей стандарта [112]. Программное средство *G4LTL_ST* генерирует решение в виде полного автомата Мили, который записывается на языке *Structured Text*. Предоставляется только графический

интерфейс пользователя, позволяющий в том числе задавать параметр метода, называемый *unroll bound*. Для некоторых формул слишком маленькое значение этого параметра может не позволить ни найти решение, ни доказать, что его не существует. Поэтому приходится вручную инкрементально увеличивать его значение.

1.4.4. Метод генерации автоматных моделей программ с инвариантами

В [99] предложен метод генерации автоматных моделей программ по трассировкам, в котором наряду с моделью генерируются также простые *темпоральные инварианты*. Метод состоит из нескольких этапов: выделение трассировок из логов работы программы, построение инвариантов, генерация автоматной модели.

Для выделения трассировок из логов используются регулярные выражения, предоставляемые пользователем. Каждый элемент трассировки содержит временную метку и так называемый *тип события*. В генерируемых моделях типы событий соответствуют состояниям.

Из имеющихся трассировок путем простого подсчета извлекаются темпоральные инварианты трех типов:

— $a \rightarrow b$: после события типа a в той же трассировке всегда встречается событие типа b;

— $a \not\rightarrow b$: после события типа a в трассировке никогда не встречается событие типа b;

— $a \leftarrow b$: перед событием типа b всегда в трассировке всегда встречается событие типа a.

Наконец, генерируется автоматная модель, удовлетворяющая всем темпоральным инвариантам. Сначала строится граф трассировок – множество линейных графов, каждый из которых соответствует одной трассировке. Начальное решение получается путем объединения всех вершин графа

трассировок, соответствующих одному типу события. Так как полученный автомат может не удовлетворять всем инвариантам, запускается итеративная процедура: вершины, в которых нарушаются инварианты, разделяются; те вершины, которые можно объединить без нарушения инвариантов, объединяются.

В этом подходе не поддерживаются выходные воздействия, а также поддерживаются только очень простые темпоральные свойства – инварианты. Также можно заметить, что инварианты, сгенерированные по трассировкам, могут быть неверными в силу того, что длина трассировок конечна. Поэтому автомат, сгенерированный с учетом этих инвариантов, также может быть неверным.

1.5. О свойствах пространства поиска в одной задаче генерации конечных автоматов

Дополнительной сложностью при применении методов поисковой оптимизации для генерации конечных автоматов является «плохой» *ландшафт функции приспособленности*. В простейшем случае двухпараметрического пространства поиска ландшафт ФП можно представить как поверхность, задающую график ФП, где абсцисса и ордината обозначают значения параметров особи, а аппликата соответствует значению ФП этой особи. В работе [143] автором настоящей диссертации были изучены свойства ландшафта ФП в задаче об «Умном муравье».

В этой задаче требуется построить автомат, который оптимальным образом управляет некоторым агентом в игре. Игра происходит на поле, которое представляет собой тор размера 32×32 клетки. На поле вдоль некоторой строго заданной ломаной расположены «яблоки». Поле изображено на рисунке 7.

Агентом в игре является «муравей», который в начале игры расположен в левой верхней клетке поля и «смотрит» на восток. Муравей видит на одну

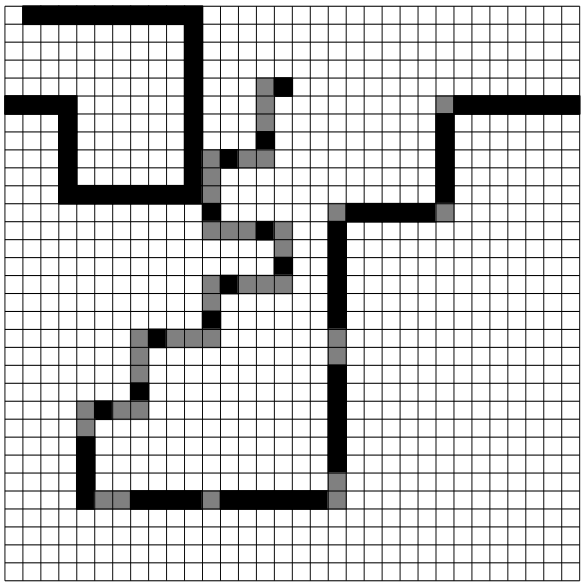

Рисунок 7 – Поле в задаче об «Умном муравье». Черные клетки обозначают яблоки, белые клетки пусты, а серые клетки изображают ломаную

клетку вперед – он может определить, есть в следующей клетке еда (событие F) или нет (событие $\neg F$). На каждом шаге муравей может повернуть налево (выходное воздействие L), повернуть направо (выходное воздействие R) или перейти на одну клетку вперед (выходное воздействие M). Если в клетке, в которую перешел муравей, находилось яблоко, муравей его съедает. Целью игры является построение конечного автомата, управляющего муравьем так, чтобы он мог съесть всю еду не более, чем за 200 шагов. Функция приспособленности в этой задаче учитывает как число съеденных яблок n_{food}, так и номер шага s_{last}, на котором было съедено последнее яблоко:

$$f = n_{\text{food}} + \frac{200 - s_{\text{last}} - 1}{200}.$$

Был рассмотрен оптимальный конечный автомат из семи состояний, сгенерированный с помощью генетического алгоритма [7] и решающий задачу за 189 шагов. Значение ФП этого автомата равно 89,05. Были рассмотрены все соседи этого автомата – все автоматы, полученные из него применением одной операции мутации. Использовалось два оператора мутации: изменение состояния, в которое ведет переход, и изменение действия на переходе. Гисто-

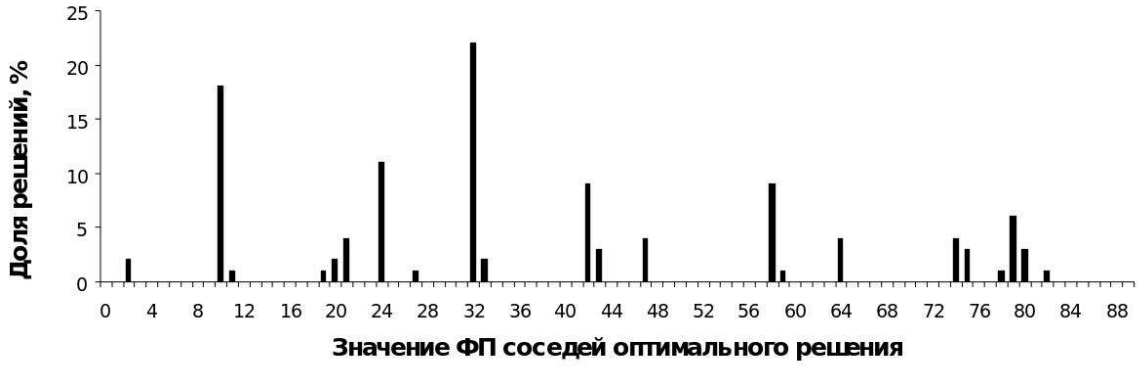

Рисунок 8 – Гистограмма значений ФП соседей оптимального решения задачи об «Умном муравье»

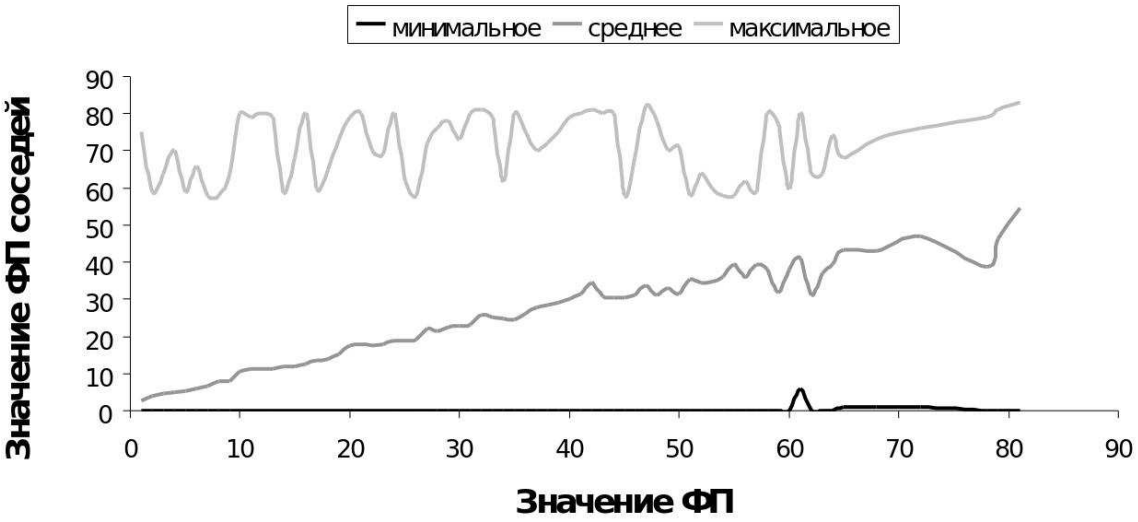

Рисунок 9 – Минимальное, среднее и максимальное значение ФП соседей в зависимости от значения ФП автоматов из семи состояний

грамма значений ФП этих автоматов изображена на рисунке 8. Из гистограммы видно, что ни один из соседей оптимального решения не имеет значения ФП выше, чем 82. При этом автоматы, имеющие значения ФП из отрезка $[83, 88]$, существуют. Фактически это означает, что эволюционные алгоритмы, старающиеся максимизировать значения ФП своих особей, могут найти оптимальное решение лишь случайно.

Во втором эксперименте было рассмотрено 10^4 случайных автоматов из семи состояний. Для каждого из них были построены все соседние автоматы

и записано значение ФП исходного автомата, а также минимальное, среднее и максимальное значение ФП соседей. Графики зависимости среднего, минимального и максимального значения ФП соседей от значения ФП исходного автомата приведены на рисунке 9. Эти графики показывают, что с ростом значения ФП автомата значения ФП его соседей в среднем также возрастают. Однако вместе с тем видно, что разброс значений ФП соседей для всех значений ФП автоматов велик, что негативно влияет на производительность эволюционных алгоритмов при решении данной задачи.

Эти результаты указывают на то, что поисковый алгоритм генерации конечных автоматов должен производить активный поиск в окрестности субоптимальных решений. Одним из вариантов реализации такого подхода является использование некой долговременной общей памяти по аналогии с муравьиными алгоритмами.

1.6. Постановка задачи генерации управляющих конечных автоматов по сценариям работы и темпоральным формулам

Одним из типов исходных данных в задаче генерации автоматов по спецификации являются примеры поведения, которое пользователь хочет наблюдать у искомого автомата. В качестве таких примеров поведения могут выступать тестовые примеры, сценарии работы, негативные сценарии [11, 12].

Тестовый пример или *тест* в [11, 12] определяется как две последовательности элементов – входная и выходная. Элемент входной последовательности состоит из входного события и булевой формулы от входных переменных, а элемент выходной последовательности – выходное воздействие. *Сценарий работы* – это последовательность элементов, каждый из которых состоит из входного события, булевой формулы от входных переменных и последовательности выходных воздействий. В [12] было показано, что использование сценариев работы эффективнее использования тестов.

Второй тип исходных данных – темпоральные формулы, которым должен удовлетворять искомый автомат. Темпоральные формулы в данной работе задаются на языке логики линейного времени *LTL*.

Формально, *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, $\varphi_i^j \in 2^X$ – булева формула от входных переменных и $O_i^j \in Z^*$ – последовательность выходных воздействий. Говорят, 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 рисунке 2, 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 $\langle e_2, 1, (z_2)\rangle\langle e_2, \neg x, (z_1)\rangle$.

Язык *LTL* включает в себя специфичные для конкретной решаемой задачи пропозициональные переменные, логические операторы $(\wedge, \vee, \neg)$, а также следующие темпоральные операторы.

— ***G****lobally*. $G(f)$ – f выполняется всегда.

— *ne****X****t*. $X(t)$ – f выполняется в следующем состоянии.

— ***F**uture*. $F(f)$ – f выполняется в каком-либо последующем состоянии.

— ***U**ntil*. $U(f, g)$ – f выполняется по крайней мере до тех пор, пока не выполнится g, g выполняется в текущем или в каком-либо последующем состоянии.

— ***R**elease*. $R(f, g)$ – g выполняется до того момента, пока не выполнится f. Если f никогда не выполнится, то g должно выполняться всегда.

Формулы, рассматриваемые в данной работе, 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 рисунке 2, 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. Автомат не удовлетворяет формуле $\boldsymbol{G}\,(\neg\text{wasEvent}(e_2) \vee \boldsymbol{X}\,(\text{wasEvent}(e_1)))$, 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.

Решается задача генерации управляющего конечного автомата с заданным числом состояний, удовлетворяющего заданному набору сценариев работы S и набору *LTL*-формул. Предлагаемый алгоритм осуществляет направленный перебор решений-кандидатов. Для оценки того, насколько хорошо решение-кандидат удовлетворяет заданному набору сценариев и *LTL*-формул, используется *функция приспособленности*.

1.7. Задачи, решаемые в диссертационной работе

Из проведенного обзора следует, что большинство существующих методов генерации конечных автоматов не учитывает темпоральные формулы.

Методы на основе генетических алгоритмов и AE-парадигмы, в которых учитываются темпоральные формулы, работают долго даже для простых примеров автоматов с небольшим числом состояний. Ландшафт функции приспособленности в задачах генерации автоматов может быть таким, что метаэвристический алгоритм сможет найти оптимальное решение лишь случайно. Поэтому целесообразно применять для генерации автоматов муравьиные алгоритмы – в них используется долговременная общая память об исследованной области пространства поиска, что может помочь усилить поиск в окрестности субоптимальных решений. Отметим, что муравьиные алгоритмы ранее не применялись для генерации конечных автоматов. Кроме того, показано, что использование *классических* муравьиных алгоритмов для генерации конечных автоматов неэффективно.

На основании проведенного обзора сформулируем **цель** диссертационной работы — развитие существующих подходов к генерации конечных автоматов по сценариям работы и темпоральным формулам за счет применения муравьиных алгоритмов.

Задачи, решаемые в диссертационной работе:

1. Разработать метод генерации конечных автоматов по сценариям работы и темпоральным формулам на основе муравьиного алгоритма.
2. Разработать методы генерации конечных автоматов по сценариям работы и темпоральным формулам на основе параллельных муравьиных алгоритмов.
3. Разработать инструментальное средство, реализующее предложенные методы.
4. Внедрить результаты работы при генерации автоматной логики для базисных функциональных блоков стандарта *IEC 61499* и в учебный процесс.

Выводы по главе 1

1. Проведен обзор работ по темам «Автоматное программирование», «Метаэвристические алгоритмы», «Методы генерации конечных автоматов».
2. Существующие методы генерации конечных автоматов с учетом темпоральных формул требуют много времени для нахождения решения даже для простых примеров небольших автоматов.
3. Муравьиные алгоритмы ранее не применялись для генерации конечных автоматов.
4. Показано, что генерация конечных автоматов с помощью классических муравьиных алгоритмов неэффективна.
5. Приведена формальная постановка задачи генерации конечных автоматов по сценариям работы и LTL-формулам.
6. Сформулирована цель и задачи диссертационной работы.

ГЛАВА 2. МЕТОД ГЕНЕРАЦИИ КОНЕЧНЫХ АВТОМАТОВ ПО СЦЕНАРИЯМ РАБОТЫ И ТЕМПОРАЛЬНЫМ ФОРМУЛАМ НА ОСНОВЕ МУРАВЬИНОГО АЛГОРИТМА

В данной главе описывается предлагаемый метод *MuACO* (*Mutation-based Ant Colony Optimization algorithm*) генерации конечных автоматов на основе муравьиного алгоритма, использующего предложенный автором граф мутаций. Приведены описание и результаты проведенных численных экспериментов. Часть материалов данной главы заимствовано из магистерской диссертации автора [25].

Предлагаемый алгоритм осуществляет направленный перебор решений-кандидатов. Для оценки того, насколько хорошо решение-кандидат удовлетворяет заданному набору сценариев и *LTL*-формул, используется *функция приспособленности*.

2.1. Функция приспособленности

Функция приспособленности, оценивающая степень соответствия автомата заданному набору сценариев S и *LTL*-формулам была предложена в [12]. Эта ФП основана на редакционном расстоянии между строками [89], верификаторе автоматных программ *AutomataVerificator* [12] и имеет вид:

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

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

Первый компонент ФП F_{sc} оценивает соответствие автомата заданному набору сценариев работы S. При вычислении значения ФП автомата 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 [89] $\mathrm{ED}(O_i, A_i)$. Выражение для F_{sc} имеет вид:

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

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 F_{LTL} оценивает соответствие автомата заданным темпоральным формулам. Верификатор, разработанный в [12], 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 $t^i_{\mathrm{reachable}}$. F_{LTL} вычисляется как среднее значение этой величины по всем *LTL*-формулам:

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

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 третьей компоненты ФП определяется тем, что автоматы, содержащие небольшое число переходов, считаются более предпочтительными, чем автоматы с большим числом переходов.

2.2. Способ представления пространства поиска

Основой предложенного в диссертации метода построения конечных автоматов является представление пространства поиска (множества всех автоматов с заданными параметрами) в виде ориентированного графа G, который

мы будем называть *графом мутаций*. Каждая вершина этого графа ассоциирована с конечным автоматом, а ребра соответствуют мутациям автоматов. Мутацией автомата будем называть некое изменение его структуры – таблиц, задающих функции переходов и выходов. При решении задачи построения управляющих конечных автоматов по сценариям и темпоральным формулам используются следующие операторы мутации, предложенные в [11, 12].

1. **Изменение состояния, в которое ведет переход**. Каждый переход автомата изменяется с определенной вероятностью. Переходы, не входящие ни в один контрпример, полученный от верификатора, изменяются с вероятностью $\frac{1}{\textit{число переходов в автомате}}$. Переходы, входящие в контрпример, изменяются с удвоенной вероятностью. Изменение перехода заключается в изменении состояния y, в которое он ведет, на другое состояние, которое выбирается случайным образом из множества $Y \setminus \{y\}$.
2. **Добавление или удаление переходов**. Наличие некоторых переходов в состоянии автомата может сделать его противоречащим *LTL*-формуле. Поэтому необходимо иметь возможность периодически удалять и, соответственно, добавлять переходы. Оператор мутации сканирует состояния автомата и с определенной вероятностью изменяет набор переходов в выбранном состоянии. Случайным образом решается, добавить или удалить переход. При этом переход добавляется лишь в том случае, когда в этом состоянии нет перехода, помеченного какой-либо встречающейся в сценариях комбинацией входного события и булевой формулы. Тогда в состояние добавляется новый переход, который ведет в случайно выбранное состояние. В случае удаления перехода, из текущего состояния удаляется случайно выбранный переход.

Граф мутаций обладает следующим свойством. Пусть вершина графа u соответствует автомату A_1, а вершина v соответствует автомату A_2. Тогда, если автомат A_2 может быть получен из автомата A_1 путем применения одного

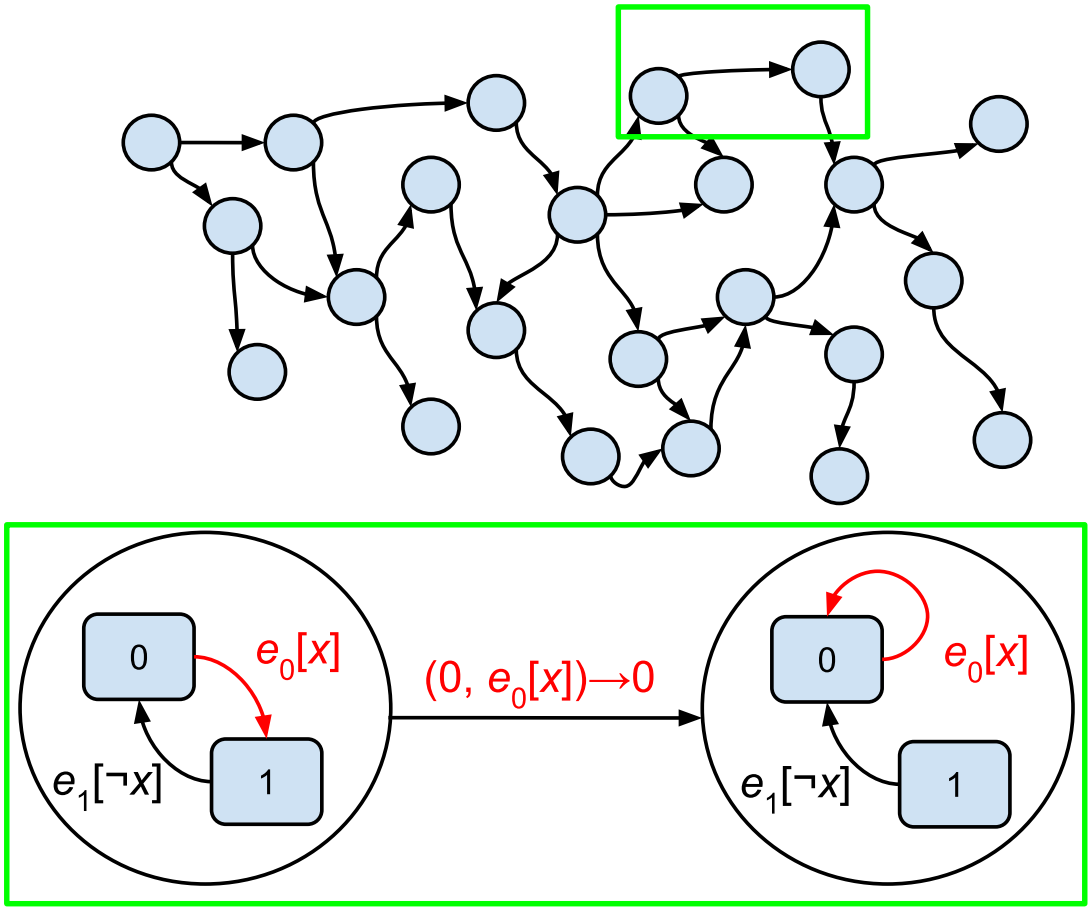

Рисунок 10 – Пример графа мутаций

оператора мутации, то в графе может содержаться ребро uv. Иначе, вершины u и v не могут быть соединены ребром. Также отметим, что для любых двух автоматов A_1 и A_2 и соответствующих им вершин графа мутаций u и v может существовать путь как из u в v, так и из v в u.

Пример графа мутаций приведен на рисунке 10. Вершины соответствуют автоматам, а ребра – мутациям автоматов. Красным цветом отмечен переход автомата, измененный мутацией. Запись на ребре $(0, e_0[x]) \to 0$ означает, что соответствующая мутация изменила состояние, в которое ведет переход из состояния 0 по событию e_0 и формуле x, на состояние 0.

Отметим, что в классических муравьиных алгоритмах используется граф конструирования, ребра которого соответствуют компонентам решений, в то время как полные решения строятся муравьями в процессе обхода графа. В предложенном же методе применяется граф мутаций, вершины которого

соответствуют полным решениям, а муравьи перебирают их, перемещаясь по графу.

2.3. Выбор начального приближения

Перед началом работы алгоритма граф мутаций не содержит ни одной вершины. Первой вершиной графа становится автомат, сгенерированный случайным образом. Другой способ генерации начального приближения рассмотрен в разделе 2.7.

2.4. Эвристическая информация

На каждом ребре графа мутаций uv хранятся значения эвристической информации η_{uv} и феромона τ_{uv}. Эвристическая информация на ребре uv вычисляется по формуле

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

где $\eta_{\min} = const = 10^{-3}$. Значения феромона τ_{uv} изначально равны $\tau_{\min} = const = 10^{-3}$ и изменяются в процессе работы алгоритма.

Общая схема предлагаемого алгоритма напоминает классический муравьиный алгоритм. Пока не выполнено одно из условий останова выполняются следующие действия:

— построение решений муравьями (*ConstructAntSolutions*);

— обновление значений феромона на ребрах графа (*UpdatePheromone*).

Необязательный этап *DaemonActions* не используется. Одну итерацию этого цикла будем называть *итерацией колонии*.

2.5. Построение решений муравьями

Процедуру построения решений муравьями на каждой итерации колонии можно разделить на два этапа. На первом этапе выбираются вершины графа, из которых муравьи начнут поиск решений. В процессе разработки алгоритма [134] рассматривалось несколько способов выбора стартовых вершин.

— Муравьи стартуют из вершин пути лучшего муравья предыдущей итерации.
— Муравьи стартуют из вершин, выбранных методом рулетки [100] из всех вершин графа, вес вершины равен значению ФП ассоциированного с ней автомата.
— Все муравьи стартуют из вершины, ассоциированной с лучшим найденным решением.

Последний способ оказался наиболее эффективным и был выбран для последующего использования.

На втором этапе 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 двух действий: построение новых решений либо вероятностный выбор. Если у вершины u нет инцидентных ей ребер, то муравей всегда выполняет построение новых решений.

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 мутаций он выбирает лучшую из построенных вершин v 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, то создать ее и добавить в граф;
   — добавить в граф ребро ut.

   Построение новых решений проиллюстрировано на рисунке 11. Отметим, что переход осуществляется даже в том случае, когда значение ФП у лучшего из построенных с помощью мутаций автоматов меньше зна-

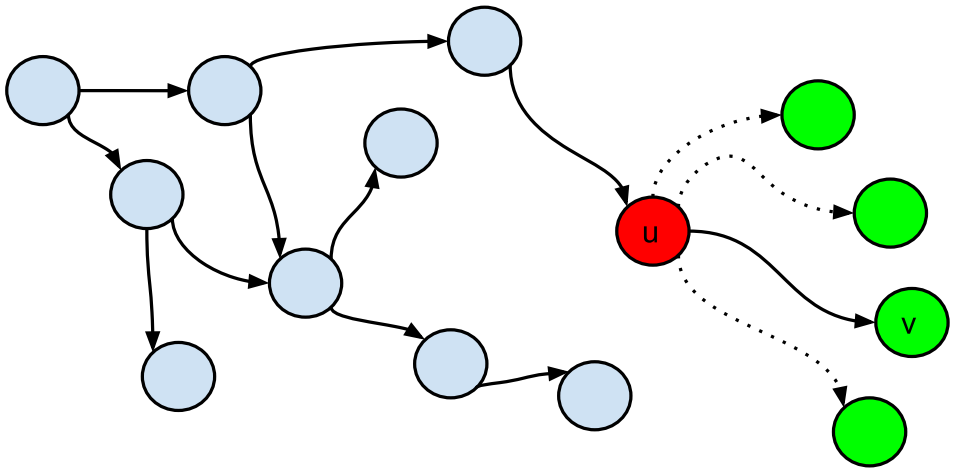

Рисунок 11 – Построение новых решений

чения ФП автомата A. Это свойство алгоритма позволяет ему выходить из локальных оптимумов.

2. **Вероятностный выбор**. С вероятностью $1 - p_{\text{new}}$ муравей выбирает следующую вершину из множества N_u вершин, инцидентных вершине u. Вершина v выбирается методом рулетки [100] с вероятностью p_{uv}, определяемой классической в муравьиных алгоритмах формулой [37]:

$$p_{uv} = \frac{\tau_{uv}^{\alpha} \cdot \eta_{uv}^{\beta}}{\sum_{w \in N_u} \tau_{uw}^{\alpha} \cdot \eta_{uw}^{\beta}}, \tag{2.1}$$

где $v \in N_u$, а $\alpha, \beta \in [1, 5]$ – параметры, определяющие значимость значений феромона и эвристической информации при выборе пути. Вероятностный выбор проиллюстрирован на рисунке 12. Во всех экспериментах значения параметров α и β были зафиксированы и равны единице.

Решения строятся набором из N_{ants} муравьев – *колонией*. Каждый муравей в колонии делает по одному шагу до тех пор, пока не остановится. Каждый муравей может выполнить не более n_{stag} шагов, на каждом из которых происходит построение новых решений либо вероятностный выбор, без увеличения своего значения ФП. Когда это ограничение превышено, муравей останавливается.

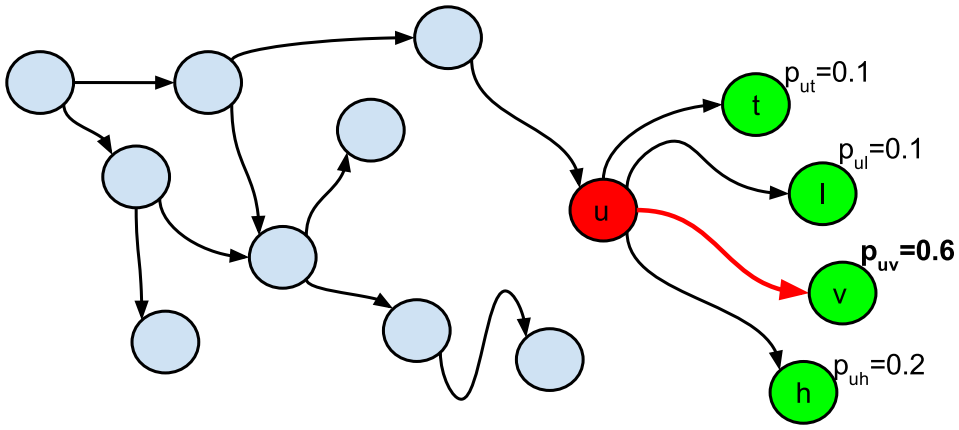

Рисунок 12 – Вероятностный выбор

Аналогично, колония муравьев может выполнить не более N_{stag} итераций без увеличения максимального значения ФП. Если указанное число итераций превышено, то считается, что алгоритм пришел в *состояние стагнации*. В этом случае он перезапускается.

2.6. Обновление значений феромона

После того, как все муравьи завершили построение решений на текущей итерации, выполняется обновление значений феромона на ребрах графа мутаций. Процедура состоит из двух частей – *откладывание феромона* на ребрах путей, пройденных муравьями и *испарение феромона* со всех ребер графа.

Значение феромона, которое муравей откладывает на ребрах своего пути, равно максимальному значению ФП всех автоматов, посещенных этим муравьем. Для каждого ребра uv графа G дополнительно хранится величина τ_{uv}^{best} – наибольшее из значений феромона, когда-либо отложенных на этом ребре. Последовательно рассматриваются все пути муравьев на текущей итерации. Для каждого пути выделяется отрезок от начала пути до вершины m, соответствующей автомату с наибольшим на пути значением ФП (отрезок выделен зеленым цветом на рисунке 13).

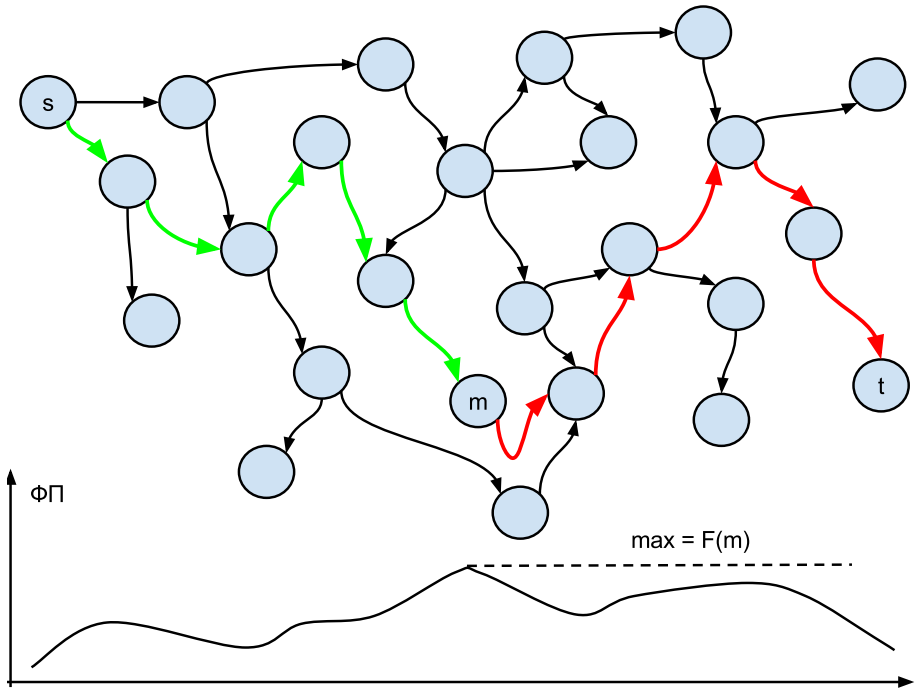

Рисунок 13 – Выделение отрезка пути муравья от начала до вершины, соответствующей автомату с наибольшим на пути значением ФП

Для всех ребер из этого отрезка обновляются значения τ_{uv}^{best}, а затем значения феромона на всех ребрах графа G обновляются по формуле:

$$\tau_{uv} = \max(\tau_{\min}, (1-\rho)\tau_{uv} + \tau_{uv}^{\text{best}}), \tag{2.2}$$

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

Псевдокод алгоритма приведен в листинге 1. Главной процедурой алгоритма является процедура `MuACO`. Сначала с помощью функции `randomSolution` создается случайное начальное решение, которое становится первой вершиной графа G. Затем, пока не выполнены условия останова (функция `terminate`), в цикле выполняется запуск колонии муравьев: построение новых решений выполняется процедурой `launchAntColony`, обновление значений феромона выполняется функцией `updatePheromone`.

Процедура `launchAntColony` описывает процесс конструирования новых решений. Сначала для каждого муравья функцией `selectStartNodes` выби-

Листинг 1 – Псевдокод метода MuACO

```
procedure MuACO
    x ← randomSolution( )
    best ← x
    G ← new MutationGraph(x)
    while !terminate( ) do
        paths ← launchAntColony(G)
        best ← bestSolution(paths)
        updatePheromone(paths, best, G)
    end while
return best
end procedure
procedure launchAntColony(G)
    startNodes ← selectStartNodes(G)
    ants ← new Ant()
    for node: startNodes do
        ants[i] ← Ant(node)
    end for
    while not all ants stopped do
        for ant : ants do
            if not ant.isStopped() then
                ant.step( )
            end if
        end for
    end while
end procedure
```

рается стартовая вершина. Затем муравьи совершают шаги (метод `step`) до тех пор, пока не будут остановлены.

2.7. Генерация начального решения по сценариям работы с помощью алгоритма на основе решения задачи выполнимости

В работе [132] диссертантом был предложен способ выбора начального решения для метода *MuACO*, позволяющий увеличить его эффективность. Было предложено вместо генерации случайного начального решения строить его только по сценариям работы с помощью точного алгоритма на основе решения задачи *CSP*, предложенного В. Ульянцевым в той же работе. Отметим что, в общем случае, построенное таким способом начальное решение не будет удовлетворять заданным *LTL*-формулам. Позже было установлено, что использование точного алгоритма на основе решения задачи *SAT* [46] более эффективно. Назовем комбинированный алгоритм, в котором начальное решение для метода *MuACO* строится с помощью точного алгоритма *efsmSAT* на основе решения задачи выполнимости, *SAT+MuACO*.

2.8. Вычислительные эксперименты

В настоящем разделе приводится описание вычислительных экспериментов по сравнению эффективности предложенного метода *MuACO* на основе муравьиного алгоритма и метода на основе генетического алгоритма [11, 12]. Описывается способ генерации входных данных, обосновывается выбор способа сравнения эффективности алгоритмов, процедура настройки значений параметров сравниваемых алгоритмов, приводятся данные о показателях эффективности алгоритмов и их статистическом анализе.

2.8.1. Подготовка входных данных

В экспериментах использовалось два множества примеров задачи – обучающее и тестовое. Обучающее множество использовалось при настройке значений параметров алгоритмов, а тестовое – для сравнения эффективности алгоритмов. Обучающее множество состояло из 200 примеров, каждый пример состоял из набора сценариев и набора темпоральных формул.

Для создания каждого примера сначала генерировался случайный конечный автомат, число состояний $|Y|$ которого выбиралось случайным образом из отрезка $[4, 10]$. Автоматы содержали два входных события, одну входную переменную, а длина последовательности выходных воздействий варьировалась от нуля до двух. По каждому автомату был сгенерирован набор из $5|Y|$ сценариев общей длиной $100|Y|$, а также, для примера, две *LTL*-формулы.

Каждый сценарий соответствует случайному пути в автомате. Начало каждого сценария совпадает с начальным состоянием автомата. Длина сценария равна числу переходов, вошедших в путь.

Опишем способ генерации *LTL*-формул, которым удовлетворяет автомат A. Алгоритму построения формул подается на вход автомат A, для которого их необходимо построить, и выбранное число формул n_{LTL} (здесь $n_{\text{LTL}} = 2$). Сначала с помощью программного средства *randltl* [101] генерируется $100n_{\text{LTL}}$ случайных LTL-формул, причем число вершин в дереве разбора формулы не превышает 15. Далее, с помощью верификатора [12] отбираются только те формулы, которым A удовлетворяет.

Однако необходимо, чтобы построенные формулы были не только верны для автомата A, но и были достаточно сложны. Для того, чтобы генерировать такие формулы, используется понятие эмпирической сложности формулы $h(\varphi)$. Сложность формулы измеряется с помощью следующего алгоритма.

Генерируется 50 случайных автоматов. Сложность $h(\varphi)$ формулы φ вычисляется как доля случайных автоматов, не удовлетворяющих формуле. Формула φ добавляется в итоговое множество формул, если $h(\varphi) > 0,5$. Процесс генерации формул прерывается, когда получено требуемое число формул. В случае, когда было сгенерировано недостаточное число формул, процесс запускается заново.

2.8.2. Сравнение эффективности алгоритмов

В диссертации эксперимент по сравнению эффективности двух алгоритмов A и B заключается в запуске каждого алгоритма на выбранном наборе примеров задачи. Существует два основных способа проведения экспериментов по сравнению эффективности метаэвристических (и не только) алгоритмов поисковой оптимизации. В первом способе каждому алгоритму на каждый его запуск отводится определенное одинаковое количество ресурсов. Например, можно зафиксировать максимальное число вычислений ФП или максимальное время работы алгоритма. Тогда в результате проведения эксперимента получится набор решений, не все из которых являются оптимальными. При таком подходе к сравнению эффективности алгоритмов сопоставляются выборки из распределений значений ФП – например, могут сравниваться их медианы или средние значения.

Второй подход – убедиться, что для каждого примера задачи существует оптимальное решение и разрешить алгоритму работать до тех пор, пока он не найдет это оптимальное решение. Тогда в качестве характеристики одного запуска алгоритма может рассматриваться число вычислений ФП либо время, которое потребовалось для нахождения оптимального решения. Эффективность алгоритмов сравнивается путем сопоставления выборок из распределений числа вычислений ФП или времени работы.

Первый подход обладает существенным недостатком, который можно пояснить на следующем примере. Пусть при некотором ограничении $t_{\max}$ на время работы алгоритмов A и B было установлено, что алгоритм A находит оптимальное решение в 90% случаев, а алгоритм B – только в 80%. Проблема заключается в том, что этот результат ничего не говорит о соотношении эффективности этих алгоритмов при ограничении $t^*_{\max} > t_{\max}$ или $t^*_{\max} < t_{\max}$: например, при большем ограничении по времени алгоритм A может находить оптимум в тех же 90% случаев, а B – в 95%.

Кроме этого, первый подход имеет смысл только в том случае, когда неоптимальное решение может так или иначе удовлетворить нужды пользователя. Например, решение задачи коммивояжера, на 5% отличающееся от оптимального, скорее всего, удовлетворит заказчика. Однако автомат, правильно обрабатывающий только 95% сценариев или удовлетворяющий только части темпоральных формул, не может быть использован. Поэтому в настоящей работе используется второй способ проведения экспериментов.

Помимо сравнения медианных значений выборок также проводится статистическое исследование результатов экспериментов, для чего используется непараметрический тест Уилкоксона [102]. При проведении множественных статистических тестов вычисленные значения *p-value* корректируются с использованием метода Холма [103].

2.8.3. Настройка значений параметров

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 свободно распространяемое программное средство *irace* [104] для настройки значений параметров алгоритмов. Суть процедуры настройки значений параметров в следующем. Пусть задано распределение примеров задачи P и необходимо выбрать значения параметров алгоритма, которые позволят хорошо решать примеры из этого распределения. 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 ите-

Таблица 1 – Значения параметров метода *MuACO*, полученные с помощью *irace*

Параметр	**Значение**
Число муравьев N_{ants}	4
Число мутаций N_{mut}	44
Максимальное число итераций колонии без увеличения значения ФП N_{stag}	28
Максимальное число шагов муравья без увеличения значения ФП n_{stag}	45
Скорость испарения феромона ρ	0,52

Таблица 2 – Значения параметров генетического алгоритма, полученные с помощью *irace*

Параметр	**Значение**
Размер популяции	201
Доля элитарных особей	0,25
Вероятность мутации	0,06
Число итераций до малой мутации поколения	483
Число итераций до большой мутации поколения	100

рации неэффективные конфигурации отбрасываются с помощью следующего механизма. Настраиваемый алгоритм параметризуется конфигурацией и последовательно запускается для нескольких примеров из I, вычисляются показатели эффективности алгоритма. Конфигурация считается неэффективной и отбрасывается, если набор значений показателей эффективности алгоритма для нее статистически хуже, чем для какой-либо другой конфигурации.

На настройку значений параметров было отведено 12 часов на компьютере с процессором *AMD Phenom(tm) II x4 955* @ 3,2 ГГц. В результате были

получены значения параметров метода *MuACO* и генетического алгоритма, указанные в таблице 1 и таблице 2, которые далее использовались во всех экспериментах данной главы.

2.8.4. Пример: генерация автомата управления дверьми лифта

В данном разделе решалась конкретная задача генерации автомата управления дверьми лифта. Система характеризуется пятью входными событиями:

— e_{11} – нажата кнопка «Открыть двери»;

— e_{12} – нажата кнопка «Закрыть двери»;

— e_2 – двери успешно открыты или закрыты;

— 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.

Входные переменные не используются. Спецификация состоит из девяти сценариев работы и 13 *LTL*-формул [12], которые приведены в Приложении Б. Поиск решений осуществлялся среди автоматов с не более, чем $N_{\text{states}} = 6$ состояниями. Автомат из пяти состояний, решающий задачу, изображен на рисунке 14.

В каждом эксперименте было проведено по 1000 запусков метода *MuACO* и генетического алгоритма, каждый запуск продолжался до достижения лучшим решением значения ФП, равного 2,0075, что соответствует автомату, полностью удовлетворяющему заданным сценариям работы и темпоральным формулам. Для сравнения указанных алгоритмов измерялось медианное время работы и число вычислений ФП. Медианное время работы генетического

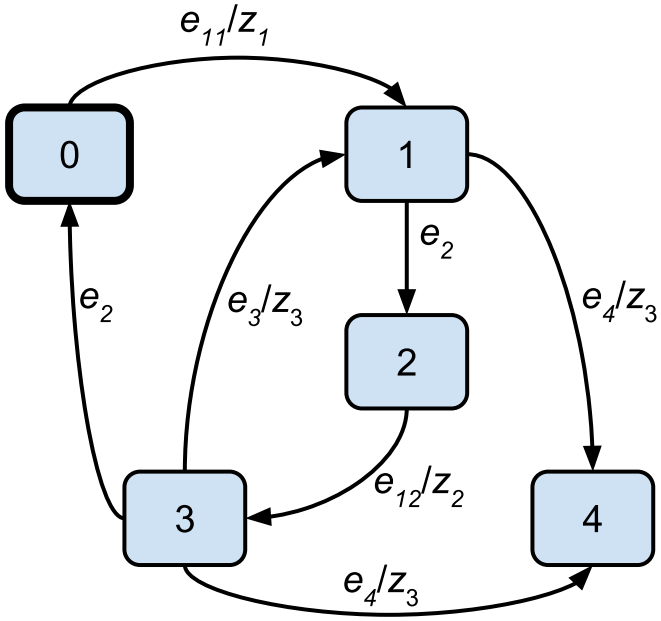

Рисунок 14 – Автомат управления дверьми лифта

алгоритма составило 7,3 с, а муравьиного – 3,5 с. Медианное число вычислений ФП генетического алгоритма равно 41902, а муравьиного – 10119.

На рисунке 15 приведена ящичная диаграмма, изображающая распределения запусков муравьиного и генетического алгоритмов по времени работы. Нижний и верхний края «ящика» изображают первый и третий квартили распределения, черта внутри ящика соответствует медиане, а горизонтальные линии снизу и сверху от ящика обозначают края статистически значимой выборки. Точки сверху от этих линий соответствуют выбросам.

Полученные в эксперименте результаты позволяют утверждать, что муравьиный алгоритм решает рассматриваемую задачу в два раза быстрее генетического алгоритма. Более того, как видно из рисунка 15, муравьиный алгоритм, по сравнению с генетическим алгоритмом, обеспечивает меньший разброс времени работы. Статистическая значимость полученных результатов была подтверждена с помощью теста Уилкоксона [102] (значение p-$value = 3,93 \cdot 10^{-13}$).

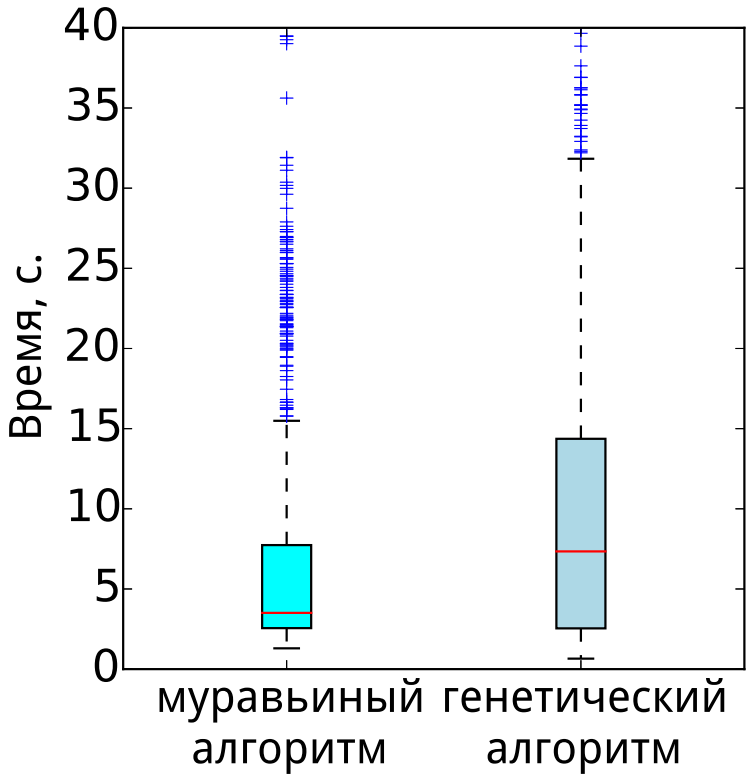

Рисунок 15 – Ящичная диаграмма распределений времени работы муравьиного и генетического алгоритмов по времени работы

2.8.5. Генерация автоматов по случайно сгенерированным входным данным

Во второй части экспериментального исследования рассматривались случайно сгенерированные автоматы, содержащие от четырех до десяти состояний.

Каждый эксперимент продолжался до нахождения алгоритмом решения, удовлетворяющего заданным сценариям и темпоральным формулам. Замерялось число совершенных вычислений ФП, а также время работы алгоритмов. Графики зависимости медианного значения числа вычислений ФП от числа состояний автомата приведены на рисунке 16, а ящичные диаграммы распределений запусков алгоритмов по времени работы изображены на рисунке 17. Из графиков и диаграмм видно, что метод *MuACO* быстрее генетического алгоритма, а метод *SAT+MuACO* быстрее, чем *MuACO*.

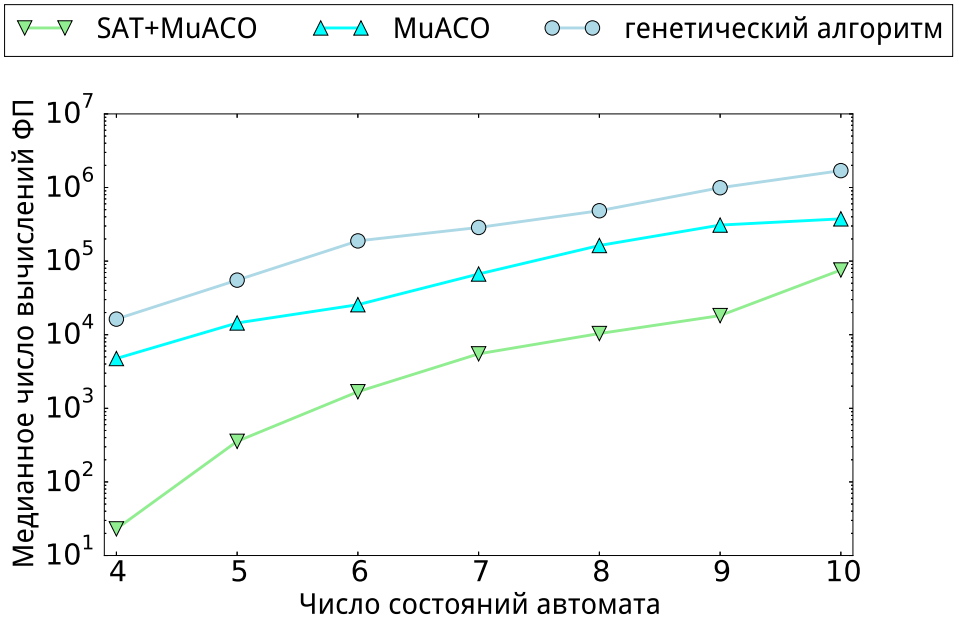

Рисунок 16 – Графики зависимости медианных значений числа вычислений ФП от числа состояний автомата для методов *MuACO*, *SAT+MuACO* и генетического алгоритмов

Для проверки статистической значимости этих результатов был использован статистический тест Уилкоксона, вычисленные значения *p-value* приведены в таблице 3. Тест проводился отдельно для каждого значения числа состояний автомата и каждой пары алгоритмов. Во втором столбце приведены значения *p-value*, полученные при сравнении метода *MuACO* с генетическим алгоритмом, а в третьем столбце – значения, вычисленные при сравнении метода *SAT+MuACO* с *MuACO*.

Нулевая гипотеза утверждала, что медиана разности выборок равна нулю, а альтернативная – что медиана разности выборок меньше нуля. Предельный уровень значимости был равен 0,05. Так как проводился не один, а несколько тестов, полученные значения *p-value* были откорректированы по методу Холма [103]. Как видно из таблице 3, все значения *p-value* меньше предельного уровня значимости.

Результаты статистических тестов позволяют сделать вывод о том, что на случайных данных предложенный метод *MuACO* работает существенно

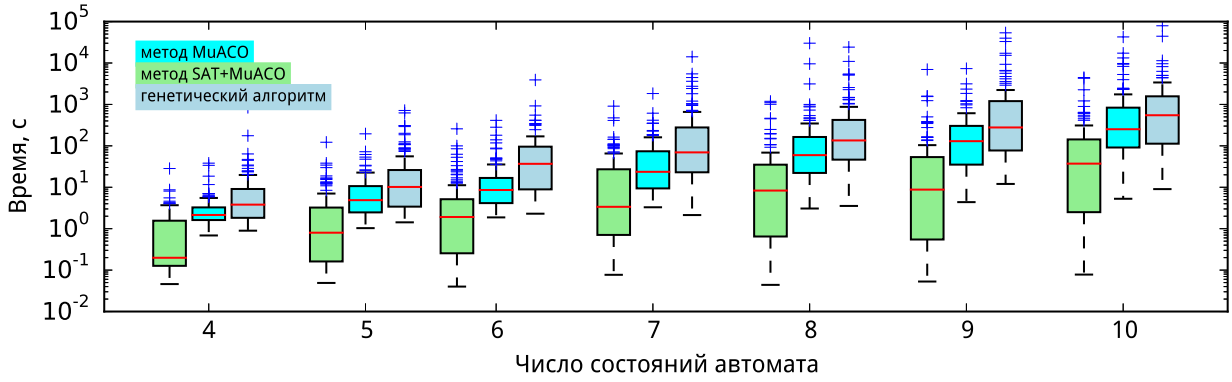

Рисунок 17 – Ящичные диаграммы распределений запусков методов *MuACO*, *SAT+MuACO* и генетического алгоритма по времени работы для экспериментов со случайными автоматами с $N_{\text{states}} \in [4, 10]$

быстрее генетического алгоритма, а метод *SAT+MuACO*, в свою очередь, существенно быстрее метода *MuACO*.

2.9. Сравнение с точными методами генерации конечных автоматов по *LTL*-формулам

В настоящем разделе приводятся результаты экспериментального сравнения разработанного в настоящей диссертации метода и методов, реализованных в программных средствах *lily* [55], *unbeast* [56] и *G4LTL_ST* [57].

Сравнение проводилось на примере решения задачи генерации автомата управления дверьми лифта из раздела 2.8.4. Во всех упомянутых программных средствах поддерживаются только входных и выходные переменные, а события не рассматриваются. В связи с этим для представления событий приходится моделировать их с помощью входных переменных. Так, например, событие e_1 записывается с помощью формулы $e_{11} \wedge \neg e_{12} \wedge \neg e_2 \wedge \neg e_3 \wedge \neg e_4$. Аналогично, выходное воздейтсвие z_1 кодируется как $z_1 \wedge \neg z_2 \wedge \neg z_3$.

Также ни одно из рассмотренных программных средств не поддерживает использование сценариев работы в качестве входных данных. Поэтому сценарии записываются в виде *LTL*-формул. Например, сценарий $e_{11}/z_1; e_2; e_{12}/z_2;$ для автомата Мили можно выразить с помощью следующей *LTL*-формулы:

Таблица 3 – Результаты статистического теста Уилкоксона для экспериментов со случайно сгенерированными автоматами. Сравниваются генетический алгоритм (I), метод *MuACO* (II) и метод *SAT+MuACO* (III)

$\boldsymbol{N}_{\text{states}}$	Значение *p-value*: II/I	Значение *p-value*: III/II
4	$6 \cdot 10^{-7}$	$4 \cdot 10^{-10}$
5	$1 \cdot 10^{-5}$	$4 \cdot 10^{-9}$
6	$7 \cdot 10^{-10}$	$4 \cdot 10^{-9}$
7	$1 \cdot 10^{-5}$	$3 \cdot 10^{-6}$
8	$4 \cdot 10^{-4}$	$6 \cdot 10^{-9}$
9	$1 \cdot 10^{-5}$	$1 \cdot 10^{-8}$
10	$5 \cdot 10^{-4}$	$4 \cdot 10^{-10}$

$$((e_{11} \wedge \neg \ldots) \rightarrow ((z_1 \wedge \neg z_2 \wedge \neg z_3) \wedge X((e_2 \wedge \neg \ldots) \rightarrow$$
$$\rightarrow ((\neg z_1 \wedge \neg z_2 \wedge \neg z_3) \wedge X((e_{12} \wedge \neg \ldots) \rightarrow (z_2 \wedge \neg z_1 \wedge \neg z_3))))))),$$

где в выражениях $e_* \wedge \neg \ldots$ «$\ldots$» обозначает конъюнкцию отрицаний всех входных переменных, кроме «e_*».

Программное средство *lily* [55] генерирует автоматы Мура, поэтому все сценарии и *LTL*-формулы были преобразованы соответствующим образом. Запуск *lily* на всех сценариях и *LTL*-формулах не завершился за 30 минут, поэтому было проведено несколько запусков, где в каждом следующем запуске добавлялась одна *LTL*-формула. Результаты экспериментов представлены в таблице 4. Данные результаты указывают на то, что даже для такого простого примера, как автомат управления дверьми лифта, время работы программного средства *lily* существенно превышает время работы предложенного в диссертации алгоритма, который находит решение, удовлетворяющее всем 13 формулам, в среднем за 3,5 с. Кроме того, сгенерированные с помощью *lily* автоматы содержат 15 состояний, что в три раза больше, чем

Таблица 4 – Результаты экспериментальных запусков программного средства *lily*

Число *LTL*-формул	Время работы, с
1	36
2	72
3	340
4	453

число состояний автомата, сгенерированного предложенным в диссертации алгоритмом.

Программное средство *unbeast* [56] генерирует автоматы Мили. Генерация автомата управления дверьми лифта по всем сценариям работы и темпоральным формулам занимает 5-6 с, что в 1,5 раза больше времени, которое требуется предложенному в диссертации алгоритму. Кроме того, *unbeast* генерирует решения с очень большим числом состояний. Решение генерируется в виде *бинарной диаграммы решений* [105] и не выдается на выход в явном виде. Однако число состояний можно оценить с помощью интерфейса для тестирования решения, включив при этом опцию вывода состояния игры. Так, найденное *unbeast* решение при обработке сценариев работы использует 11 состояний, что более чем в два раза больше размера модели, найденной предложенным в диссертации алгоритмом.

Программное средство *G4LTL_ST* также генерирует автоматы Мили. Генерация автомата, удовлетворяюшего всем сценариям и *LTL*-формулам заняла около 9 с (почти в три раза больше, чем предложенный алгоритм), число состояний этого автомата равно 30 (в шесть раз больше, чем число состояний автомата, найденного предложенным алгоритмом).

2.10. Использование предложенного метода *MuACO* для генерации автоматов управления моделью беспилотного самолета

В работе [131] предложенный метод *MuACO* был использован для генерации конечных автоматов управления моделью беспилотного самолета. Решалась задача генерации автомата, который смог бы воспроизвести действия пилота при выполнении двух фигур высшего пилотажа – мертвой петли и бочки. Для этого с помощью авиасимулятора *FlightGear* [125] фигуры пилотажа были выполнены вручную, при этом записывались сценарии работы.

В этой статье сценарий работы представлет собой последовательность элементов сценария, каждый из которых состоит из набора параметров полета (высота, скорость, углы крена, тангажа и курса) и параметров управления (положение элеронов, руля направления, руля высоты). Применение предложенного в настоящей диссертации метода *MuACO* ускорило построение автоматов по сравенению с генетическим алгоритмом, который ранее применялся для решения этой задачи в [106].

Выводы по главе 2

1. Предложен метод генерации конечных автоматов по сценариям работы и темпоральным формулам на основе муравьиного алгоритма с предложенным автором графом мутаций.
2. Проведены вычислительные эксперименты, результаты которых свидетельствуют о том, что предложенный метод превосходит по эффективности метод на основе генетического алгоритма, а также точные методы, реализованные в программных средствах *lily*, *unbeast*, и *G4LTL_ST*.
3. Результаты, описанные в данной главе, опубликованы в трудах международной конференции «Genetic and Evolutionary Computation Conference» [134, 141] и в журналах, входящих в перечень ВАК [129, 131].

4. Предложенный метод также применялся для решения задач, в которых функция приспособленности вычисляется на основе моделирования [133—135, 138, 146].

ГЛАВА 3. МЕТОДЫ ГЕНЕРАЦИИ КОНЕЧНЫХ АВТОМАТОВ ПО СЦЕНАРИЯМ РАБОТЫ И ТЕМПОРАЛЬНЫМ ФОРМУЛАМ НА ОСНОВЕ ПАРАЛЛЕЛЬНЫХ МУРАВЬИНЫХ АЛГОРИТМОВ

Данная глава посвящена описанию предлагаемых методов генерации конечных автоматов по сценариям работы и темпоральным формулам на основе параллельных муравьиных алгоритмов. В первом методе *pMuACO* (*parallel MuACO*) в каждом потоке запускается алгоритм *MuACO*, а взаимодействие между отдельными потоками реализовано с помощью архива лучших решений и операции кроссовера. Второй метод *psMuACO* (*parallel SAT MuACO*) отличается от *pMuACO* тем, что в качестве начального решения во всех потоках используется автомат, сгенерированный точным методом *efsmSAT* генерации автоматов только по сценариям работы, основанным на решении задачи выполнимости. Третий метод *pstMuACO* (*parallel SAT thin-out MuACO*) совмещает процедуру прореживания сценариев, алгоритм *efsmSAT* и метод *pMuACO*.

3.1. Метод генерации конечных автоматов *pMuACO*

В данном разделе описывается метод генерации конечных автоматов на основе параллельного муравьиного алгоритма, предложенный диссертантом в [142]. 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 [142], взаимодействие между отдельными потоками в алгоритме существенно повышает его эффективность. Реализовано два механизма взаимодействия, основанные на архиве K лучших решений всех по-

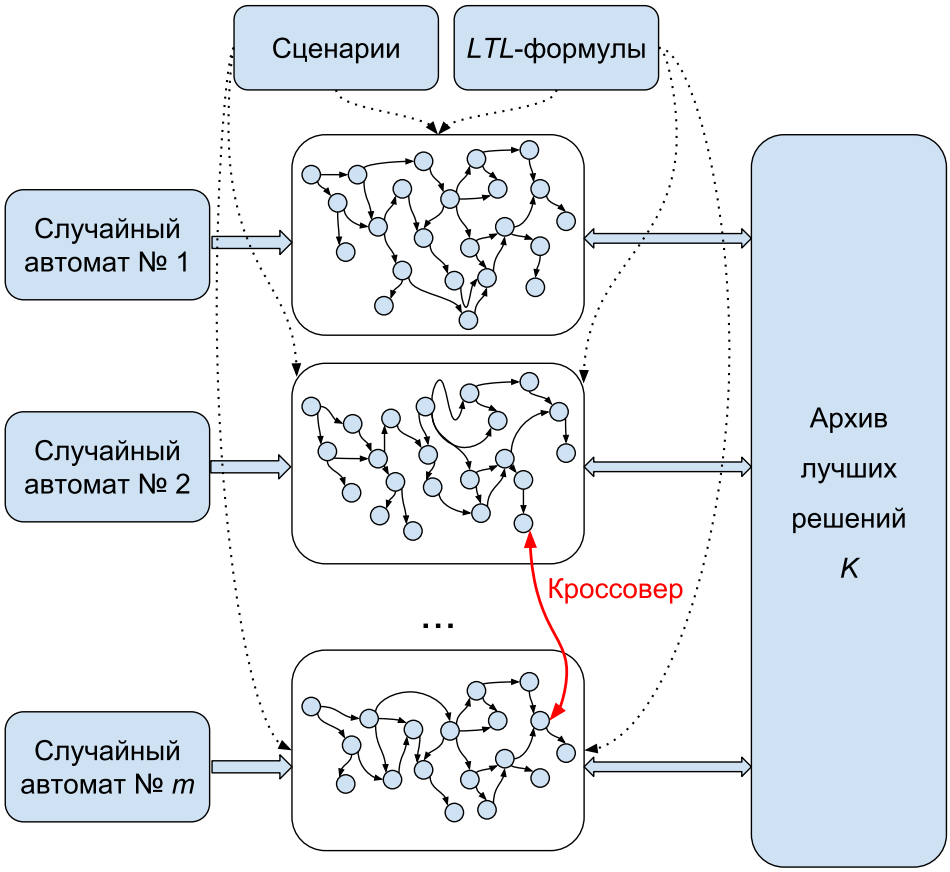

Рисунок 18 – Схема параллельного муравьиного алгоритма *pMuACO* с архивом лучших решений

токов. 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 стартовое решение не генерируется случайным образом, а выбирается из архива лучших решений $K \setminus \{K_i\}$ (рисунок 18).

Второй механизм работает на каждой итерации колонии каждого алгоритма *MuACO* и использует генетический оператор *скрещивания* управляющих конечных автоматов, предложенный в [52] и учитывающий поведение родительских особей при вычислении значения ФП. На вход оператор принимает на вход два автомата (родители) и возвращает два автомата (потомки). Каждый из потомков содержит переходы, выбранные из обоих родителей.

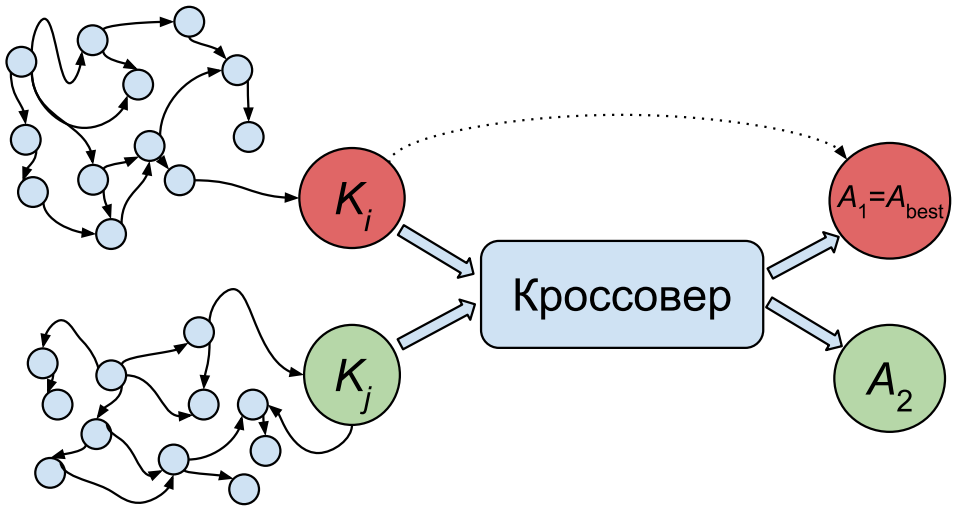

Рисунок 19 – Применение операции кроссовера в алгоритме *pMuACO*: лучшее решение K_i, найденное i-м алгоритмом, скрещивается с лучшим решением K_j, найденным j-м алгоритмом.

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. Из получившихся автоматов выбирается тот, значение ФП которого больше – A_{best}. Процесс применения оператора кроссовера проиллюстрирован на рисунке 19. Автомат A_{best} добавляется в граф мутаций i-го алгоритма *MuACO* в качестве ребенка вершины, ассоциированной с K_i (пунктирная линия на рисунке 19). 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.

3.2. Методы *psMuACO* и *pstMuACO*

В данном разделе описываются предлагаемые метод *psMuACO* и *pstMuACO*, в которых для генерации начальных решений применяется точный метод *efsmSAT* [46] генерации конечных автоматов только по сценариям работы, основанный на сведении к задаче выполнимости.

Данная модификация является расширением метода *CSP+MuACO* [132], описанного в разделе 2.7. В статье [132] диссертантом было предложено для получения начального приближения для метода *MuACO* использовать точный метод генерации 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 этих задач используются сторонние программные средства, такие как *cryptominisat* [126] и *Choco* [123].

В [132] было показано, что совместное применение точного алгоритма на основе *CSP* и метода *MuACO* [134] существенно сокращает необходимое для построения автоматов время. Хотя метод на основе *CSP* проще метода на основе *SAT*, позднее было установлено, что применение последнего эффективнее. Поэтому в диссертации для генерации автоматов по сценариям используется метод *efsmSAT* на основе *SAT*.

Очевидным развитием описанного подхода является его применение для получения начального приближения в параллельных алгоритмах. Назовем этот метод *psMuACO* (*parallel SAT MuACO*). Сначала с помощью точного алгоритма *efsmSAT* [46] генерации автоматов по сценариям работы на основе решения задачи выполнимости, строится автомат, удовлетворяющий только сценариям работы. Далее этот автомат используется в качестве начального решения для всех потоков в методе *pMuACO*. Схема метода *psMuACO* приведена на рисунке 20.

Стоит однако учесть, что параллельные алгоритмы комбинаторной оптимизации достигают наибольшей эффективности, если поиск в разных потоках начинается с разных начальных решений. Это объясняется тем, что при различных начальных решениях существенно возрастает размер пространства поиска, исследуемого алгоритмом в единицу времени [107]. Заметим, что при условии использования точных программных средств решения *SAT* и *CSP*,

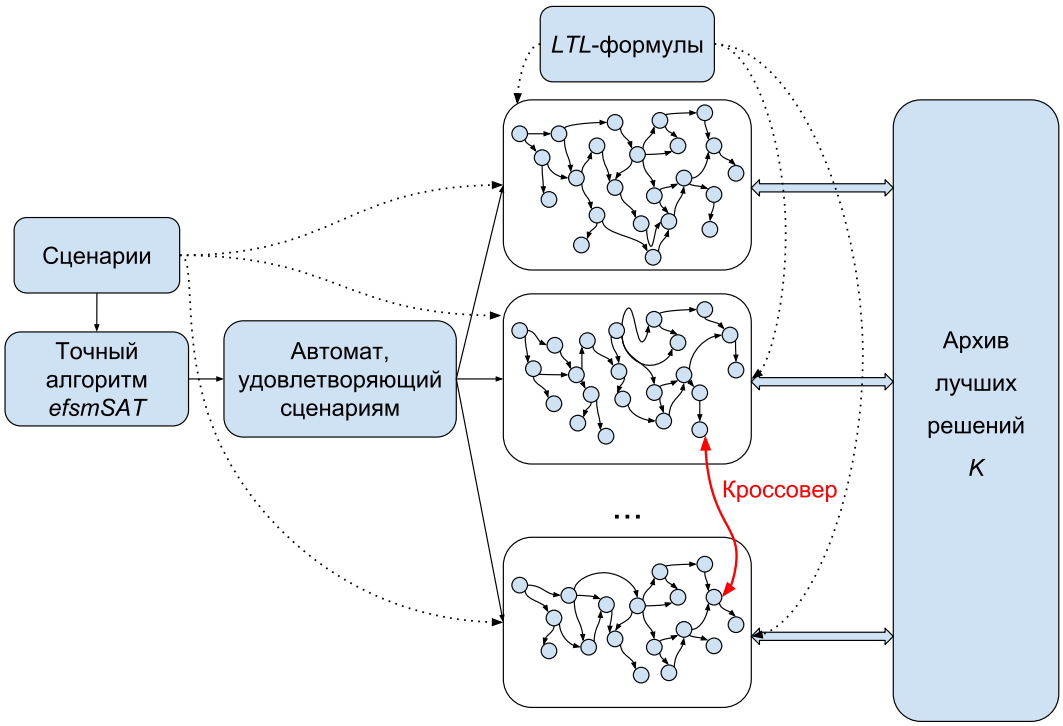

Рисунок 20 – Схема метода *psMuACO*

алгоритмы на их основе обладают детерминированным поведением – по одному и тому же набору сценариев они всегда строят один и тот же автомат. Для применения 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 $T'_0, \ldots, T'_{n_{\text{start}}-1}$. Каждый такой набор получается из исходного набора T путем прореживания – удаления каждого сценария с вероятностью p_{thin}.
2. **Построение начальных решений по сценариям с помощью метода *efsmSAT***. Каждый i-й автомат строится по сокращенному набору сценариев T'_i с помощью метода *efsmSAT* на основе сведения к

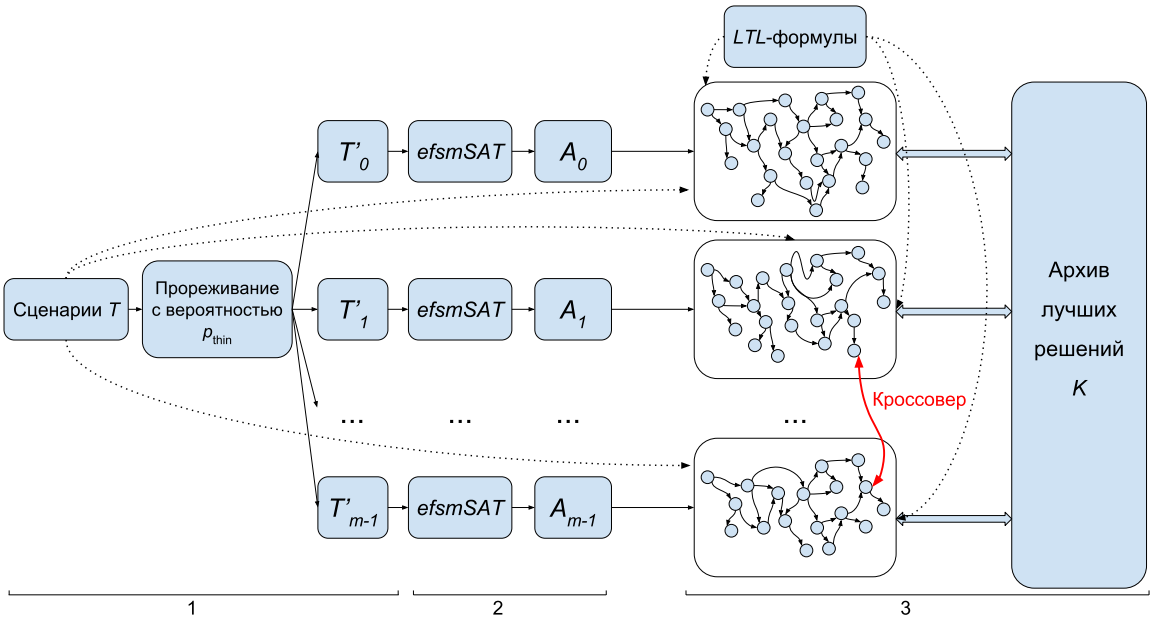

Рисунок 21 – Схема предлагаемого метода *pstMuACO*

задаче *SAT* [46]. 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 $n_{\text{start}} = m$, i-й автомат, построенный по сокращенному набору сценариев $T_i^{'}$, используется в качестве начального решения для i-го потока алгоритма. Если $n_{\text{start}} < m$, то начальное решение в каждом потоке выбирается случайным образом из множества сгенерированных начальных решений.

Число генерируемых с помощью *efsmSAT* автоматов n_{start} и вероятность удаления сценария p_{thin} являются параметрами алгоритма. Схема предлагаемого метода *pstMuACO* (*parallel SAT thinned out MuACO*) для случая, когда $m = n_{\text{start}}$, приведена на рисунке 21.

3.3. Вычислительные эксперименты

В данном разделе описываются проведенные вычислительные эксперименты, процесс подготовки входных данных и выбор значений параметров алгоритмов. Проводилось сравнение методов *pMuACO*, *psMuACO* и *pstMuACO* и генетического алгоритма. 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.

Входные данные генерировались так же, как в разделе 2.8.1 за тем исключением, что число состояний для каждого генерируемого автомата выбиралось из отрезка $[10, 20]$. Значения параметров методов настраивались с помощью программного средства *irace*, на настройку каждого из них было отведено по 24 часа. Обучающий набор для настройки состоял из 200 примеров, построенных по случайно сгенрированным автоматам из 10–20 состояний. Полученные значения параметров методов на основе муравьиных алгоритмов приведены в таблице 5, а значения параметров генетического алгоритма следующие: размер популяции равен 201, доля элитарных особей равна 0,25, вероятность мутации равна 0,06, период малой мутации поколения равен 483, а большой – 100.

Для тестирования было сгенерировано три набора из 50 примеров каждый, построенные по автоматам из 10, 15 и 20 состояний, соответственно. Ящичные диаграммы распределений времени работы методов приведены на рисунке 22.

Из диаграмм видно, что метод *pstMuACO* во всех случаях существенно быстрее, чем *psMuACO*, который, в свою очередь, быстрее метода *pMuACO*. Для автоматов из 10 и 15 состояний метод *pMuACO* быстрее генетического алгоритма, при этом 16 запусков генетического алгоритма не завершились за шесть часов. Поэтому для 20 состояний генетический алгоритм не запускался.

Таблица 5 – Значения параметров методов *pMuACO*, *psMuACO* и *pstMuACO*, полученные с помощью *irace*

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

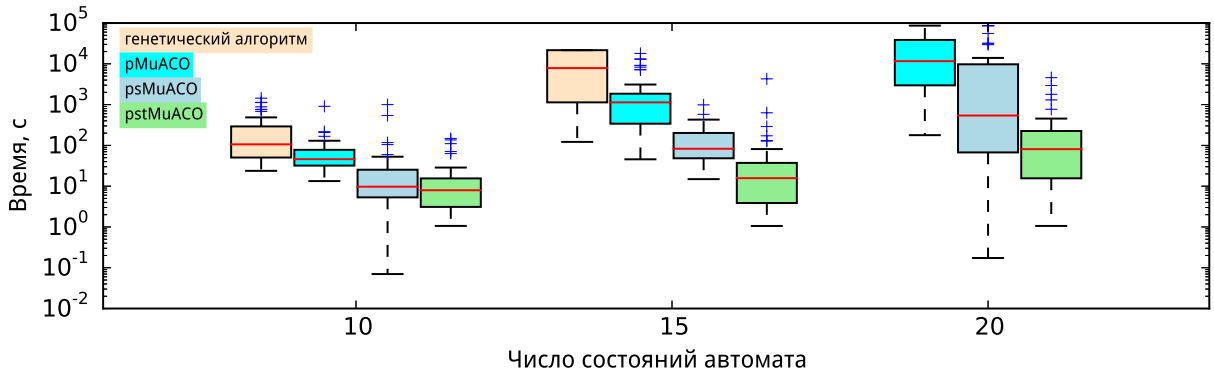

Рисунок 22 – Ящичные диаграммы распределений времени работы методов *pMuACO*, *psMuACO*, *pstMuACO* и генетического алгоритма

Отметим также, что для автоматов из 20 состояний в шести из 50 случаев методы *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.

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

Для проверки статистической значимости различий во времени работы методов использовался тест Уилкоксона [102]. Тест отдельно проводился для каждого значения числа состояний и каждой пары методов. Нулевая гипотеза утверждала, что медиана разности выборок равна нулю. Альтернативная гипотеза утверждала, что медиана разности выборок меньше нуля. Предельный уровень значимости был равен 0,05. Приводятся значения *p-value* после корректировки по методу Холма [103]. Для *pMuACO* и генетического алгоритма были получены следующие значений *p-value*: $7,6 \cdot 10^{-6}$ (10 состояний) и $1,4 \cdot 10^{-6}$ (15 состояний). Для пары *pMuACO* и *psMuACO* были получены следующие значений *p-value*: $1,035 \cdot 10^{-5}$ (10 состояний), $2,2 \cdot 10^{-4}$ (15 состояний) и $2,5 \cdot 10^{-3}$ (20 состояний). Для пары *psMuACO* и *pstMuACO* были получены следующие значения *p-value*: $0,03$ (10 состояний), $3,264 \cdot 10^{-6}$ (15 состояний) и $7,8 \cdot 10^{-5}$ (20 состояний). Эти результаты показывают, что метод *psMuACO* во всех случаях статистически быстрее метода *pMuACO*, а метод *pstMuACO* статистически быстрее метода *psMuACO*.

Выводы по главе 3

1. Предложен метод *pMuACO* на основе параллельного запуска нескольких взаимодействующих алгоритмов *MuACO* и превосходящий по производительности однопоточный алгоритм *MuACO* и независимо-параллельный генетический алгоритм.
2. Предложен метод *psMuACO*, в котором начальное решение для всех потоков генерируется с помощью точного алгоритм *efsmSAT* построения автоматов по сценариям работы. Показано, что *psMuACO* работает быстрее метода *pMuACO*.
3. Предложен метод генерации конечных автоматов по сценариям работы и темпоральным формулам *pstMuACO*, совмещающий параллельный муравьиный алгоритм *pMuACO*, точный алгоритм *efsmSAT* и процедуру прореживания сценариев. За счет генерации нескольких начальных решений по сокращенным наборам сценариев, этот метод работает быстрее, чем ранее применявшиеся подходы. Так, для автоматов из 20 состояний новый алгоритм в среднем в 40 раз быстрее метода *pMuACO* и в 11 раз быстрее метода *psMuACO*.
4. Часть описанных в данной главе результатов опубликована в трудах международной конференции «International Student Workshop on Bioinspired Optimization Methods and their Applications» [142]. Статья по остальным результатам данной главы принята к печати в журнал «Автоматика и телемеханика»: Чивилихин Д.С., Ульянцев В.И., Шалыто А.А. Модифицированный муравьиный алгоритм для построения конечных автоматов по сценариям работы и темпоральным формулам.

ГЛАВА 4. ИНСТРУМЕНТАЛЬНОЕ СРЕДСТВО И БИБЛИОТЕКА ДЛЯ ГЕНЕРАЦИИ КОНЕЧНЫХ АВТОМАТОВ

Разработанные в диссертации методы и алгоритмы были реализованы в виде инструментального средства *muaco.jar*, исходный код которого доступен в сети Интернет по адресу `https://bitbucket.org/chivilikhin_daniil/aco-for-automata`. Программное средство можно использовать как для генерации управляющих конечных автоматов по сценариям работы и темпоральным формулам, так и в качестве библиотеки для реализации классов, позволяющих генерировать автоматы для других задач. На программное средство и библиотеку получены свидетельства о регистрации программ для ЭВМ (Приложение А).

Программная реализация инструментального средства выполнена на языке программирования *Java*. Схема взаимодействия основных классов программной реализации приведена на рисунке 23.

Точка входа в приложение расположена в классе `OptimizationRunner`. Здесь происходит чтение конфигурационных файлов и запуск соответствующего алгоритма.

Класс `TestsModelCheckingTask` осуществляет чтение входных данных (сценариев работы и темпоральных формул), а также реализует вычисление значения функции приспособленности автомата.

Класс `FSM` реализует представление управляющего конечного автомата, который хранится в виде полной таблицы переходов. Также в этом классе реализована операция хэширования автоматов, необходимая для реализации графа мутаций.

`Mutator` – абстрактный класс, наследники которого реализуют различные операторы мутации автоматов. Элементарная мутация конечного автомата представлена классом `FsmMutation`, и характеризуется состоянием, из которого выполняется переход и состоянием, в которое выполняется переход.

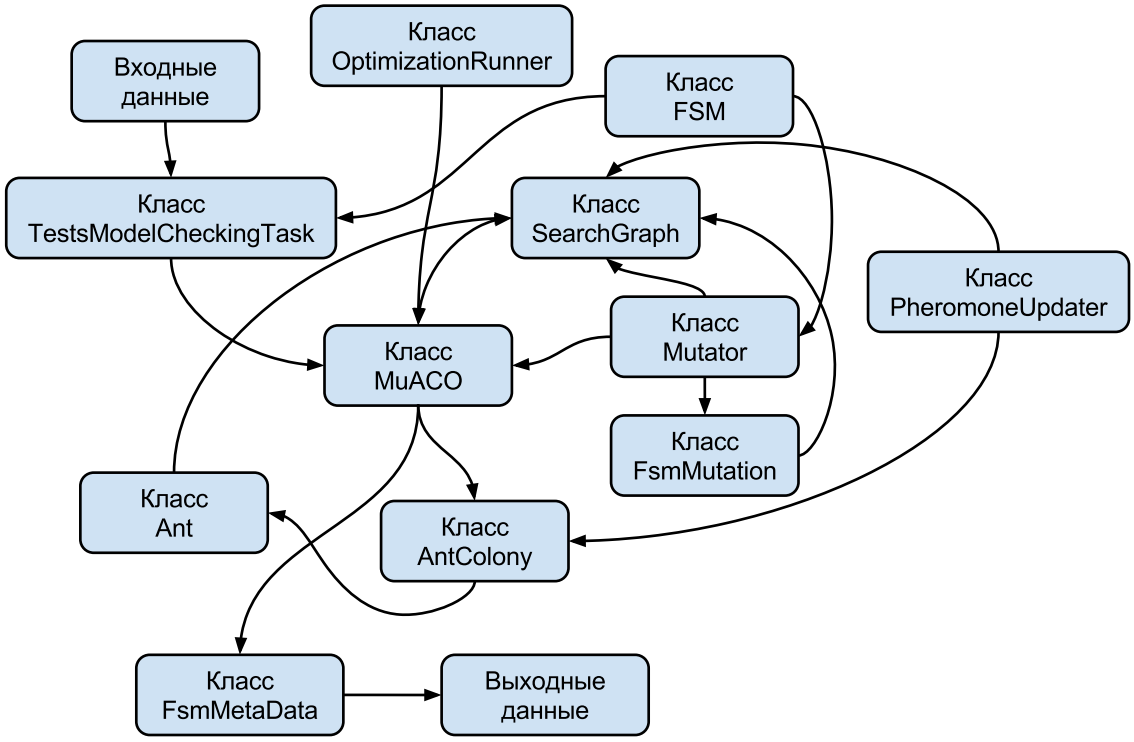

Рисунок 23 – Схема основных классов программной реализации инструментального средства

Граф мутаций реализован в классе `SearchGraph`. Отметим, что все автоматы в явном виде в графе не хранятся. Вместо этого хранится только первый автомат, а на ребрах графа хранятся мутации автоматов. Дополнительно для каждой вершины хранится ссылка на ее родителя. Также хранится хэш-таблица, позволяющая по автомату найти вершину графа, которой он соответствует. Для восстановления автомата с помощью хэш-таблицы находится вершина, соответствующая автомату. Затем, с помощью прохода по ссылкам на родителя до корневой вершины, составляется список мутаций, которые нужно применить к первому автомату, чтобы получить требуемый автомат.

Основная логика работы алгоритма реализована в классе `MuACO`. В нем генерируется случайное начальное решение, на основе которого создается граф мутаций. Далее запускается муравьиная колония, реализованная в клас-

се `AntColony`. В этом классе на каждой итерации запускаются муравьи. Логика работы муравья реализована в классе `Ant`.

Класс `PheromoneUpdater` реализует процедуру обновления значений феромона на ребрах графа мутаций.

Когда алгоритм завершает свою работу, на основе лучшего найденного решения создается экземпляр класса `FsmMetaData`, содержащий автомат, а также данные о запуске алгоритма – время работы, число вычислений ФП и т. д. Результат экспортируется в текстовом виде: в файл с расширением `.gv` записывается граф переходов автомата в формате *GraphViz* [127], а в файл с расширением `.metadata` – данные о запуске алгоритма.

4.1. Использование разработанного инструментального средства для генерации конечных автоматов по сценариям работы и темпоральным формулам

Для запуска процесса генерации автоматов необходимо оформить несколько конфигурационных файлов. Головным файлом является *experiment.properties*, пример которого приведен в листинге 2. Имена, пояснения и возможные значения параметров приведены в таблице 6.

Листинг 2 – Пример конфигурационного файла *experiment.properties*

```
algorithm-type=MUACO
instance-type=FSM
task-config-file-name=tests-model-checking.properties
solution-dir-name=attempts
number-of-experiments=1
max-evaluations=-1
max-run-time=-1
```

В отдельном конфигурационном файле приводится описание решаемой задачи: тип используемой ФП (*class-name*), целевое значение ФП (*desired-fitness*), число состояний автомата (*desired-number-of-states*), имя файла,

Таблица 6 – Описание параметров файла *experiment.properties*

Параметр	**Пояснение**	**Возможные значения**
algorithm-type	тип используемого алгоритма	*MUACO, PARALLEL_MUACO, CROSSOVER_AND_SHARED_BEST_PARALLEL_MUACO*
instance-type	тип особи алгоритма	*FSM*
task-config-file-name	имя конфигурационного файла, содержащего описание решаемой задачи	–
max-evaluations	ограничение на число вычислений ФП	-1 (без ограничений) или натуральное число
max-run-time	ограничение на время работы	-1 (без ограничений) или положительное действительное число
solution-dir-name	путь к директории, в которую будут записаны результаты работы	–

содержащего описание исходных данных для построения автоматов (*tests*). Пример такого файла для задачи генерации конечных автоматов по сценариям работы и темпоральным формулам приведен в листинге 3.

Листинг 3 – Пример конфигурационного файла задачи *tests-model-checking.properties*

```
class-name=MODEL_CHECKING
desired-number-of-states=5
desired-fitness=2
tests=problem.properties
```

Далее, в отдельном файле приводится описание входных данных для генерации автоматов. Пример такого файла *problem.properties* приведен в листинге 4. В нем указываются все встречающиеся в сценариях комбинации входных событий и охранных условий (*events*), все возможные выходные воздействия (*actions*), имя файла, содержащего сценарии работы (*scenarios*). Также указывается, следует ли использовать темпоральные формулы при вычислении ФП (*use-formulas*), а также имя файла, содержащего формулы (*formulas*). Пример наборов сценариев и формул приведен в Приложении Б.

Листинг 4 – Пример файла с описанием входных данных для генерации автоматов *problem.properties*

```
events=A[1] A[!x0] A[x0] B[1] B[!x0] B[x0]
actions=z0,z1
scenarios=scenarios
use-formulas=true
formulas=formulae
```

Наконец, в конфигурационном файле *muaco.properties* указываются значения параметров алгоритма. Пример файла приведен в листинге 5. Указывается тип генератора случайных решений (*instance-generator*), число мура-

вьев в колонии (*number-of-ants*), скорость испарения феромона (*evaporation-rate*), значение параметра n_{stag} (*stagnation-parameter*), значение параметра N_{stag} (*big-stagnation-parameter*), типы операторов мутации (*mutators*), максимальное число вершин в графе мутаций (*max-number-of-nodes*), число используемых потоков (*number-of-threads*).

Листинг 5 – Пример конфигурационного файла *muaco.properties* метода *MuACO*

```
instance-generator=FSM
number-of-ants=4
evaporation-rate=0.52
stagnation-parameter=45
big-stagnation-parameter=28
mutators=VERIFICATION_CHANGE_DEST,ADD_DELETE_TRANSITIONS
max-number-of-nodes=2000000
number-of-threads=1
```

Для запуска генерации автоматов необходимо поместить все конфигурационные файлы в одну директорию с исполняемым файлом программного средства `muaco.jar` и выполнить команду `java -jar muaco.jar`. В результате запуска генерируется файл, содержащий описание автомата в формате *GraphViz* [127]. Также генерируется файл с расширением `.metadata`, содержащий дополнительные сведения о запуске алгоритма, например, время работы и число вычислений ФП.

4.2. Использование разработанного инструментального средства для решения других задач генерации конечных автоматов

В данном разделе приводится пошаговая инструкция по реализации классов, необходимых для применения алгоритма *MuACO* для решения других задач генерации конечных автоматов.

1. Реализация представления автоматов. Необходимо расширить абстрактный класс автомата `AbstractFSM`. Также необходимо реализовать интерфейс `InstanceMutation`, определяющий элементарную мутацию автомата и вспомогательный интерфейс `InstanceMetaData`.
2. Реализация генератора случайных особей – необходимо реализовать интерфейс `InstanceGenerator`.
3. Реализация операторов мутации – необходимо реализовать интерфейс `Mutator`. Оператор мутации принимает на вход особь, представляющую автомат, и возвращает измененную особь, а также коллекцию совершенных мутаций типа `InstanceMutation`.
4. Реализация функции приспособленности – необходимо расширить абстрактный класс `AbstractAutomatonTask`.

Выводы по главе 4

1. Предложенные в диссертации алгоритмы реализованы в виде инструментального средства и библиотеки, исходный код которых находится в открытом доступе.
2. Инструментальное средство предназначено для генерации конечных автоматов по сценариям работы и темпоральным формулам. Получено свидетельство о регистрации программы для ЭВМ.
3. Библиотека предназначена для решения других задач генерации конечных автоматов. Получено свидетельство о регистрации программы для ЭВМ.

ГЛАВА 5. ВНЕДРЕНИЕ РЕЗУЛЬТАТОВ РАБОТЫ ПРИ ГЕНЕРАЦИИ АВТОМАТНОЙ ЛОГИКИ ДЛЯ БАЗИСНЫХ ФУНКЦИОНАЛЬНЫХ БЛОКОВ СТАНДАРТА *IEC 61499*

IEC 61499 [113, 58] – это стандарт в промышленной автоматике, определяющий открытую архитектуру для разработки систем распределенного и централизованного управления и автоматизации. Этот стандарт дополняет и расширяет стандарт *IEC 61131* [112], ориентированный на разработку систем управления на основе программируемых логических контроллеров (ПЛК). Элементарным компонентом стандарта *IEC 61499* является *функциональный блок* (ФБ). Функциональные блоки характеризуются *интерфейсом*, который определяет использующиеся входные/выходные события и входные/выходные переменные. *Базисный* ФБ задается с помощью событийной модели, называемой диаграммой управления выполнением (ДУВ), и представляющей собой конечный автомат Мура специального вида. *Составной* ФБ задается сетью других ФБ, среди которых могут быть как базисные, так и составные.

В дополнение к гибкости и распреленности приложений на основе *IEC 61499*, основанное на конечных автоматах программирование функциональных блоков стандарта *IEC 61499* позволяется добиться лучшей читаемости кода и более эффективно поддерживать приложения. Часто возникает необходимость перейти от системы на основе ПЛК к эквивалентной по поведению системе нового поколения на основе *IEC 61499*. Такой переход называется *миграцией*. Отметим, что в большинстве работ по миграции предполагается наличие кода ПЛК [108—110]. Такие подходы неприменимы в тех случаях, когда исходный код недоступен, либо нет специалистов, которые могли бы быстро в нем разобраться.

С помощью разработанного в настоящей диссертации метода делается первый шаг к автоматизации процесса миграции к приложениям *IEC 61499* для тех случаев, когда другие методы неприменимы. Предлагается подход,

позволяющий восстановить ДУВ базисного функционального блока по примерам поведения – последовательностям входных/выходных событий и значений входных/выходных переменных, которые в данной главе будем называть *сценариями работы* или просто *сценариями*.

5.1. Стандарт *IEC 61499*

Стандарт *IEC 61499* предполагает разработку приложений в виде сети взаимосвязанных ФБ. Каждый ФБ характеризуется *интерфейсом*, который определяет входные/выходные события и входные/выходные переменные. Переменные могут быть, например, логическими, целочисленными, или вещественными. Также переменные могут быть ассоциированы со входными и/или выходными событиями. Такие ассоциации означают, что при обработке события ФБ запрашивает самые последние значения ассоциированных с событием переменных. Пример интерфейса ФБ приведен на рисунке 24.

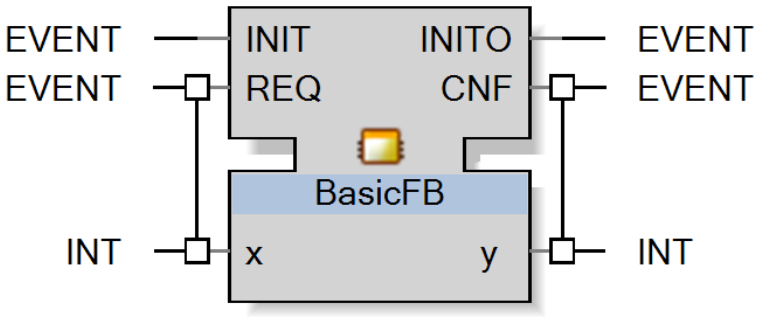

Рисунок 24 – Пример интерфейса функционального блока

Поведение *базисного* ФБ обычно задается диаграммой управления выполнением – конечным автоматом Мура специального вида. ДУВ представляет собой множество состояний, соединенных переходами. Особо выделяется *начальное состояние*, в котором ДУВ находится перед началом работы. При поступлении входного события ДУВ переходит в новое состояние в том случае, если активизируется один из переходов. Это происходит, если выполняется *охранное условие*, записанное на переходе – булева формула, которая

в общем случае может зависеть от входных, выходных и внутренних переменных, а также может содержать сравнение с константами.

Отметим, что в ДУВ может присутствовать структурный недетерминизм – в одном состоянии могут существовать два перехода по одному и тому же событию, охранные условия которых имеют общую выполняющую подстановку переменных. Такие ситуации разрешаются с помощью особой семантики выполнения, предусмотренной стандартом. Переходы проверяются в том порядке, в котором они записаны в исходном тексте программы – активизируется первый переход, для которого выполняется охранное условие.

Каждое состояние может иметь несколько ассоциированных *действий*, каждое из которых может включать выполнение *алгоритма* и генерацию выходного события. Алгоритмы в состояниях обычно реализуются на языке *Structured Text* и могут использоваться, например, для изменения значений выходных переменных. Пример ДУВ базисного ФБ приведен на рисунке 25. В верхней части интерфейса изображены входы и выходы, соответствующие событиям, а в нижней части – входы и выходы, соответствующие переменным.

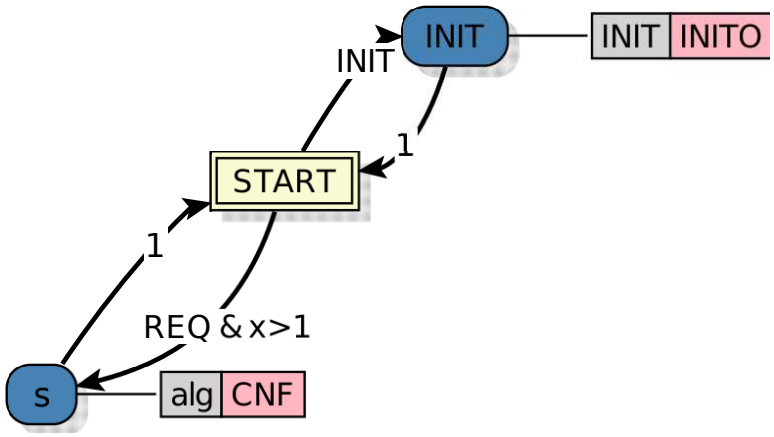

Рисунок 25 – Пример ДУВ базисного ФБ

Составной ФБ задается сетью блоков, каждый из которых может быть как базисным, так и составным. Блоки в сети соединяются связями: выходы, соответствующие выходным событиям/переменным одного ФБ соединяются

со входами, которые соответствуют входным событиям/переменным другого ФБ. Уровень вложенности ФБ не ограничен.

5.2. Постановка задачи

Будем рассматривать только базисные ФБ, все входные и выходные переменные которых являются логическими. Кроме этого, будем считать, что охранные условия на переходах ДУВ зависят только от входных переменных и не зависят от выходных и внутренних переменных, а также не содержат констант. Принимая во внимание эти упрощения, введем формальное определение диаграммы управления выполнением базисного ФБ.

Диаграммой управления выполнением (ДУВ) в данной работе будем называть семерку $\langle Y, EI, X, EO, Z, y_0, \phi, \delta \rangle$, где Y – конечное множество состояний, EI – множество входных событий, X – множество булевых входных переменных, EO – множество выходных событий, Z – множество булевых выходных переменных, $y_0 \in Y$ – начальное состояние, $\phi\colon Y \times EI \times 2^X \to Y$ – отношение переходов, а $\delta\colon Y \to \left\{\{0,1\}^{|Z|} \to \{0,1\}^{|Z|}\right\} \times EO$ – отношение действий. Для простоты будем считать, что начальное состояние всегда имеет номер ноль.

Сценарием работы ДУВ s будем называть последовательность элементов сценария s_i, каждый из которых состоит из входного события $s_i.e^{\text{in}} \in EI$, множества значений входных переменных $s_i.\text{in}$, множества значений выходных переменных $s_i.\text{out}$, и, опционально, выходного события $s_i.e^{\text{out}} \in EO$. Ниже приведен пример сценария работы для случая, когда интерфейс ФБ задает одно входное событие REQ, три входные и две выходные переменные, а также одно выходное событие CNF:

$$\langle REQ, 000, 00, CNF \rangle, \langle REQ, 001, 01, CNF \rangle, \langle REQ, 101, 11, CNF \rangle.$$

Решается задача *восстановления* неизвестной диаграммы управления выполнением базисного функционального блока с *известным интерфейсом*

по заданному набору *сценариям работы* S. Функциональный блок, диаграмму управления выполнением которого требуется восстановить, будем называть *целевым*.

5.3. Предлагаемый подход

Общая схема предлагаемого подхода восстановления ДУВ представлена на рисунке 26.

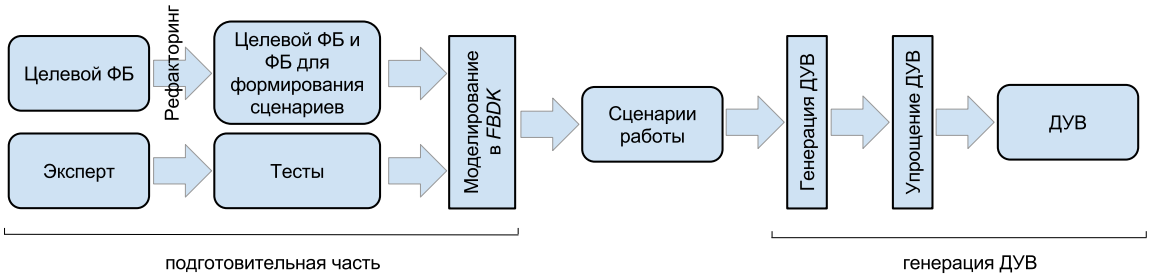

Рисунок 26 – Схема предлагаемого подхода восстановления ДУВ

Сначала для осуществления возможности формирования сценариев работы выполняется рефакторинг сети функциональных блоков, содержащей целевой ФБ. После этого путем исполнения приложения в среде разработки FBDK [128] формируется набор сценариев работы. Полученный набор используется в качестве входных данных для алгоритма $pMuACO_{ecc}$, который представляет собой предложенный в третьей главе алгоритм $pMuACO$, параметризованный набором операторов мутации для ДУВ, способом представления ДУВ и специальной функцией приспособленности. Отметим, что логика работы алгоритма $pMuACO$ при этом остается неизменной. Алгоритм генерирует ДУВ, удовлетворяющую заданным сценариям. Окончательный результат получается путем упрощения сгенерированного на предыдущем этапе решения.

5.3.1. Формирование сценариев работы

Для осуществления возможности формирования сценариев работы целевого ФБ, сеть блоков, в которой он находится, подвергается автоматизированному рефакторингу. Целевой ФБ изымается из сети и заменяется составным

ФБ под названием *ProxyLogger* с тем же интерфейсом, что и целевой ФБ. Сеть блока *ProxyLogger* состоит из трех базисных функциональных блоков – блока *InputLogger*, целевого ФБ, и блока *OutputLogger*. При поступлении очередного входного события ФБ *InputLogger* сохраняет его и текущие значения входных переменных, после чего передает их в неизменном виде целевому ФБ. Блок *OutputLogger* производит аналогичную операцию с выходным событием и значениями выходных переменных, полученными от целевого ФБ. Пример сети, реализующей составной ФБ *ProxyLogger*, приведен на рисунке 27.

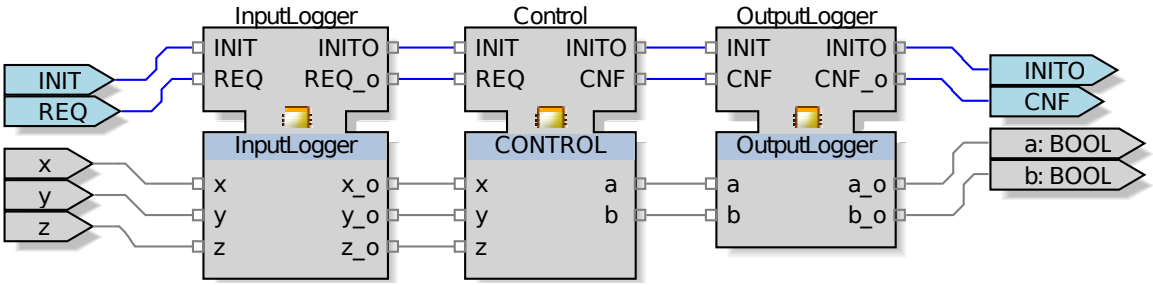

Рисунок 27 – Пример сети функционального блока *ProxyLogger*

5.3.2. Способ представления диаграммы управления выполнением

Простейший способ представления ДУВ – хранить для каждого состояния таблицу размера $|EI| \times 2^{|X|} \times 2$. В такой таблице для каждого входного события (их $|EI|$) и каждой возможной комбинации значений входных переменных (их $2^{|X|}$) хранится алгоритм (в виде строки длины $|Z|$) и выходное событие. Однако в силу того, что число входных переменных может быть довольно велико, на практике применение такого подхода затруднено.

Более оптимальным решением является применение *метода сокращенных таблиц*, предложенного в [9]. В этом подходе предполагается, что не все входные переменные необходимы для выбора нужного перехода в каждом состоянии. Необходимые переменные называются *значимыми*. Действительно, зачастую интерфейс ФБ содержит около десяти входных переменных, однако в охранных условиях на переходах используется две или три переменные. В

методе сокращенных таблиц в каждом состоянии хранится *маска значимости* входных переменных m. Если в каком-то состоянии $m_i = true$, входная переменная x_i является значимой в этом состоянии.

Однако метода сокращенных таблиц недостаточно для моделирования ДУВ, поскольку он позволяет задавать лишь такие охранные условия на переходах, в которых используются только операции конъюнкции и отрицания и не используются скобки, например, $x_1 \wedge \neg x_2 \wedge x_3$. Охранные условия, использующиеся в ДУВ, зачастую имеют более сложный вид, например, $x_1 \wedge (\neg x_2 \vee \neg x_3)$.

Для представления подобных охранных условий можно воспользоваться тем фактом, что любую булеву формулу можно представить в дизъюнктивной нормальной форме (ДНФ): $(\ldots \wedge \ldots \wedge \ldots) \vee \ldots \vee (\ldots \wedge \ldots)$. В предлагаемой модели ДУВ для каждого состояния и каждого входного события хранится набор *групп переходов*, а в каждой группе хранится своя маска значимости входных переменных. Таким образом, формула $x_1 \wedge (\neg x_2 \vee \neg x_3)$, эквивалентная формуле в ДНФ $(x_1 \wedge \neg x_2) \vee (x_1 \wedge \neg x_3)$, может быть представлена с помощью двух групп переходов: в первой группе значимыми являются переменные x_1 и x_2, а во второй группе значимы x_1 и x_3.

Итак, предлагается представлять ДУВ в виде множества состояний Y, где в каждом состоянии y_i для каждого входного события хранится множество групп переходов T_i. Для каждой группы переходов $t \in T_i$ задан логический массив, называемый маской значимости m_t и таблица переходов Φ_t из $2^{\mathrm{sum}(m_t)}$ элементов, где $\mathrm{sum}(m_t)$ – число элементов m_t, равных $true$. Каждый j-й элемент таблицы Φ_t хранит номер состояния, в которое нужно перейти, когда двоичное число, составленное из значений входных переменных, равно 2^j. Если переход отсутствует, то в ячейке таблицы хранится значение *null*.

Например, пусть в некотором состоянии для некоторого события значимы переменные x_0–x_3. Тогда в Φ_t^5 хранится состояние, в которое должна перейти ДУВ, когда $(x_0, x_1, x_2, x_3) = (0, 1, 0, 1)$, так как 0101 в двоичной си-

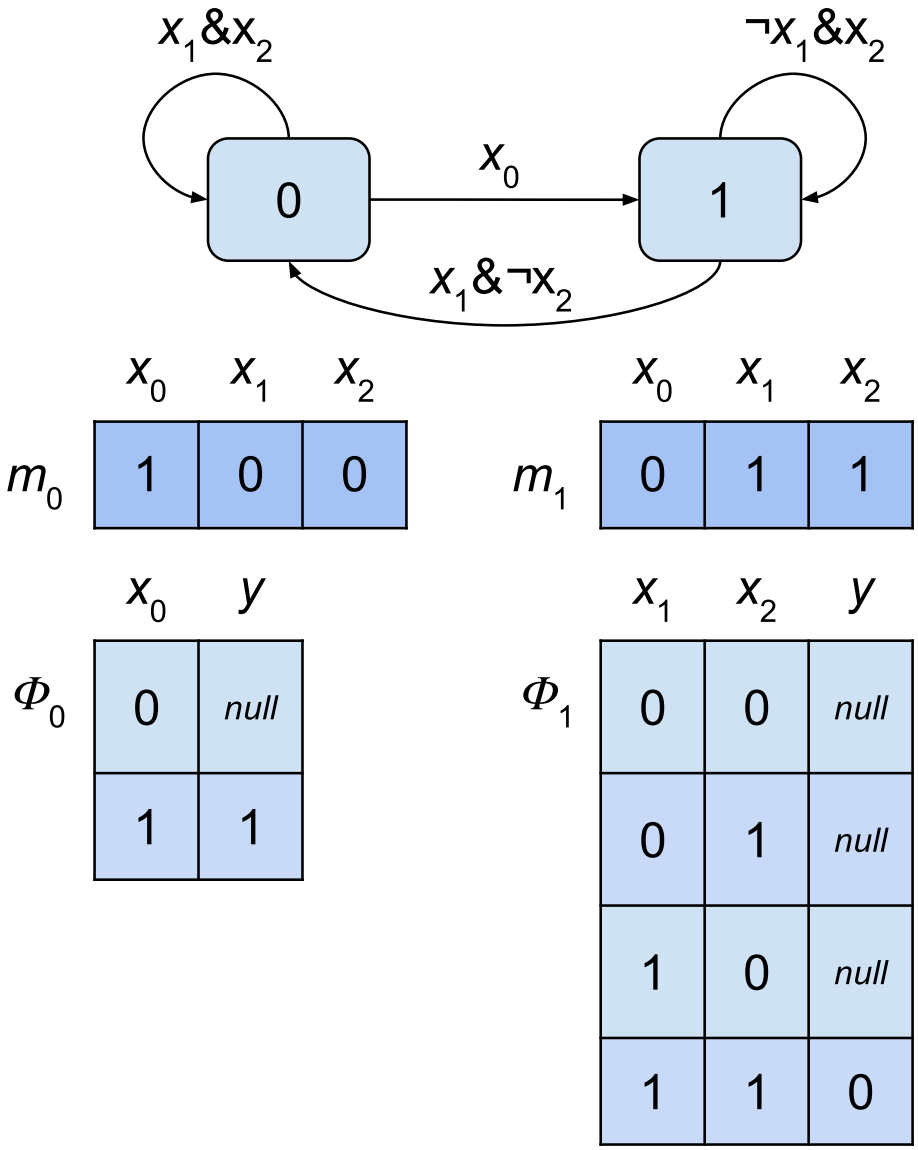

Рисунок 28 – Пример модели диаграммы управления выполнением и представление состояния 0

стеме счисления равно пяти. Пример модели ДУВ и представления состояния 0 в виде двух групп переходов приведен на рисунке 28.

Отметим, что в модели не хранятся выходные воздействия. По аналогии с подходом, предложенным в [42], они определяются исходя из сценариев работы перед каждым вычислением значения ФП.

5.3.3. Операторы мутации

В алгоритме используются следующие операторы мутации, построенные по аналогии с операторами для управляющих конечных автоматов, описанными в главе 2.

1. **Изменение состояния, в которое ведет переход**. Равномерно и случайно выбирается состояние, входное событие, группа переходов, и переход в этой группе. Состояние y, в которое ведет этот переход, изменяется на другое состояние, которое выбирается равномерно и случайно из множества $Y \setminus y$.
2. **Добавление/удаление переходов**. С определенной вероятностью изменяются группы переходов для каждого состояния. Для этого в каждом состоянии равномерно и случайно выбирается входное событие, а затем группа переходов. Равновероятно решается, добавлять или удалять переход из выбранной группы.

   Добавление перехода. Пусть в группе переходов значимыми являются k входных переменных. Тогда существует всего 2^k различных охранных условий. Из этого множества выбирается охранное условие, переход для которого не определен – в соответствующей ячейке таблицы переходов Φ_t записано значение *null*. В группу добавляется переход, помеченный упомянутым охранным условием и ведущий в состояние, выбираемое равномерно и случайно из множества всех состояний Y.

   Удаление перехода. Из группы переходов удаляется случайно выбранный переход. При этом значение соответствующей ячейки Φ_t становится равным *null*.
3. **Изменение маски значимости группы переходов**. Равномерно и случайно выбирается состояние, входное событие и группа переходов t таким образом, чтобы t содержала хотя бы один переход. В маске значимости этой группы переходов одна переменная будет сделана значимой,

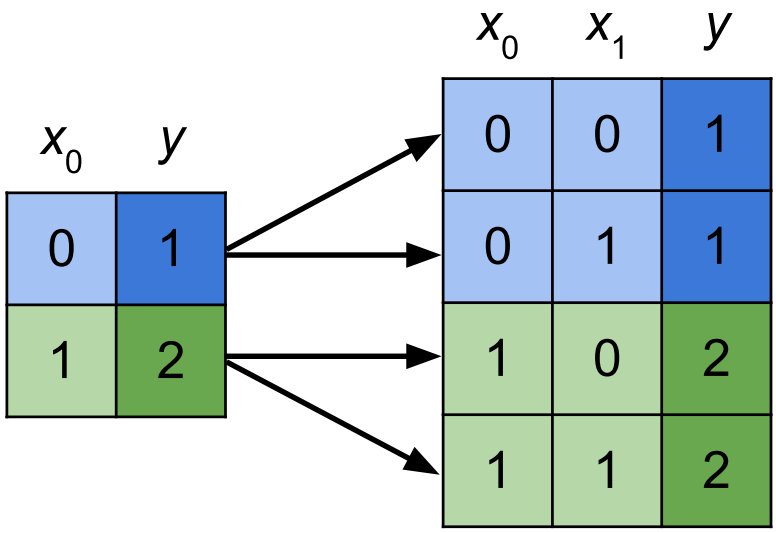

Рисунок 29 – Добавление значимой переменной

а другая – незначимой. Для этого равномерно и случайно выбираются индексы i и j такие, что значения маски значимости m_i и m_j равны $true$ и $false$ соответственно. Оператор мутации меняет значения в выбранных позициях: $m_i \leftarrow false$, $m_j \leftarrow true$.

4. **Изменение множества значимых переменных группы переходов**. Оператор мутации изменяет множество значимых переменных в случайно выбранной группе переходов. Равновероятно решается, добавить или удалить значимую переменную.

   Добавление значимой переменной. Сначала случайно выбранная незначимая переменная делается значимой: $m_i \leftarrow true$. Размер таблицы переходов выбранной группы удваивается. Для каждого старого перехода, ведущего в некое состояние y добавляются два перехода, в первом из которых $x_i = false$, а во втором $x_i = true$. Оба новых перехода ведут в состояние y. Работа этого оператора проиллюстрирована на рисунке 29. Рассмотренный оператор мутации не меняет поведения ДУВ.

   Удаление значимой переменной. Случайно выбранная значимая переменная делается незначимой: $m_i \leftarrow false$. Размер таблицы переходов выбранной группы сокращается вдвое. В данном случае необходимо выбрать, какой из каждой пары переходов оставить: с $x_i = true$ или

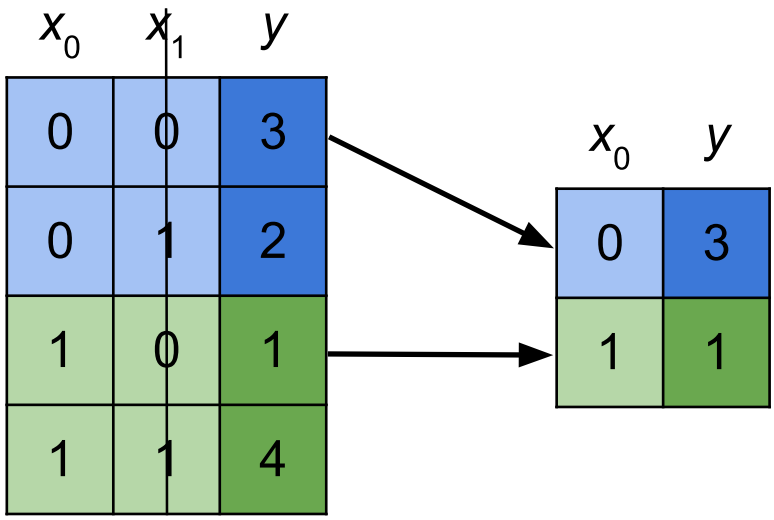

Рисунок 30 – Удаление значимой переменной

$x_i = false$. Для этого хранится счетчик добавленных переходов. Если значение счетчика делится на два, сохраняется переход с $x_i = false$, в противном случае сохраняется переход с $x_i = true$. Работа этого оператора проиллюстрирована на рисунке 30.

5.3.4. Алгоритм расстановки пометок в состояниях

Для вычисления значения ФП необходимо сначала определить алгоритмы и выходные события в состояниях. Так как все выходные переменные являются логическими, достаточно иметь один алгоритм в каждом состоянии. Алгоритмы в состояниях диаграммы управления выполнением будем далее называть *ДУВ-алгоритмами*.

В рассматриваемом случае, когда все выходные переменные являются логическими, ДУВ-алгоритмы можно представлять в виде строк длины $|Z|$ над алфавитом $\{0, 1, x\}$. Обозначим как a_i ДУВ-алгоритм, ассоциированный с состоянием y_i. Определим семантику ДУВ-алгоритмов следующим образом:

— $a_i^j = 0$: установить $z_j = 0$;

— $a_i^j = 1$: установить $z_j = 1$;

— $a_i^j = x$: сохранить текущее значение z_j.

Ниже описывается алгоритм расстановки пометок в состояниях. Для каждого состояния y будем хранить список пар строк P_y. Алгоритм последо-

вательно обрабатывает все сценарии работы. Перед обработкой очередного сценария работы ДУВ находится в начальном состоянии.

Пусть обрабатывается k-й элемент сценария ($k > 0$) и ДУВ находится в состоянии y. Сначала ДУВ выполняет переход, индуцированный событием $s_k.e^{\text{in}}$ и набором значений входных переменных $s_k.\text{in}$, и переходит в новое состояние. Далее, если k-й набор выходных переменных $s_k.\text{out}$ отличен от $s_{k-1}.\text{out}$, пара $\langle s_{k-1}.\text{out}, s_k.\text{out}\rangle$ добавляется в список P_y. После обработки всех сценариев работы логические ДУВ-алгоритмы в состояниях определяются с помощью следующего алгоритма, описанного на псевдокоде в листинге 6.

Листинг 6 – Алгоритм расстановки пометок в состояниях

```
1:  procedure EccStateLabeling
2:      for all y ∈ Y do
3:          a_y ← new char[|Z|]
4:          for i = 0 to |Z| − 1 do
5:              M ← new int[2][2]
6:
7:              for all (l, r) ∈ P_y do
8:                  M_{l_i,r_i} ← M_{l_i,r_i} + 1;
9:              end for
10:             C ← {⟨0, M_{1,0}⟩; ⟨1, M_{0,1}⟩; x, M_{0,0} + M_{1,1}⟩}
11:             a_y^i ← arg max C
12:         end for
13:     end for
14: end procedure
```

ДУВ-алгоритмы в состояниях вычисляются независимо. Сначала инициализируются строка a_y, предназначенная для хранения ДУВ-алгоритма в состоянии y. Затем начинается цикл по символам в a_y. Каждый символ

вычисляется независимо от других. Создается вспомогательный двумерный целочисленный массив M размера 2×2. Предполагается, что после создания массив заполнен нулями. В цикле сканируется список пар P_y и вычисляется, сколько раз «0» заменялся на «1», «1» на «0» и т. д. Вводится отношение C, которое используется для принятия решения о том, какой символ должен стоять на i-й позиции ДУВ-алгоритма. За $M_{1,0}$ обозначено число замен единицы на ноль, $M_{0,1}$ – число замен нуля на единицу, а $M_{0,0} + M_{1,1}$ – число случаев, когда значение i-й выходной переменной не изменилось. Наконец, i-й элемент ДУВ-алгоритма выбирается как аргумент максимального значения отношения C.

5.3.5. Функция приспособленности

Предложенная функция приспособленности состоит из трех компонентов:

$$F = c_1 F_{\text{ed}} + c_2 F_{\text{fe}} + c_3 F_{\text{sc}},$$

где компонент F_{ed} вычисляется на основе редакционного расстояния между строками, F_{fe} основан на позиции первой ошибки, которую делает решение-кандидат, F_{sc} – число смен состояний решения-кандидата в процессе вычисления значения первой компоненты ФП, а c_1, c_2 и c_3 – константы, значения которых были выбраны эвристически: $c_1 = 0,9$, $c_2 = 0,1$, $c_3 = 0,0001$. Приведенная ФП была построена по аналогии с ФП, определенной в разделе 2.1. Псевдокод алгоритма для вычисления значения ФП приведен в листинге 7.

Каждый сценарий работы обрабатывается отдельно. Перед началом обработки сценария ДУВ находится в состоянии 0. Переменная $n_{\text{stateChanges}}$ используется для подсчета числа смен состояний в процессе обработки текущего сценария. Переменная $n_{\text{firstError}}$ хранит номер элемента сценария, при обработке которого выявляется первая ошибка – значения выходных переменных не равны эталонным значениям, взятым из сценария. Переменная a используется для хранения текущих значений выходных переменных. Функ-

Листинг 7 – Функция приспособленности для ДУВ

1: **procedure** EccFitness
2: $F_{\text{ed}} \leftarrow 0$, $F_{\text{fe}} \leftarrow 0$, $F_{\text{sc}} \leftarrow 0$
3: **for all** scenarios $s \in S$ **do**
4: $y \leftarrow 0$
5: $n_{\text{stateChanges}} \leftarrow 0$
6: $n_{\text{firstError}} \leftarrow -1$
7: $a \leftarrow 0^{|Z|}$
8: $a \leftarrow \texttt{applyAlgorithm}(a, a_{\text{currentState}})$
9: outputs $\leftarrow$ new List()
10: **for** $i = 0$ to $|s| - 1$ **do**
11: $y_{\text{next}} \leftarrow I.\texttt{nextState}(y, s_i.\text{in})$
12: **if** $y_{\text{next}} \neq -1$ **then**
13: $y \leftarrow y_{\text{next}}$
14: $a \leftarrow \texttt{applyAlgorithm}(a, a_{\text{currentState}})$
15: $n_{\text{stateChanges}} \leftarrow n_{\text{stateChanges}} + 1$
16: **end if**
17: **if** $n_{\text{firstError}} == -1$ **then**
18: **if** $a \neq s_i.\text{out}$ **then**
19: $n_{\text{firstError}} \leftarrow i$
20: **end if**
21: **end if**
22: outputs.add(a)
23: **end for**
24: $F_{\text{ed}} \leftarrow F_{\text{ed}} + 1 - \frac{\texttt{ED}(\text{outputs}, s.getOutputs())}{\max(|\text{outputs}|, |s.getOutputs()|)}$
25: $F_{\text{fe}} \leftarrow F_{\text{fe}} + \frac{n_{\text{firstError}}}{|s|-1}$
26: $F_{\text{sc}} \leftarrow F_{\text{sc}} + 1 - \frac{n_{\text{stateChanges}}}{|s|}$
27: **end for**
28: **return** $\frac{1}{|S|}(c_1 F_{\text{ed}} + c_2 F_{\text{fe}} + c_3 F_{\text{sc}})$
29: **end procedure**

ция `applyAlgorithm`($\arg_1$, $\arg_2$) применяет ДУВ-алгоритм $\arg_2$ к значениям выходных переменных $\arg_1$. Например, `applyAlgorithm`$(0110, x0x1) = 0011$. Внутренний цикл перебирает элементы текущего сценария. Для каждого элемента, следующее состояние определяется с помощью функции `nextState`, которая принимает в качестве аргументов текущее состояние y и набор текущих значений входных переменных $s_i.\,\text{in}$. Если переход существует (значение $y_{\text{next}} \neq -1$), то:

— текущее состояние изменяется на новое;

— ДУВ-алгоритм, записанный в новом состоянии, применяется к текущим значениям выходных переменных;

— число смен состояний $n_{\text{stateChanges}}$ увеличивается на единицу.

Затем проверяется, не произошла ли при обработке элемента сценария первая ошибка в значениях выходных переменных.

В конце обработки каждого сценария происходит обновление значений компонентов ФП. Используемая при обновлении значения F_{ED} функция ED возвращает расстояние Левенштейна [89] между двумя списками строк. Наконец, в последней строке вычисляется окончательное значение ФП. Максимально возможное значение ФП равно $1,0001$.

Вследствие того, что длина сценариев, обычно, достаточно велика, оценка всех решений с использованием всех сценариев весьма трудозатратна. Поэтому используется иерархия сокращенных наборов сценариев. Сокращение каждого сценария выполняется путем нахождения всех последовательностей одинаковых элементов сценария и сохранения не более, чем n_{scale} таких элементов.

Оценка решения-кандидата сначала выполняется с использованием самого короткого набора сценариев. Если значение ФП не меньше единицы, решение оценивается с использованием большего набора сценариев. Схожий подход был применен в работе [49].

5.3.6. Упрощение диаграмм управления выполнением

ДУВ, удовлетворяющая всем сценариев, может быть избыточной: более простой модели может быть достаточно для описания набора сценариев. Во-первых, в ДУВ могут присутствовать переходы, которые никогда не активизируются при обработке сценариев. Во-вторых, булевы формулы на переходах могут использовать избыточное число переменных.

Для упрощения сгенерированного решения применяется следующая процедура пост-обработки. Сначала для каждого перехода проверяется, нельзя ли его удалить без уменьшения значения ФП. Далее делается попытка удалить каждую значимую входную переменную, не уменьшая при этом значение ФП. Процесс повторяется до тех пор, когда нельзя удалить ни один переход и ни одну переменную, не ухудшив значение ФП.

5.3.7. Эксперименты

Экспериментальная оценка предложенного подхода проводилась на примере одного из базисных ФБ, реализующих систему *Pick-and-Place* манипулятора (*PnP*) [111]. Для формирования сценариев работы и моделирования работы сгенерированных решений использовалась среда разработки *IEC 61499* приложений *FBDK* [128]. Общий вид системы *PnP* представлен на рисунке 31. Рассматривался вариант реализации системы *PnP*, состоящий из двух горизонтальных пневматических цилиндров (I, II), одного вертикального цилиндра (III) и присоски (IV), необходимой для захвата деталей. Когда сенсоры во входных корзинах (1, 2, 3) посылают сигнал о появлении новой детали, цилиндры перемещают присоску в нужную точку, присоска захватывает деталь, которая затем переносится на выходную ленту (V).

В рассмотренной реализации [111] вся логика управления централизована и помещена в базисный ФБ *CentralizedControl*. Этот блок получает сигналы о появлении деталей во входных корзинах и посылает команды другим ФБ, управляющим движением цилиндров и присоски. Данный ФБ был

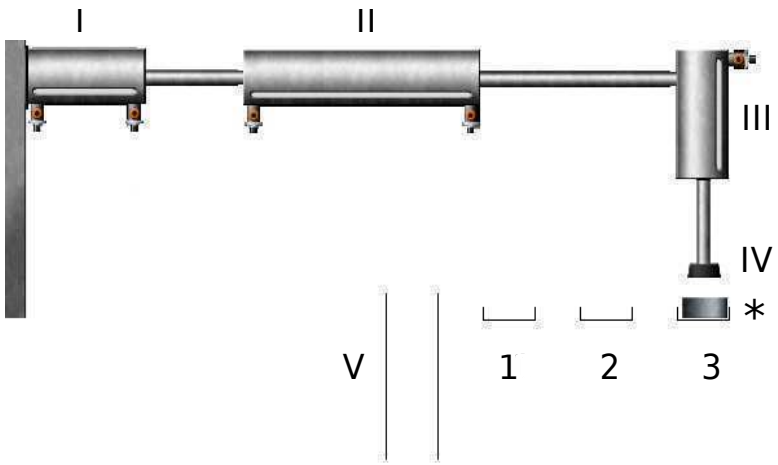

Рисунок 31 – Общий вид системы *PnP* с тремя цилиндрами

выбран потому, что в нем реализована нетривиальная логика поведения с использованием только логических входных/выходных переменных. Интерфейс ФБ *CentralizedControl* включает одно входное событие *REQ* и одно выходное событие *CNF*. Используется десять логических входных переменных:

— *c1Home*/*c1End* – горизонтальный цилиндр I находится в крайнем левом/правом положении;

— *c2Home*/*c2End* – горизонтальный цилиндр II находится в крайнем левом/правом положении;

— *vcHome*/*vcEnd* – вертикальный цилиндр III находится в крайнем верхнем/нижнем положении;

— *pp1*/*pp2*/*pp3* – во входной корзине 1/2/3 появилась деталь;

— *vac* – присоска включена.

Используется семь логических выходных переменных:

— *c1Extend*/*c1Retract* – начать двигать горизонтальный цилиндр I направо/налево;

— *c2Extend*/*c2Retract* – начать двигать горизонтальный цилиндр II направо/налево;

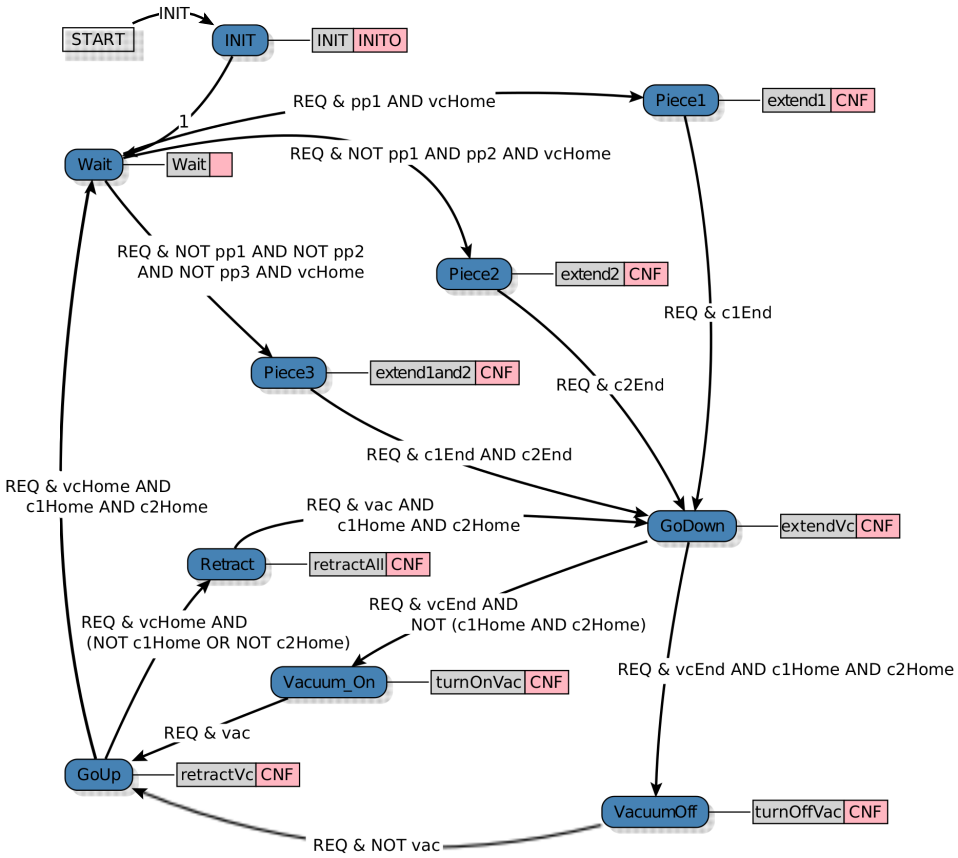

Рисунок 32 – Исходная диаграмма управления выполнением ФБ *CentralizedControl*

— *vcExtend* – начать двигать вертикальный цилиндр III вверх (предполагается, что $\neg vcExtend$ означает «начать двигать вертикальный цилиндр вниз»);

— *vacuum_on*/*vacuum_off* – включить/выключить присоску.

Сгенерированные ДУВ сравнивались с ДУВ функционального блока *CentralizedControl*, взятой из проекта *PnP* [111]. Эту диаграмму, приведенную на рисунке 32, в дальнейшем будем называть *исходной*. Она содержит девять состояний (без учета стандартных состояний *Init* и *Start*) и 15 переходов. Отметим, что исходная диаграмма ни коим образом не используется в предложенном подходе и приведена только для сравнения.

5.3.7.1. Генерация диаграмм выполнения управлением

Эксперимент выполнялся следующим образом.

1. Формирование сценариев работы.

2. Генерация ДУВ с помощью предложенного подхода.
3. Проверка корректности работы сгенерированных ДУВ путем моделирования в среде *FBDK*.

Было записано десять сценариев работы, каждый из которых определяется *тестом* – последовательностью деталей, которые должна обработать система. Например, тест «1-2-3» задает сценарий, в котором система сначала обрабатывает деталь в первой, во второй, и, наконец, в третьей корзине. Были записаны сценарии по следующим тестам: «1», «1-2», «1-2-3», «2», «2-1», «2-3», «3», «3-2», «3-2-1», а также тест «123», в котором все три детали помещаются во входные корзины одновременно.

В ФП использовался полный набор сценариев и два сокращенных набора сценариев: с $n_{\text{scale}} = 3$ и $n_{\text{scale}} = 20$. Суммарная длина трех наборов сценариев, измеренная в числе элементов сценариев, равнялась 1580, 5383 и 30302 соответственно. Алгоритм искал решение, содержащее не более десяти состояний, в каждом состоянии могло быть не более четырех групп переходов для каждого события. Использовался компьютер с 64-ядерным процессором *AMD Opteron(TM) 6378* @ 2.4 ГГц, параллельный муравьиный алгоритм $pMuACO_{\text{ecc}}$ использовал 16 потоков.

Эксперимент был повторен 20 раз. Среднее время генерации автомата, удовлетворяющего всем сценариям, составило около 4,5 часов. Все сгенерированные ДУВ были проверены путем моделирования в среде *FBDK*. Для этого диаграмма автоматически конвертировалась в *XML*-формат, поддерживаемый *FBDK*. В процессе моделирования было установлено, что все сгенерированные ДУВ корректно обрабатывают все тесты. Одна из построенных алгоритмом диаграмм приведена на рисунке 33.

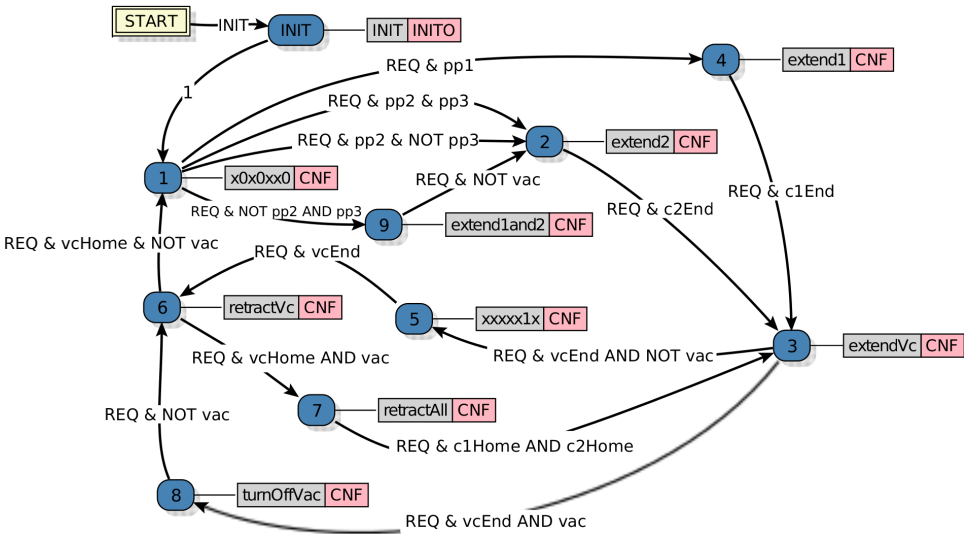

Рисунок 33 – Одна из сгенерированных диаграмм управления выполнением

5.3.8. Анализ сгенерированных ДУВ

Исходная ДУВ содержит девять состояний, 15 переходов и использует в охранных условиях 32 значимые переменные. При обработке полного набора сценариев делается 180 смен состояний.

Одна из сгенерированных с помощью предложенного подхода ДУВ содержит девять состояний, а остальные – 10 состояний. Сгенерированные ДУВ:

— содержат от 14 до 18 переходов (в среднем, 16):

— используют в охранных условиях от 18 до 35 значимых переменных (в среднем, 25);

— совершают от 180 до 9476 смен состояний при обработке сценариев (в среднем, 2581).

Ни одна из сгенерированных ДУВ не изоморфна исходной ДУВ. Из этого следует, что хотя все полученные решения обладают поведением, идентичным поведению исходной ДУВ, логика их работы существенно отлична от логики работы исходной ДУВ.

Однако заметим, что сгенерированные ДУВ и исходная ДУВ имеют некоторые сходства. Так, все построенные решения имеют семь общих с исходной диаграммой алгоритмов:

1. «*extend1*» (1xxxxxx);
2. «*extend2*» (xx1xxxx);
3. «*extend1and2*» (1x1xxxx);
4. «*extendVc*» (xxxx1xx);
5. «*retractVc*» (xxxx0xx);
6. «*retractAll*» (0101xxx);
7. «*turnOffVac*» (xxxxx01).

При этом ни в одной из построенных ДУВ не используются следующие два алгоритма из исходной ДУВ.

1. Алгоритм «*Wait*» (0000000) – назначает всем выходным переменным значение *false*. Вместо этого алгоритма в сгенерированных ДУВ используется алгоритм (x0x0xx0), поскольку этого достаточно для правильной обработки заданных сценариев.
2. Алгоритм «*turnOnVac*» (xxxxx10). Вместо него в построенных ДУВ применяется аналогичный алгоритм (xxxxx1x), отличающийся тем, что значение переменной *vacuum_ off* не устанавливается равным *false*.

Наконец, алгоритм упрощения ДУВ был применен к исходной диаграмме. Полученная упрощенная исходная ДУВ содержит 14 состояний (вместо 15 у исходной) и использует в охранных условиях только 20 значимых переменных (вместо 32 в исходной). Было установлено, что все сгенерированные ДУВ используют тот же набор алгоритмов, что и упрощенная исходная ДУВ. При этом ни одна из построенных ДУВ не изоморфна упрощенной исходной ДУВ.

Из полученных результатов можно сделать вывод о том, что предложенный подход позволяет генерировать диаграммы, достаточно похожие на исходую ДУВ, и обладающие при этом эквивалентным с ней поведением на

заданном наборе сценариев. Для получения ДУВ, более похожих на исходную, следует увеличить набор тестов и сценариев.

5.4. Полиномиальный алгоритм генерации ДУВ специального вида

Описанный в предыдущем разделе подход на основе предложенного в диссертации метаэвристического алгоритма является достаточно общим: он допускает любое распределение ДУВ-алгоритмов по состояниям ДУВ, а также может быть относительно легко расширен для поддержки других типов входных/выходных переменных.

Однако в частном случае, когда все выходные переменные являются логическими и каждый алгоритм используется *ровно в одном* состоянии ДУВ, построение ДУВ может быть осуществлено за полиномиальное от размера сценариев время. Основная идея заключается в том, что в таком случае можно автоматически определить близкое к минимальному множество алгоритмов, необходимых для описания заданных сценариев работы. Опишем полиномиальный алгоритм построения ДУВ, предложенный диссертантом в [140]. Алгоритм состоит из следующих основных шагов.

1. Вычисление множества ДУВ-алгоритмов.
2. Построение ДУВ по сценариям и множеству ДУВ-алгоритмов.
3. Упрощение ДУВ.

5.4.1. Вычисление множества ДУВ-алгоритмов

Обозначим за A множество ДУВ-алгоритмов, изначально оно пусто. Сначала для каждого сценария s и каждых двух последовательных элементов сценария s_i и s_{i+1} в A добавляется ДУВ-алгоритм a, трансформирующий $s_i.\,\mathrm{out}$ в $s_{i+1}.\,\mathrm{out}$. Каждый элемент a_j этого алгоритма a вычисляется с по-

мощью функции $\mathrm{calcAlg}(s_i, s_{i+1})$, работающей следущим образом:

$$a_j = \begin{cases} x, & \text{если } s_i^j = s_{i+1}^j; \\ 0, & \text{если } s_i^j = 1 \wedge s_{i+1}^j = 0; \\ 1, & \text{если } s_i^j = 0 \wedge s_{i+1}^j = 1. \end{cases}$$

Далее A минимизируется жадным образом путем последовательного объединения каждой пары ДУВ-алгоритмов, если это возможно. Поддерживается инвариант – с помощью текущего множества ДУВ-алгоритмов A всегда можно описать сценарии работы. А именно: для каждых двух последовательных элементов сценария s_i и s_{i+1} существует ДУВ-алгоритм $a \in A$, такой что $\mathrm{applyAlg}(s_i.\,\mathrm{out}, a) = s_{i+1}.\,\mathrm{out}$. Для начального множества ДУВ-алгоритмов инвариант выполняется по построению.

Перед объединением ДУВ-алгоритмов a и b проверяется, не являются ли они *противоречивыми*. Алгоритмы a и b считаются противоречивыми, если они противоречивы хотя бы в одной позиции:

$$\exists i : (a_i = 0 \wedge b_i = 1) \vee (a_i = 1 \wedge b_i = 0)\,.$$

Например, алгоритмы $x01$ и $10x$ непротиворечивы (результат объединения равен $x0x$), а алгоритмы $a = x00$ и $b = x10$ противоречивы, так как $a_1 = 0$ и $b_1 = 1$. Противоречивые алгоритмы не объединяются.

Каждый элемент объединения m^{ab} алгоритмов a и b вычисляется с помощью функции $\mathrm{merge}(a, b)$, работающей в соответствии со следующей формулой:

$$m_i^{ab} = \begin{cases} a_i, & \text{если } a_i = b_i; \\ x, & \text{если } a_i = x \vee b_i = x. \end{cases} \tag{5.1}$$

После объединения алгоритмы a и b удаляются из множества A, а вместо них в A добавляется объединенный алгоритм m^{ab}. Далее необходимо проверить выполнение инварианта. Для этого проверяются все пары последовательных элементов сценария s_i и s_{i+1}, в которых для трансформации $s_i.\,\mathrm{out}$

в $s_{i+1}.\,\text{out}$ используется алгоритм a или алгоритм b. Если для всех таких пар использование алгоритма m^{ab} дает тот же результат, что и использование алгоритма a (b), то объединение принимается. В противном случае объединение отклоняется, m^{ab} удаляется из A, а a и b снова добавляются в A.

Процесс перезапускается после каждого удачного объединения алгоритмов и повторяется до момента, когда нельзя сделать ни одного объединения. Трудоемкость описанного алгоритма составляет $O(N + N\hat{A}^3) = O(N\hat{A}^3)$, где $N = |S|$ – суммарное число элементов сценариев, а $\hat{A}$ – изначальная мощность множества A. Так как в худшем случае $\hat{A} = 3^{|Z|}$, итоговая трудоемкость составляет $O(N \cdot 3^{|Z|^3})$.

Псевдокод алгоритма представлен в листинге 8. Можно заметить, что так как изначальное множество ДУВ-алгоритмов конечно и каждое объединение сокращает размер множества, то алгоритм в какой-то момент завершится.

5.4.2. Построение ДУВ по сценариям и известному множеству алгоритмов

С помощью найденного множества ДУВ-алгоритмов A можно построить ДУВ по набору сценариев. Пусть A_{used} – список использованных алгоритмов и пусть $\{\tau_i\}_{i=0}^{|A|-1}$ – список списков переходов ДУВ. Сначала определим алгоритм $a_0 \in A$, совместимый с первым элементом всех заданных сценариев – такой, что

$$\forall s \in S : \text{applyAlg}(s_0.\,\text{out}, a_0) = s_1.\,\text{out}\,.$$

Алгоритм a_0 добавляется в A_{used}.

Далее отдельно обрабатывается каждый сценарий. Выберем значение начального состояния $y_{\text{current}} = 0$. На каждом шаге рассматриваются два последовательных элемента сценария s_i и s_{i+1}. Если $s_i.\,\text{out} = s_{i+1}.\,\text{out}$, то значение i увеличивается на единицу и рассматривается следущая пара элементов

Листинг 8 – Вычисление множества ДУВ-алгоритмов

```
1: procedure GetEccAlgorithmSet(S)
2:     A ← new Set()
3:     for all scenarios s ∈ S do
4:         for i = 0 to |s| − 1 do
5:             A ← A ∪ {calcAlg(s_i. out, s_{i+1}. out)}
6:         end for
7:     end for
8:     while true do
9:         changed ← false
10:
11:        for all a ∈ A do
12:            for all b ∈ A, b ≠ a do
13:                m^{ab} ← merge(a, b)
14:                if объединение допустимо then
15:                    A ← A \ {a, b}
16:                    A ← A ∪ {m^{ab}}
17:                    changed ← true
18:                    goto line 22
19:                end if
20:            end for
21:        end for
22:        if ¬ changed then
23:            break
24:        end if
25:    end while
26: return A
27: end procedure
```

Листинг 9 – Построение ДУВ по сценариям и множеству алгоритмов

procedure ConstructECC(S, A)
  $A_{\text{used}} \leftarrow \{a \in A : \forall s \in S \text{ applyAlg}(s_0.\text{out}, a) = s_1.\text{out}\}$
  for all $s \in S$ **do**
    for $i = 0$ to $|s| - 1$ **do**
      $y_{\text{current}} \leftarrow 0$
      if $s_i.\text{out} == s_{i+1}.\text{out}$ **then**
        continue
      end if
      $a \leftarrow \text{getBestMatch}(s_i.\text{out}, s_{i+1}.\text{out}, A)$
      $y_{\text{new}} \leftarrow -1$
      if $a \in A_{\text{used}}$ **then**
        $y_{\text{new}} \leftarrow A.\text{indexof}(a)$
      else
        $A_{\text{used}} \leftarrow A_{\text{used}} \cup \{a\}$
        $y_{\text{new}} \leftarrow |A_{\text{used}}| - 1$
      end if
      t $\leftarrow$ new Transition($s_{i+1}.e^{\text{in}}$, $s_{i+1}.\text{in}$, y_{new})
      if $t \notin \tau_{y_{\text{current}}}$ **then**
        $\tau_{y_{\text{current}}} \leftarrow \tau_{y_{\text{current}}} \cup \{t\}$
      end if
      if недетерминированное поведение **then**
        print "Недетерминированное поведение"
        exit
      end if
      $y_{\text{current}} \leftarrow y_{\text{new}}$
    end for
  end for
end procedure

сценария. Если же $s_i.\,\mathrm{out} \neq s_{i+1}.\,\mathrm{out}$, то из множества алгоритмов A выбирается алгоритм a такой, что $\mathrm{applyAlg}(s_i.\,\mathrm{out}, a) = s_{i+1}.\,\mathrm{out}$.

Если $a \in A_{\mathrm{used}}$, то новое состояние y_{new} вычисляется как позиция элемента a в списке A_{used}. В противном случае a добавляется в множество A_{used} и $y_{\mathrm{new}} = |A_{\mathrm{used}}| - 1$.

Затем в список переходов $\tau_{y_{\mathrm{current}}}$ из состояния y_{current} добавляется новый переход, помеченный входным событием $s_{i+1}.e^{\mathrm{in}}$, кортежем значений входных переменных $s_{i+1}.\,\mathrm{in}$, и состоянием y_{new}. При этом проверяется, что если $\tau_{y_{\mathrm{current}}}$ уже содержит переходы, помеченные событием $s_{i+1}.e^{\mathrm{in}}$ и кортежем $s_{i+1}.\,\mathrm{in}$, то все такие переходы должны вести в одно и то же состояние y_{new}. Если это не так, то алгоритм завершается, выдав сообщение о том, что в сценариях содержится недетерминированное поведение. Наконец, обновляется значение текущего состояния: $y_{\mathrm{current}} \leftarrow y_{\mathrm{new}}$. После обработки всех сценариев состояниям ДУВ назначаются соответствующие алгоритмы и выходные события. Трудоемкость построения ДУВ составляет $O(N)$. Псевдокод алгоритма приведен в листинге 9. Время, необходимое описанному алгоритму для построения ДУВ для примера, описанного в разделе 5.3.7, не превышает одной минуты при запуске на персональном компьютере с процессором *AMD Phenom(tm) II X4 955* @ 3,2 ГГц.

Выводы по главе 5

1. Предложенный метод генерации конечных автоматов *pMuACO* был использован при генерации ДУВ для базисных ФБ стандарта *IEC 61499* в случае логических входных/выходных переменных. Были получены ДУВ, достаточно близкие к исходной и обладающие эквивалентным с ней поведением на заданном наборе сценариев. Представленный подход может быть расширен для учета входных/выходных переменных других типов.

2. Для частного случая, когда каждый алгоритм используется ровно в одном состоянии ДУВ и все входные/выходные переменные являются логическими, был предложен полиномиальный алгоритм генерации ДУВ.
3. Результаты, описанные в данной главе, опубликованы в трудах международных конференций «IEEE International Conference on Industrial Informatics» [139] и «IEEE International Symposium on Parallel and Distributed Processing with Applications» [140].

ЗАКЛЮЧЕНИЕ

В диссертации получены следующие результаты.

1. Предложен метод *MuACO* генерации конечных автоматов по сценариям работы и темпоральным формулам, основанный на муравьином алгоритме с предложенным автором графом мутаций. Показано, что метод работает быстрее известных методов на основе генетических алгоритмов и *AE*-парадигмы.
2. Предложены методы *pMuACO*, *psMuACO* и *pstMuACO* генерации конечных автоматов по сценариям работы и темпоральным формулам на основе параллельных муравьиных алгоритмов. Последний из них, *pstMuACO*, совмещает параллельный муравьиный алгоритм *pMuACO*, точный метод *efsmSAT* генерации конечных автоматов по сценариям работы на основе сведения к задаче выполнимости булевой формулы и процедуру прореживания сценариев. Показано, что метод *pstMuACO* работает быстрее параллельного генетического алгоритма, а также методов *pMuACO* и *psMuACO*.
3. Разработано программное средство *muaco.jar*, реализующее предложенные методы, исходный код которого размещен в открытом доступе в сети Интернет. Средство позволяет генерировать автоматы по сценариям работы и темпоральным формулам, а также может быть использовано для решения других задач генерации автоматов.
4. Результаты диссертации были внедрены в Технологическом университете Лулео (Швеция) при генерации автоматной логики для базисных функциональных блоков стандарта *IEC 61499* [139, 140], имеется акт внедрения. Также результаты были использованы в учебном процессе на кафедре «Компьютерные технологии» Университета ИТМО в рамках курсов «Теория автоматов и программирование» и «Генетическое программирование», имеется акт использования. Наконец, некоторые

результаты диссертации были использованы при генерации конечных автоматов для управления моделью беспилотного самолета [131].

Полученные результаты могут быть использованы для решения задач генерации конечных автоматов. В качестве направлений дальнейшей разработки темы диссертации можно выделить доработку предложенных методов для генерации диаграмм управления выполнением базисных функциональных блоков стандарта *IEC 61499* с учетом темпоральных формул, а также целочисленных и вещественных входных/выходных переменных.

СПИСОК ИСТОЧНИКОВ

Печатные издания на русском языке

1. *Поликарпова Н. И.*, *Шалыто А. А.* Автоматное программирование. — СПб : Питер, 2009. — 176 с.

2. *Шалыто А. А.* Использование граф-схем и графов переходов при программной реализации алгоритмов логического управления. II // АиТ. — 1996. — № 7. — С. 144—169.

3. *Шалыто А. А.*, *Туккель Н. И.* SWITCH-технология: автоматный подход к созданию программного обеспечения «реактивных» систем // Программирование. — 2001. — № 5. — С. 45—62.

4. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. — СПб : Наука, 1998. — 628 с.

5. Верификация автоматных программ / С. Э. Вельдер, М. А. Лукин, А. А. Шалыто, Б. Р. Яминов. — СПб : Наука, 2011. — 244 с.

6. *Лобанов П. Г.*, *Шалыто А. А.* Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о флибах // Известия РАН. Теория и системы управления. — 2007. — № 5. — С. 127—136.

7. *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. Т. 2. — СПбГЭТУ «ЛЭТИ», 2007. — С. 88—91.

8. *Лобанов П. Г.* Использование генетических алгоритмов для генерации конечных автоматов. — 2008. — Диссертация на соискание ученой степени кандидата технических наук. СПбГУ ИТМО.

9. *Поликарпова Н. И.*, *Точилин В. Н.*, *Шалыто А. А.* Метод сокращенных таблиц для генерации автоматов с большим числом входных переменных на основе генетического программирования // Известия РАН. Теория и системы управления. — 2010. — № 2. — С. 100—117.

10. *Царев Ф. Н.*, *Егоров К. В.*, *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. — 2010. — 5(69). — С. 81—86.

11. *Царев Ф. Н.* Методы построения конечных автоматов на основе эволюционных алгоритмов. — 2012. — Диссертация на соискание ученой степени кандидата технических наук. НИУ ИТМО.

12. *Егоров К. В.* Генерация управляющих автоматов на основе генетического программирования и верификации. — 2013. — Диссертация на соискание ученой степени кандидата технических наук. НИУ ИТМО.

13. *Карпов Ю. Г.* Model Checking. Верификация параллельных и распределенных программных систем. — СПб. : БХВ-Петербург, 2010. — 560 с.

14. *Скобцов Ю. А.*, *Федоров Е. Е.* Метаэвристики. — Донецк : Ноулидж, 2013. — 426 с.

15. Биоинспирированные методы в оптимизации / Л. А. Гладков, В. В. Курейчик, В. М. Курейчик, П. В. Сороколетов. — М. : Физматлит, 2010. — 384 с.

16. *Емельянов В. В.*, *Курейчик В. В.*, *Курейчик В. М.* Теория и практика эволюционного моделирования. — М. : Физматлит, 2003. — 432 с.

17. *Курейчик В. М.*, *Кажаров А. А.* Применение пчелиного алгоритма для раскраски графов // Известия ЮФУ. Технические науки. — 2010. — Т. 113, № 12. — С. 30—36.

18. *Николенко С. И.*, *Тулупьев А. Л.* Самообучающиеся системы. — М. : МЦНМО, 2009. — 288 с.

19. *Кажаров А. А.*, *Курейчик В. М.* Муравьиные алгоритмы для решения транспортных задач // Известия РАН. Теория и системы управления. — 2010. — № 1. — С. 32—45.

20. *Курейчик В. М.*, *Лебедев Б. К.*, *Лебедев О. Б.* Гибридный алгоритм разбиения на основе природных механизмов принятия решений // Искусственный интеллект и принятие решений. — 2012. — № 2. — С. 3—15.

21. *Фогель Л.*, *Оуэнс А.*, *Уолш М.* Искусственный интеллект и эволюционное моделирование. — Мир, 1969. — 230 с.

22. *Ахи А. А.*, *Станкевич А. С.*, *Шалыто А. А.* Алгоритм построения флибов со 100%-ной точностью предсказания // Информационные технологии. — 2011. — № 7. — С. 34—37.

23. *Данилов В. Р.* Метод представления автоматов деревьями решений для использования в генетическом программировании // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. — 2011. — Т. 53, № 8. — С. 103—107.

24. *Данилов В. Р.*, *Шалыто А. А.* Метод представления автоматов линейными бинарными графами для использования в генетическом программировании // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. — 2011. — Т. 72, № 2. — С. 54—57.

25. *Чивилихин Д. С.* Метод построения конечных автоматов на основе муравьиного алгоритма. — 2013. — Магистерская диссертация. НИУ ИТМО.

Печатные издания на английском языке

26. *Clarke E. M.*, *Grumberg O.*, *Peled D. A.* Model checking. — MIT press, 1999. — 330 p.

27. The Genesys System: Evolution as a Theme in Artificial Life / D. Jefferson, R. Collins, C. Cooper, M. Dyer, M. Flowers, R. Korf, C. Taylor, A. Wang // Proceedings of Second Conference on Artificial Life. — Addison Wesley, 1992. — P. 549–578.

28. *Harman M.*, *Mansouri S. A.*, *Zhang Y.* Search Based Software Engineering: A Comprehensive Analysis and Review of Trends, Technologies and Applications: tech. rep. / Department of Computer Science, King's College London. — 2009.

29. *McMinn P.* Search-based Software Test Data Generation: A Survey: Research Articles // Software Testing, Verification & Reliability. — Chichester, UK, 2004. — Vol. 14, no. 2. — P. 105–156.

30. *Greer D.*, *Ruhe G.* Software release planning: an evolutionary and iterative approach // Information and Software Technology. — 2004. — Vol. 46. — P. 243–253.

31. *Antoniol G.*, *Di Penta M.*, *Harman M.* Search-based techniques applied to optimization of project planning for a massive maintenance project // Proceedings of the 21st IEEE International Conference on Software Maintenance. — 2005. — P. 240–249.

32. *Alba E.*, *Chicano F.* Software Project Management with GAs // Information Sciences. — New York, NY, USA, 2007. — Vol. 177, no. 11. — P. 2380–2401.

33. Handbook of Evolutionary Computation / ed. by T. Back, D. B. Fogel, Z. Michalewicz. — Bristol, UK : IOP Publishing Ltd., 1997. — 1130 p.

34. *Mitchell M.* An Introduction to Genetic Algorithms. — MIT Press, 1996. — 209 p.

35. *Beyer H.-G.*, *Schwefel H.-P.* Evolution strategies – A comprehensive introduction // Natural Computing. — 2002. — Vol. 1, no. 1. — P. 3–52.

36. *Dorigo M.*, *Maniezzo V.*, *Colorni A.* Ant System: optimization by a colony of cooperating agents // IEEE Transactions on Systems, Man and Cybernetics. — 1996. — Vol. 26, no. 1. — P. 29–41.

37. *Dorigo M.*, *Stützle T.* Ant Colony Optimization. — MIT Press, 2004. — 319 p.

38. *Dorigo M.*, *Gambardella L.* Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem // IEEE Transactions on Evolutionary Computation. — 1997. — Vol. 1, no. 1. — P. 53–66.

39. *Stützle T.*, *Hoos H.* MAX MIN Ant System // Future Generation Computer Systems. — 2000. — Vol. 16. — P. 889–914.

40. *Bullnheimer B.*, *Hartl R. F.*, *Strauss C.* A new rank-based version of the Ant System: A computational study // Central European Journal for Operations Research and Economics. — 1999. — Vol. 7, no. 1. — P. 25–38.

41. *Lucas S.*, *Reynolds J.* Learning Finite State Transducers: Evolution versus Heuristic State Merging // IEEE Transactions on Evolutionary Computation. — 2007. — Vol. 11, no. 3. — P. 308–325.

42. *Lucas S.*, *Reynolds 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.

43. *Lang K. J.*, *Pearlmutter B. A.*, *Price R. A.* Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm // Grammatical Inference. — Springer, 1998. — P. 1–12. — (Lecture Notes in Computer Science ; 1433).

44. *Heule M.*, *Verwer S.* Exact DFA Identification Using SAT Solvers // Proceedings of the 10th International Colloquium Conference on Grammatical Inference: Theoretical Results and Applications. — Berlin, Heidelberg : Springer-Verlag, 2010. — P. 66–79.

45. *Ulyantsev V.*, *Zakirzyanov I.*, *Shalyto A.* BFS-Based Symmetry Breaking Predicates for DFA Identification // Language and Automata Theory and Applications. — Springer International Publishing, 2015. — P. 611–622. — (Lecture Notes in Computer Science ; 8977).

46. *Ulyantsev V.*, *Tsarev F.* Extended Finite-State Machine Induction Using SAT-Solver // Proceedings of the 10th International Conference on Machine Learning and Applications. Vol. 2. — Los Alamitos, CA, USA : IEEE Computer Society, 2011. — P. 346–349.

47. *Heule M.*, *Verwer S.* Software model synthesis using satisfiability solvers // Empirical Software Engineering. — 2013. — Vol. 18, no. 4. — P. 825–856.

48. *Chellapilla K.*, *Czarnecki D.* A preliminary investigation into evolving modular finite state machines // Proceedings of the 1999 Congress on Evolutionary Computation. Vol. 2. — 1999. — P. 1349–1356.

49. *Spears W.*, *Gordon D.* Evolving finite-state machine strategies for protecting resources // Proceedings of the International Symposium on Methodologies for Intelligent Systems. — 2000. — P. 166–175.

50. *Johnson C.* Genetic Programming with Fitness Based on Model Checking // Genetic Programming. — Springer Berlin Heidelberg, 2007. — P. 114–124. — (Lecture Notes in Computer Science ; 4445).

51. *Walkinshaw N.*, *Bogdanov K.* Inferring Finite-State Models with Temporal Constraints // Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering. — Washington, DC, USA : IEEE Computer Society, 2008. — P. 248–257.

52. *Tsarev F.*, *Egorov K.* Finite-state machine induction using genetic algorithm based on testing and model checking // Proceedings of Genetic and Evolutionary Computation Conference companion. — 2011. — P. 759–762.

53. *Rosner R.* Modular Synthesis of Reactive Systems. — Wezimann Institute of Science, 1992. — PhD thesis.

54. *Pnueli A.*, *Rosner R.* On the Synthesis of a Reactive Module // Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — New York, NY, USA : ACM, 1989. — P. 179–190.

55. *Jobstmann B.*, *Bloem R.* Optimizations for LTL Synthesis // Formal Methods in Computer Aided Design. — 2006. — P. 117–124.

56. *Ehlers R.* Symbolic bounded synthesis // Formal Methods in System Design. — 2012. — Vol. 40, no. 2. — P. 232–262.

57. G4LTL-ST: Automatic Generation of PLC Programs / C.-H. Cheng, C.-H. Huang, H. Ruess, S. Stattelmann // Computer Aided Verification. — 2014. — P. 541–549. — (Lecture Notes in Computer Science ; 8559).

58. *Vyatkin V.* IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design. — ISA, 2012. — 259 p.

59. *Vyatkin V.*, *Hanisch H.-M.* Verification of distributed control systems in intelligent manufacturing // Journal of Intelligent Manufacturing. — 2003. — Vol. 14, no. 1. — P. 123–136.

60. *Dubinin V.*, *Vyatkin V.*, *Hanisch H.-M.* Modelling and Verification of IEC 61499 Applications using Prolog // IEEE Conference on Emerging Technologies and Factory Automation. — 2006. — P. 774–781.

61. *Blum C.*, *Roli A.* Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison // ACM Comput. Surv. — New York, NY, USA, 2003. — Vol. 35, no. 3. — P. 268–308.

62. *Kirkpatrick S.*, *Gelatt C. D.*, *Vecchi M. P.* Optimization by Simulated Annealing // Science. — 1983. — Vol. 220, no. 4598. — P. 671–680.

63. *Glover F.* Future Paths for Integer Programming and Links to Artificial Intelligence // Computers & Operations Research. — Oxford, UK, 1986. — Vol. 13, no. 5. — P. 533–549.

64. *Kennedy J.*, *Eberhart R.* Particle swarm optimization // Proceedings of the IEEE International Conference on Neural Networks. Vol. 4. — 1995. — P. 1942–1948.

65. The Bees Algorithm: tech. rep. / D. T. Pham, S. Otri, E. Koc, A. Ghanbarzadeh, S. Rahim, M. Zaidi ; Manufacturing Engineering Centre, Cardiff University, Cardiff, UK. — 2005.

66. *Yang X.-S.* Nature-Inspired Metaheuristic Algorithms. — Luniver Press, 2008. — 148 p.

67. *Karaboga D.* An Idea Based On Honey Bee Swarm for Numerical Optimization: tech. rep. / Erciyes University, Engineering Faculty, Computer Engineering Department. — 2005. — Technical Report-TR06.

68. *Kephart J.* A biologically inspired immune system for computers // Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems. — 1994. — P. 130–139.

69. *Koza J.* Genetic Programming: On the Programming of Computers by Natural Selection. — MIT Press, 1992. — 836 p.

70. *Storn R.*, *Price K.* Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces // Journal of Global Optimization. — 1997. — Vol. 11, no. 4. — P. 341–359.

71. *Katz G.*, *Peled D.* Model Checking-Based Genetic Programming with an Application to Mutual Exclusion // Tools and Algorithms for the Construction and Analysis of Systems. — Springer Berlin Heidelberg, 2008. — P. 141–156. — (Lecture Notes in Computer Science ; 4963).

72. *Nolfi S.*, *Floreano D.* Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines. — MIT press, 2000. — 332 p.

73. *Hornby G. S.*, *Lohn J. D.*, *Linden D. S.* Computer-automated Evolution of an X-band Antenna for Nasa's Space Technology 5 Mission // Evolutionary Computation. — Cambridge, MA, USA, 2011. — Vol. 19, no. 1. — P. 1–23.

74. *Bräysy O.*, *Dullaert W.*, *Gendreau M.* Evolutionary Algorithms for the Vehicle Routing Problem with Time Windows // Journal of Heuristics. — 2004. — Vol. 10, no. 6. — P. 587–611.

75. The self-organizing exploratory pattern of the argentine ant / J.-L. Deneubourg, S. Aron, S. Goss, J. Pasteels // Journal of Insect Behavior. — 1990. — Vol. 3, no. 2. — P. 159–168.

76. *Bonabeau E.*, *Dorigo M.*, *Theraulaz G.* Swarm Intelligence: From Natural to Artificial Systems. — New York, NY, USA : Oxford University Press, Inc., 1999. — 307 p.

77. Ant Colony Optimization for the Two-dimensional Loading Vehicle Routing Problem / G. Fuellerer, K. F. Doerner, R. F. Hartl, M. Iori // Comput. Oper. Res. — Oxford, UK, 2009. — Vol. 36, no. 3. — P. 655–673.

78. *Hernández H.*, *Blum C.* Ant colony optimization for multicasting in static wireless ad-hoc networks // Swarm Intelligence. — 2009. — Vol. 3, no. 2. — P. 125–148.

79. *Solnon C.* Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization // European Journal of Operational Research. — 2008. — Vol. 191, no. 3. — P. 1043–1055.

80. *Alba E.*, *Chicano F.* ACOhg: Dealing with Huge Graphs // Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation. — New York, NY, USA : ACM, 2007. — P. 10–17.

81. *Harman M.* Software Engineering Meets Evolutionary Computation // Computer. — 2011. — Vol. 44, no. 10. — P. 31–39.

82. *Langdon W. B.*, *Harman M.* Optimizing Existing Software With Genetic Programming // IEEE Transactions on Evolutionary Computation. — 2015. — Vol. 19, no. 1. — P. 118–135.

83. Improving CUDA DNA Analysis Software with Genetic Programming / W. B. Langdon, B. Y. H. Lam, J. Petke, M. Harman // Proceedings of the 2015 on Genetic and Evolutionary Computation Conference. — New York, NY, USA : ACM, 2015. — P. 1063–1070.

84. *Langmead B.*, *Salzberg S. L.* Fast gapped-read alignment with Bowtie 2 // Nature Methods. — 2012. — Vol. 9, no. 4. — P. 357–359.

85. BarraCUDA – a fast short read sequence aligner using graphics processing units / P. Klus, S. Lam, D. Lyberg, M. S. Cheung, G. Pullan, I. McFarlane, G. S. Yeo, B. Y. Lam // BMC research notes. — 2012. — Vol. 5, no. 1. — P. 27.

86. *Fogel L. J.* Autonomous automata // Industrial Research. — 1962. — Vol. 4. — P. 14–19.

87. *Fogel L. J.*, *Owens A. J.*, *Walsh M. J.* Artificial Intelligence Through Simulated Evolution. — John Wiley & Sons, 1966. — 170 p.

88. *Gomez J.* An Incremental-Evolutionary Approach For Learning Finite Automata // Proceedings of the IEEE Congress on Evolutionary Computation. — 2006. — P. 362–369.

89. *Levenshtein V. I.* Binary Codes Capable of Correcting Deletions, Insertions and Reversals // Soviet Physics Doklady. — 1966. — Vol. 10. — P. 707.

90. *Kim D.* Memory analysis and significance test for agent behaviours // Proceedings of the 8th annual conference on Genetic and evolutionary computation. — New York, NY, USA : ACM, 2006. — P. 151–158.

91. Genetic algorithm for induction of finite automata with continuous and discrete output actions / A. Alexandrov, A. Sergushichev, S. Kazakov, F. Tsarev // Proceedings of the 13th annual conference companion on Genetic and evolutionary computation. — New York, NY, USA : ACM, 2011. — P. 775–778.

92. *Gold M.* Complexity of Automaton Identification from Given Data // Information and Control. — 1978. — Vol. 37, no. 3. — P. 302–320.

93. *Oncina J.*, *Garcia P.* Identifying Regular Languages In Polynomial Time // Advances in structural and syntatics pattern recognition. — World Scientific, 1992. — P. 99–108.

94. *Lucas S.*, *Reynolds J.* Learning DFA: evolution versus evidence driven state merging // Proceedings of the IEEE Congress on Evolutionary Computation. Vol. 1. — 2003. — P. 351–358.

95. *Emerson E. A.* Temporal and Modal Logic // Handbook of Theoretical Computer Science. B. — MIT Press, 1990. — P. 995–1072.

96. *Casavant T.*, *Kuhl J.* A communicating finite automata approach to modeling distributed computation and its application to distributed decision-making // IEEE Transactions on Computers. — 1990. — Vol. 39, no. 5. — P. 628–639.

97. *Church A.* Logic, arithmetic and automata // Proceedings of the International Congress of Mathematicians. — 1962. — P. 22–35.

98. *Finkbeiner B.*, *Schewe S.* Bounded synthesis // International Journal on Software Tools for Technology Transfer. — 2013. — Vol. 15, no. 5. — P. 519–539.

99. Leveraging Existing Instrumentation to Automatically Infer Invariant-constrained Models / I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, M. D. Ernst // Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. — New York, NY, USA : ACM, 2011. — P. 267–277.

100. *Baker J.* Reducing bias and efficiency in the selection algorithm // Proceedings of the Second International Conference on Genetic Algorithms and their application. — 1987. — P. 14–21.

101. *Duret-Lutz A.* Manipulating LTL Formulas Using Spot 1.0 // Automated Technology for Verification and Analysis. — Springer International Publishing, 2013. — P. 442–445. — (Lecture Notes in Computer Science ; 8172).

102. *Wilcoxon F.* Individual Comparisons by Ranking Methods // Biometrics Bulletin. — 1945. — Vol. 1, no. 6. — P. 80–83.

103. *Holm S.* A simple sequentially rejective multiple test procedure // Scandinavian Journal of Statistics. — 1979. — Vol. 6. — P. 65–70.

104. The `irace` package, Iterated Race for Automatic Algorithm Configuration: tech. rep. / M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, M. Birattari ; IRIDIA, Université Libre de Bruxelles, Belgium. — 2011. — TR/IRIDIA/2011-004.

105. *Drechsler R.*, *Becker B.* Binary Decision Diagrams. Theory and Implementation. — Springer US, 1998. — 200 p.

106. Genetic Algorithm for Induction of Finite Automata with Continuous and Discrete Output Actions / A. Alexandrov, A. Sergushichev, S. Kazakov, F. Tsarev // Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation. — 2011. — P. 775–778.

107. *Alba E.* Parallel Metaheuristics: A New Class of Algorithms. — Wiley-Interscience, 2005. — 576 p.

108. *Dai W.*, *Dubinin V.*, *Vyatkin V.* Migration From PLC to IEC 61499 Using Semantic Web Technologies // IEEE Transactions on Systems, Man, and Cybernetics: Systems. — 2014. — Vol. 44, no. 3. — P. 277–291.

109. *Gerber C.*, *Hanisch H.-M.*, *Ebbinghaus S.* From IEC 61131 to IEC 61499 for Distributed Systems: A Case Study // EURASIP J. Embedded Syst. — 2008. — No. 4. — P. 1–8.

110. *Dai W.*, *Vyatkin V.* Redesign Distributed PLC Control Systems Using IEC 61499 Function Blocks // IEEE Transactions on Automation Science and Engineering. — 2012. — Vol. 9, no. 2. — P. 390–401.

111. *Patil S.*, *Vyatkin V.*, *Sorouri M.* Formal verification of Intelligent Mechatronic Systems with decentralized control logic // Proceedings of the 17th IEEE Conference on Emerging Technologies Factory Automation. — 2012. — P. 1–7.

Ресурсы сети Интернет

112. International standard IEC 61131. — URL: https://webstore.iec.ch/searchform%5C&q=61131.

113. International standard IEC 61499. — URL: https://webstore.iec.ch/searchform%5C&q=61499.

114. Search based software engineering repository. — URL: http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/.

115. Государственный контракт «Технология генетического программирования для генерации автоматов управления системами со сложным поведением». Промежуточный отчет по этапу I «Выбор направления исследований и базовых компонентов». — URL: http://is.ifmo.ru/genalg/_2007_01_patent-genetic.pdf.

116. Natural Selection, Inc. — URL: http://natural-selection.com/.

117. Red Cedar Technology, a CD-adapco company. — URL: http://www.redcedartech.com/.

118. NeuroSolutions. — URL: http://www.neurosolutions.com/.

119. Eurobios. — URL: http://www.eurobios.com.

120. AntOptima. — URL: http://www.antoptima.ch.

121. Learning DFA from Noisy Samples: A Contest for GECCO 2004. — URL: http://cswww.essex.ac.uk/staff/sml/gecco/NoisyDFA.html.

122. Lingeling, Plingeling and Treengeling. — URL: http://fmv.jku.at/lingeling/.

123. A Free and Open-Source Java Library for Constraint Programming. — URL: http://choco-solver.org/.

124. Model Checking @ CMU. — URL: http://www.cs.cmu.edu/~modelcheck/smv.html.

125. FlightGear Flight Simulator. — URL: http://www.flightgear.org/.

126. An advanced SAT solver. — URL: https://github.com/msoos/cryptominisat.

127. Graphviz – Graph Visualization Software. — URL: http://www.graphviz.org.

128. FBDK – The Function Block Development Kit. — URL: http://www.holobloc.com/doc/fbdk.

ПУБЛИКАЦИИ АВТОРА ПО ТЕМЕ ДИССЕРТАЦИИ

Статьи в журналах из перечня ВАК

129. Построение автоматных программ по спецификации с помощью муравьиного алгоритма на основе графа мутаций / Д. С. Чивилихин, В. И. Ульянцев, В. В. Вяткин, А. А. Шалыто // Научно-технический вестник информационных технологий, механики и оптики. — 2014. — 6(94). — С. 98—105.

130. *Чивилихин Д. С.*, *Ульянцев В. И.* Метод построения управляющих автоматов на основе муравьиных алгоритмов // Научно-технический вестник информационных технологий, механики и оптики. — 2012. — 6(82). — С. 72—76.

131. Генерация управляющих конечных автоматов по обучающим примерам на основе муравьиного алгоритма / И. П. Бужинский, В. И. Ульянцев, Д. С. Чивилихин, А. А. Шалыто // Известия РАН. Теория и системы управления. — 2014. — № 2. — С. 111—121.

Публикации в рецензируемых изданиях, индексируемых Web of Science или Scopus

132. *Chivilikhin D.*, *Ulyantsev V.*, *Shalyto A.* Combining Exact and Metaheuristic Techniques for Learning Extended Finite-State Machines from Test Scenarios and Temporal Properties // Proceedings of the 13th International Conference on Machine Learning and Applications. — IEEE Computer Society, 2014. — P. 350–355.

133. *Chivilikhin D.*, *Ulyantsev V.* Learning Finite-State Machines: Conserving Fitness Function Evaluations by Marking Used Transitions // Proceedings of the 12th International Conference on Machine Learning and Applications. Vol. 2. — IEEE Computer Society, 2013. — P. 90–95.

134. *Chivilikhin D.*, *Ulyantsev V.* MuACOsm: a new mutation-based ant colony optimization algorithm for learning finite-state machines // Proceedings of the 15th Genetic and Evolutionary Computation Conference. — ACM, 2013. — P. 511–518.

135. *Chivilikhin D.*, *Ulyantsev V.*, *Shalyto A.* Solving five instances of the artificial ant problem with ant colony optimization // Proceedings of the 7th IFAC Conference on Manufacturing Modelling, Management, and Control. — IFAC / Elsevier, 2013. — P. 1043–1048.

136. *Chivilikhin D.*, *Ulyantsev V.* Learning Finite-State Machines with Classical and Mutation-Based Ant Colony Optimization: Experimental Evaluation // Proceedings of the 1st BRICS countries Congress on Computational Intelligence. — IEEE Computer Society, 2013. — P. 528–533.

137. *Chivilikhin D.*, *Ulyantsev V.*, *Tsarev F.* Test-based Extended Finite-State Machines Induction with Evolutionary Algorithms and Ant Colony Optimization // Proceedings of the 14th Genetic and Evolutionary Computation Conference companion. — ACM, 2012. — P. 603–606.

138. *Chivilikhin D.*, *Ulyantsev V.* Learning Finite-State Machines with Ant Colony Optimization // Swarm Intelligence. — Springer Berlin / Heidelberg, 2012. — P. 268–275. — (Lecture Notes in Computer Science ; 7461).

139. Reconstruction of Function Block Logic using Metaheuristic Algorithm: Initial Explorations / D. Chivilikhin, A. Shalyto, S. Patil, V. Vyatkin // Proceedings of the 13th IEEE International Conference on Industrial Informatics. — IEEE Computer Society, 2015. — P. 1239–1242.

140. *Chivilikhin D.*, *Shalyto A.*, *Vyatkin V.* Inferring Automata Logic From Manual Control Scenarios: Implementation in Function Blocks // Proceedings of the 13th IEEE International Symposium on Parallel and Dis-

tributed Processing with Applications. — IEEE Computer Society, 2015. — P. 307–312.

141. *Chivilikhin D.*, *Ulyantsev V.* Inferring Automata-Based Programs from Specification With Mutation-Based Ant Colony Optimization // Proceedings of the 16th Genetic and Evolutionary Computation Conference companion. — ACM, 2014. — P. 67–68.

142. *Chivilikhin D.*, *Ulyantsev V.*, *Shalyto A.* Extended Finite-State Machine Inference With Parallel Ant Colony Based Algorithms // Proceedings of the 6th International Student Workshop on Bioinspired Optimization Methods and Their Applications. — Jozef Stefan Institute, 2014. — P. 117–126.

Другие публикации

143. *Чивилихин Д. С.* Эволюционные стратегии с адаптивным параметром на основе свойств ландшафта функции приспособленности // Всероссийская научная конференция по проблемам информатики СПИСОК. — СПб. : ВВМ. СПбГУ, 2013. — С. 525—531.

144. *Чивилихин Д. С.*, *Ульянцев В. И.*, *Шалыто А. А.* Муравьиный алгоритм для построения автоматных программ по спецификации // XII Всероссийское совещание по проблемам управления ВСПУ-2014. — М. : Институт проблем управления им. В.А. Трапезникова РАН, 2014. — С. 4531—4542.

145. *Чивилихин Д. С.*, *Ульянцев В. И.* Применение муравьиных алгоритмов для построения конечных автоматов // Всероссийская научная конференция по проблемам информатики СПИСОК. — СПб. : ВВМ. СПбГУ, 2012. — С. 409—410.

146. *Чивилихин Д. С.*, *Ульянцев В. И.* Метод построения конечных автоматов на основе муравьиного алгоритма // Всероссийская научная конференция по проблемам информатики СПИСОК. — СПб. : ВВМ. СПбГУ, 2013. — С. 517—524.

147. *Чивилихин Д. С.*, *Ульянцев В. И.*, *Шалыто А. А.* Метод построения конечных автоматов на основе муравьиного алгоритма // Интегрированные модели и мягкие вычисления в искусственном интеллекте. Сборник тезисов докладов VII-й Международной научно-технической конференции. — М. : Физматлит, 2013. — С. 931—942.

ПРИЛОЖЕНИЕ А. СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ ПРОГРАММ ДЛЯ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2013661249

Программное средство, реализующее муравьиный алгоритм для построения конечных автоматов

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Чивилихин Даниил Сергеевич (RU),***
Ульянцев Владимир Игоревич (RU)

Заявка № **2013618898**
Дата поступления **03 октября 2013 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***03 декабря 2013 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2015610291

Библиотека параллельных муравьиных алгоритмов для построения управляющих конечных автоматов

Правообладатель: ***федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Чивилихин Даниил Сергеевич (RU), Ульянцев Владимир Игоревич (RU)***

Заявка № **2014661300**
Дата поступления **06 ноября 2014 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***12 января 2015 г.***

Врио руководителя Федеральной службы по интеллектуальной собственности

Л.Л. Кирий

ПРИЛОЖЕНИЕ Б. СЦЕНАРИИ РАБОТЫ И ТЕМПОРАЛЬНЫЕ ФОРМУЛЫ АВТОМАТА УПРАВЛЕНИЯ ДВЕРЬМИ ЛИФТА

9

e11/z1; e2; e12/z2; e2

e11/z1; e2; e12/z2; e2; e11/z1; e2; e12/z2; e2

e11/z1; e2; e12/z2; e3/z1; e2; e12/z2; e2

e11/z1; e2; e12/z2; e2; e11/z1; e2; e12/z2; e3/z1; e2; e12/z2; e2

e11/z1; e2; e12/z2; e3/z1; e2; e12/z2; e3/z1; e2; e12/z2; e2

e11/z1; e4/z3

e11/z1; e2; e12/z2; e4/z3

e11/z1; e2; e12/z2; e2; e11/z1; e4/z3

e11/z1; e2; e12/z2; e3/z1; e4/z3

G(!wasEvent(ep.e11) or wasAction(co.z1))

G((!wasEvent(ep.e12) or wasAction(co.z2)) and (!wasAction(co.z2) or wasEvent(ep.e12)))

G((!wasEvent(ep.e4) or wasAction(co.z3)) and (!wasAction(co.z3) or wasEvent(ep.e4)))

G(!wasEvent(ep.e3) or wasAction(co.z1))

G(!wasEvent(ep.e2) or X(wasEvent(ep.e11) or wasEvent(ep.e12))i)

G(!wasEvent(ep.e11) or X(wasEvent(ep.e4) or wasEvent(ep.e2)))

G(!wasAction(co.z1) or X(wasEvent(ep.e2) or wasEvent(ep.e4)))

G(!wasEvent(ep.e12) or X(wasEvent(ep.e2) or wasEvent(ep.e3) or wasEvent(ep.e4)))

G(!wasEvent(ep.e3) or X(wasEvent(ep.e2) or wasEvent(ep.e4)))

G(!X(wasEvent(ep.e11) or wasEvent(ep.e12) or wasEvent(ep.e2) or wasEvent(ep.e3) or wasEvent(ep.e4)) or !wasEvent(ep.e4))

G(!(wasEvent(ep.e11) and X(wasEvent(ep.e2))) or X(X(wasEvent(ep.e12))))

G(!(wasEvent(ep.e12) and X(wasEvent(ep.e2))) or X(X(wasEvent(ep.e11))))

G(!(wasEvent(ep.e12) and X(wasEvent(ep.e3))) or X(X(wasEvent(ep.e2) or wasEvent(ep.e4))))