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

На правах рукописи

Буздалова Арина Сергеевна

Метод совместного использования эволюционных алгоритмов и обучения с подкреплением для оценки эффективности программ решения задач дискретной математики

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 Анатолий Абрамович

Санкт-Петербург — 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность темы исследования. При разработке программного обеспечения возникают ситуации, когда необходимо оценить эффективность работы программы. В частности, оценка времени ее работы в худшем случае (*верхняя оценка*) позволяет дать гарантии на время завершения. Также, пользуясь этой оценкой, можно сравнивать эффективность различных программных реализаций. Необходимость оценки времени работы программ возникает также и в учебном процессе: средства получения таких оценок помогают развить навык разработки эффективных алгоритмов.

Получить точную теоретическую оценку времени работы для многих алгоритмов сложно. Поэтому используются эмпирические оценки. Задача получения верхней оценки на время работы программы сводится к задаче поиска наборов значений входных переменных: необходимо найти значения, при которых программа работает как можно дольше. Будем называть наборы значений входных переменных *тестами*.

Из неразрешимости проблемы останова [101] следует, что не существует алгоритма, позволяющего для любой программы найти тест, на котором она работает дольше, чем на всех других возможных тестах. Поэтому на практике используется ограничение: необходимо найти тест, на котором время работы программы превышает некоторый заданный предел. Будем называть такой тест *сложным*. Если удается найти сложный тест, то программа считается неэффективной.

Рассмотрим класс программ, для которых предлагается решать задачу автоматизированного поиска сложных тестов. Во-первых, эта задача имеет смысл только для программ, применение которых подразумевает их завершение за конечное время. Программы, обеспечивающие постоянную работу каких-либо систем (например, работу веб-сервера), не подходят под это описание. Во-вторых, в данной работе рассматривается случай использования только дискретных входных переменных. Класс программ, удовлетворяющих этим огра-

ничениям, будем называть программами решения задач дискретной математики.

До последнего времени для NP-трудных задач, например такой, как задача о рюкзаке, тесты генерировались случайным образом. Однако генерация тестов таким образом не всегда позволяет найти сложный тест. Поэтому в работе [2] было предложено использовать эволюционные алгоритмы [5, 15, 55, 85] которые показали свою эффективность на ряде NP-трудных задач.

При поиске сложных тестов с помощью эволюционных алгоритмов *целевым критерием* оптимизации является время работы программы, которое необходимо максимизировать. Генерируемые тесты кодируются в виде *особей* эволюционного алгоритма. Для работы эволюционного алгоритма необходимо определить *функцию приспособленности* (ФП) особи, которая обычно совпадает с целевым критерием. Эффективность соответствующего процесса оптимизации определяется общим временем, необходимым для нахождения сложного теста.

Использование времени работы программы в качестве функции приспособленности обладает недостатками: например, при нескольких запусках одной и той же программы на одном и том же тесте можно получить разные значения времени работы из-за особенностей среды выполнения. Для устранения этих недостатков можно использовать *вспомогательные критерии,* каждый из которых соответствует счетчику числа итераций некоторого цикла программы или числа вызовов некоторой процедуры. Значения этих счетчиков не подвержены влиянию среды выполнения.

В указанном выше подходе [2] счетчики размещаются в коде вручную. При этом необходимо разместить счетчики так, чтобы они были наиболее перспективными с точки зрения поиска сложных тестов. Предлагается применять автоматический подход, при использовании которого не возникает необходимости решать задачу размещения счетчиков, так как в каждый цикл и в каждую процедуру вставляется по счетчику. После этого во время работы эволюционного

алгоритма требуется выбрать счетчики, наиболее перспективные с точки зрения поиска сложных тестов. Это, в отличие от ручного размещения, должно позволить использовать большее число счетчиков и автоматически выбирать наиболее эффективные из них.

Степень разработанности темы исследования. В существующих методах использования вспомогательных критериев они оптимизируются либо одновременно [74], либо в определенном порядке [111]. Как правило, этот порядок случайный [69]. Для генерации тестов ранее применялся Switch and Restart Algorithm (SaRA) [2], использующий вспомогательные критерии в случайном порядке. В упомянутых методах всем вспомогательным критериям вне зависимости от их эффективности предоставляется одинаковый вычислительный бюджет. Выявление и устранение из процесса оптимизации неэффективных вспомогательных критериев путем их выбора во время работы эволюционного алгоритма может сократить общее время, требующееся для нахождения сложных тестов. Будем называть выбор критериев во время работы эволюционного алгоритма *адаптивным*.

Таким образом, проблема повышения эффективности использования вспомогательных критериев в эволюционных алгоритмах при генерации тестов для оценки эффективности программ является **актуальной**.

Целью работы является уменьшение общего времени, необходимого для осуществления автоматизированной оценки эффективности программ, решающих задачи дискретной математики.

Основные задачи диссертационной работы, которые должны обеспечить выполнение указанной цели, состоят в следующем:

1. Разработать метод адаптивного выбора вспомогательных критериев, используемых в эволюционных алгоритмах при генерации тестов для указанного класса программ. Осуществить программную реализацию разработанного метода.

2. Получить асимптотические оценки времени работы алгоритма, реализующего предложенный метод, для определения его эффективности и выполнить корректировку алгоритма в случае необходимости.
3. Осуществить экспериментальное исследование предложенного метода и сравнить его с известными методами выбора вспомогательных критериев при генерации тестов, оценивающих эффективность программ решения задач дискретной математики.

Положения, выносимые на защиту.

1. Предложен новый метод адаптивного выбора вспомогательных критериев, предназначенный для использования в эволюционных алгоритмах при генерации тестов, оценивающих эффективность программ решения задач дискретной математики, и выполнена его программная реализация. Метод основан на применении обучения с подкреплением.
2. Получены асимптотические оценки времени работы алгоритма, реализующего предложенный метод, в случае использования вспомогательных критериев, которые могут уменьшать (эффективные критерии) и увеличивать (мешающие) число вычислений функции приспособленности. На основе выполненного анализа предложена модификация разработанного алгоритма.

Научная новизна первого результата состоит в том, что вспомогательные критерии ранее не выбирались с помощью обучения с подкреплением. На каждом этапе работы эволюционного алгоритма предложенный метод обучается выбирать критерий, ожидаемая эффективность которого максимальна. Научная новизна второго результата состоит в том, что впервые получены асимптотические оценки времени работы алгоритма, совмещающего обучение с подкреплением и эволюционный алгоритм.

Методология и методы исследований. Методологическую основу диссертации составляет обобщение поставленной задачи и ее формализация, конструирование методов решения задачи и математическое доказательство их

свойств, проведение вычислительных экспериментов и анализ их результатов. В работе используются методы дискретной математики, эволюционных вычислений, теории вероятностей и математической статистики.

Достоверность научных положений, выводов и практических рекомендаций, полученных в диссертации, подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, результатами асимптотического анализа времени работы алгоритма, реализующего предложенный в диссертации метод, а также результатами экспериментов по использованию предложенного метода и их статистическим анализом.

Теоретическое значение работы состоит в том, что предложен метод адаптивного выбора вспомогательных критериев оптимизации, позволяющий автоматически выбирать критерии во время работы эволюционного алгоритма, и получены асимптотические оценки времени работы алгоритмов, реализующих этот метод.

Практическое значение работы состоит в том, что предложенный метод позволяет автоматически выбирать вспомогательные критерии оптимизации, использование которых уменьшает время, необходимое для генерации тестов. Это, в частности, позволило повысить эффективность генерации тестов на примере NP-трудной задачи «Ships. Version 2» [118].

Внедрение результатов работы. Результаты диссертации внедрены при разработке тестов для архива задач с проверяющей системой *Timus Online Judge*, функционирующего на базе Уральского федерального университета имени первого Президента России Б. Н. Ельцина, г. Екатеринбург, используемого для подготовки к олимпиадам по программированию. Результаты диссертации также были использованы в курсе лекций «Алгоритмы и структуры данных», который читается автором в течение нескольких лет на кафедре «Информационные системы» Университета ИТМО. Кроме того, эти результаты применялись в учебном процессе кафедры «Компьютерные технологии» этого универ-

ситета при руководстве тремя бакалаврскими работами и пятью магистерскими диссертациями, авторы и названия которых приведены в диссертации.

Апробация результатов работы. Основные результаты работы докладывались на следующих конференциях и семинарах:

- Всероссийская научная конференция по проблемам информатики *СПИСОК* (2012, 2013, 2014, 2016, 2017, Матмех СПбГУ);
- *Genetic and Evolutionary Computation Conference* (2013, Амстердам, Нидерланды; 2014, Ванкувер, Канада; 2015, Мадрид, Испания; 2017, Берлин, Германия);
- *Dagstuhl Seminar* «Theory of Evolutionary Algorithms» и *Dagstuhl Seminar* «Theory of Randomized Optimization Heuristics» (2015, 2017, Дагштул, Германия)
- *IEEE Congress on Evolutionary Computation* (2013, Канкун, Мексика);
- *Parallel Problem Solving from Nature* (2014, Любляна, Словения);
- *International Conference on Machine Learning and Applications* (2012, Бока-Ратон, США; 2013, Майами, США; 2014, Детройт, США);
- *International Symposium on Search-Based Software Engineering* (2013, Санкт-Петербург);
- *International Conference on Soft Computing MENDEL* (2012, 2014, 2016, Брно, Чехия).

Кроме того, автор неоднократно выступал с докладами на Всероссийском конгрессе молодых ученых, проводимом Университетом ИТМО.

Личный вклад автора. Решение задач диссертации, разработанный метод и его программная реализация, экспериментальные и теоретические результаты, представленные в диссертации и выносимые на защиту, принадлежат лично автору. Адаптация эволюционного алгоритма к решению задачи генерации тестов для задач дискретной математики выполнена совместно с М. В. Буздаловым.

Публикации. Основные результаты по теме диссертации изложены в 23 публикациях [1, 3, 12, 20, 21, 29, 31–42, 94–97, 116], три из которых изданы в журналах, рекомендованных ВАК [1, 3, 12], а 20 — в изданиях, индексируемых в международных базах цитирования *Web of Science* [29, 31, 36–38] и *Scopus* [20, 21, 29, 31–42, 95–97, 116]. В указанных работах авторство принадлежит соавторам в равных долях.

Свидетельства о регистрации программ для ЭВМ. Автором по теме диссертации получено два свидетельства о государственной регистрации программ для ЭВМ:

1. № 2015610563 от 13 января 2015 года «Программная библиотека для исследования и сравнения различных методов машинного обучения», авторы Буздалов М. В., Буздалова А. С.
2. № 2013610657 от 09 января 2013 года «Программное средство для исследования алгоритмов выбора оптимальной функции приспособленности», авторы Буздалов М.В., Буздалова А.С.

Участие в научно-исследовательских работах. Результаты диссертации были применены при выполнении следующих работ:

– «Повышение эффективности эволюционных алгоритмов с помощью динамически выбираемых вспомогательных критериев оптимизации» (Грант 16-31-00380 Российского фонда фундаментальных исследований. Сроки выполнения: 2016–2018 гг.) **Руководитель проекта — автор диссертации**.
– НИР «Биоинформатика, машинное обучение, технологии программирования, теория кодирования, проактивные системы» (Программа государственной финансовой поддержки ведущих университетов Российской Федерации, субсидия 074-U01. Сроки выполнения: 2013–2018 гг.)
– «Разработка методов автоматической генерации тестов на основе эволюционных алгоритмов» (Государственный контракт №14.740.11.1430 в рамках федеральной целевой программы «Научные и научно-

педагогические кадры инновационной России на 2009–2013 годы». Сроки выполнения: 2011, 2012 гг.)

Автор диссертации является победителем конкурсов грантов 2014 и 2016 гг. Комитета по науке и высшей школе Правительства Санкт-Петербурга для студентов и аспирантов вузов, отраслевых и академических институтов, расположенных на территории Санкт-Петербурга. Темы проектов: «Повышение эффективности эволюционных алгоритмов с помощью обучения с подкреплением» и «Разработка метода выбора вспомогательных критериев оптимизации в эволюционных алгоритмах, сохраняющего особь с лучшим значением целевого критерия».

Объем и структура работы. Диссертация состоит из введения, четырех глав, заключения и приложения. Объем диссертации составляет 155 страниц с 14 рисунками, 13 таблицами и семью листингами. Список литературы содержит 121 наименование.

В **первой главе** приводится обзор методов генерации тестов для оценки эффективности программ и обосновывается необходимость разработки адаптивных алгоритмов, повышающих эффективность существующего автоматизированного метода, основанного на применении эволюционных алгоритмов. Ставится задача разработки метода выбора вспомогательных критериев оптимизации в эволюционных алгоритмах, приводится обзор существующих методов использования вспомогательных критериев. Также рассматриваются методы адаптивной настройки эволюционных алгоритмов, на основе чего показывается целесообразность применения обучения с подкреплением для решения поставленной задачи.

Во **второй главе** предлагается метод, названный автором EA + RL, позволяющий адаптивно выбирать, какие критерии оптимизации следует использовать на текущем этапе генерации тестов. Критерии выбираются из множества, состоящего из целевого критерия и множества вспомогательных критери-

ев. Выбор производится в соответствии со стратегией, обновляемой с помощью алгоритма обучения с подкреплением.

В **третьей главе** приводится теоретический анализ предложенного метода. Моделируются различные ситуации, возникающие при автоматической генерации тестов, используемых для оценки эффективности программ. На основе теоретического анализа предлагается модифицированная версия метода.

В **четвертой главе** приводится описание и результаты применения предложенного метода к выявлению неэффективных программ, решающих *NP*-трудную олимпиадную задачу по программированию — «Ships. Version 2» [118]. Предложенный метод и его модификация позволяют добиться более стабильной генерации сложных тестов за меньшее время, чем ранее использовавшийся алгоритм и существующие аналоги.

ГЛАВА 1 ОБЗОР МЕТОДОВ ГЕНЕРАЦИИ ТЕСТОВ, ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ И ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ

В данной главе приводится обзор предметных областей, имеющих отношение к цели и задачам диссертации. В разделе 1.1 приводится обзор существующих методов автоматизированной оценки эффективности программ, а также некоторых методов автоматизированного тестирования. Делается вывод о целесообразности применения эволюционных алгоритмов (ЭА), основные концепции работы которых описаны в разделе 1.2.

Эффективность ЭА может быть повышена путем введения вспомогательных критериев оптимизации. Соответствующие исследования рассматриваются в разделе 1.3. На основе приведенного обзора делается вывод о необходимости разработки метода адаптивного выбора вспомогательных критериев, показывается целесообразность выбора вспомогательных критериев с помощью обучения с подкреплением. Принципы работы алгоритмов обучения с подкреплением рассматриваются в разделе 1.4.

1.1 Автоматизированная оценка эффективности программ

Одним из способов оценки времени работы программы является теоретический анализ времени работы соответствующего алгоритма. Для многих классических алгоритмов решения задач дискретной математики известны асимптотические оценки времени работы [8]. Также в некоторых случаях можно получить теоретические оценки времени работы алгоритмов в автоматических системах доказательства, таких как Coq и Agda [47, 113]. Однако независимо от используемых инструментов, получение теоретических оценок для новых или нестандартных алгоритмов — сложная творческая задача, требующая непосредственного участия человека. Кроме того, теоретические оценки, как правило, не позволяют учесть специфику реализации алгоритмов и особенности среды выполнения.

На практике может возникать необходимость быстрой оценки эффективности программ, реализующих нестандартные алгоритмы. Пользуясь такой оценкой можно сравнивать эффективность различных программных реализаций, решающих интересующую практическую задачу. Автоматически оценивать время работы программ также требуется на соревнованиях по программированию и в учебном процессе для развития навыка разработки эффективных алгоритмов [119]. Рассмотрим существующие методы автоматизированного получения эмпирических оценок времени работы программ.

1.1.1 Методы автоматизированной оценки эффективности программ

Существует ряд работ, в которых тесты, позволяющие оценить время работы программ, генерируются с помощью эволюционных алгоритмов [24, 25, 61–65, 108, 110, 114]. В этих работах рассматривается особый класс программ: программы для встраиваемых устройств. В качестве функции приспособленности используется время работы программы. Показано, что время работы программы — недостаточно надежный критерий измерения приспособленности особей, так как сильно подвержено зашумлению: два разных запуска одной и той же программы могут завершиться за разное время.

Проблема неудобства использования времени работы в качестве функции приспособленности эволюционного алгоритма частично решается в работе [2]. С помощью эволюционного алгоритма генерируются тесты, выявляющие неэффективные программы решения олимпиадных задач по программированию. Помимо времени работы программы, в качестве функций приспособленности рассматриваются вспомогательные критерии — значения счетчиков числа итераций циклов и вызовов процедур, которые вручную вставляются в код. Таким образом, этот подход требует некоторой обработки каждой тестируемой программы вручную. Счетчики используются следующим образом: каждый из них по очереди используется в качестве функции приспособленности на протяжении определенного числа итераций, затем все счетчики снова перебираются в

течение большего числа итераций и так далее, пока не находится тест, на котором программа работает дольше заданного ограничения.

1.1.2 Другие методы автоматизированного тестирования

Для полноты описания перечислим методы автоматизированного тестирования программ, не решающие непосредственно задачу оценки времени работы, но имеющие к ней то или иное отношение.

В ранее рассмотренных методах оценки времени работы программ используются эволюционные алгоритмы. Эволюционные алгоритмы также могут использоваться для улучшения программ путем генерации *патчей* — небольших изменений в коде программы [76]. Корректность и эффективность программ, измененных с помощью патчей, проверяется на фиксированном наборе тестов. Возможность генерации дополнительных тестов, позволяющих проверить время работы измененных программ на наиболее сложных входных данных, могла бы повысить эффективность данного подхода.

Эволюционные алгоритмы также могут применяться для поиска входных данных, однозначно определяющих состояние конечного автомата, которые в ряде источников тоже называются тестами. Отметим работу [77], которая является первым шагом на пути теоретического анализа эволюционных алгоритмов, применяемых для генерации подобных тестов. В ней приводятся асимптотические оценки времени работы (1+1) эволюционного алгоритма.

Одной из наиболее популярных задач в области разработки программного обеспечения, решаемых автоматическими средствами, является задача покрытия кода тестами. Для решения этой задачи необходимо сгенерировать набор тестов, обеспечивающий хотя бы однократный запуск каждой инструкции программы. Эта задача решается с помощью динамического символьного вычисления [28, 103], в частности, с помощью генерации параметризованных тестов [109]. Генерация тестов, оценивающих время работы программ, с помощью этих инструментов представляется неэффективной: запуск символьного

вычисления, строящего путь выполнения, который бы соответствовал максимально возможному времени работы программы, требует заведомо больше времени, чем само максимальное время работы программы.

Существует также подход, позволяющий генерировать тесты, покрывающие все возможные пути выполнения программы [92]. Среди этих путей должен существовать путь, соответствующий максимально возможному времени работы программы с учетом ограничений на входные данные. Однако этот подход был разработан для программного обеспечения встраиваемых систем и проверялся только для программ без циклов и рекурсии. Применение данного метода для оценки эффективности программ общего вида, в частности программ решения задач дискретной математики, представляется затруднительным.

1.1.3 Обоснование необходимости разработки метода выбора вспомогательных критериев

Из приведенного обзора следует, что генерация тестов, позволяющих оценить время работы программ, в существующих исследованиях производится с помощью эволюционных алгоритмов. Однако целевой критерий оптимизации — время работы программы — оказывается сложно использовать в качестве функции приспособленности эволюционного алгоритма из-за зашумленности значений. Эта проблема частично решается добавлением вспомогательных критериев оптимизации. Однако в существующем подходе критерии добавляются вручную. При автоматическом добавлении вспомогательных критериев возникает вопрос, какие критерии наиболее эффективны. Метод адаптивного выбора критериев, наиболее эффективных на текущем этапе работы эволюционного алгоритма, может как ускорить существующий подход, так и сделать возможным использование автоматически добавляемых критериев.

1.2 Эволюционные алгоритмы

Практические задачи оптимизации часто достаточно сложны и не имеют точного решения, либо такое решение невозможно найти за разумное время.

В частности, это верно для NP-трудных задач в предположении, что $P \neq NP$. Для решения подобных задач применяются разнообразные эвристики, позволяющие получить решение приемлемого качества в условиях ограниченных вычислительных и временных ресурсов.

Эвристика может быть разработана специально для решения конкретной задачи, однако такой подход достаточно трудозатратен. Поэтому распространение получили так называемые метаэвристики, предоставляющие общие подходы, применимые для решения классов задач [56]. Многие из таких метаэвристик являются биоинспирированными. Например, задачи оптимизации на графах достаточно естественно могут быть представлены в виде, удобном для решения с помощью алгоритмов роевого интеллекта, в частности, муравьиных алгоритмов [9, 10, 14, 51, 52]. Одним из наиболее широко применяемых семейств биоинспирированных метаэвристик являются эволюционные алгоритмы. Важным достоинством эволюционных алгоритмов является их применимость для решения широкого круга задач без существенной доработки [56].

Как правило, эволюционные алгоритмы применяются для решения задач оптимизации, в которых недоступна информация о градиенте оптимизируемой функции, и, соответственно, которые невозможно решить методами градиентного спуска [82]. Для того, чтобы эволюционный алгоритм можно было применить для решения задачи, требуется следующее:

- возможность сравнивать потенциальные решения (как правило, на основе их приспособленности, определяемой *функцией приспособленности*);
- возможность генерировать потенциальные решения;
- возможность вносить изменения в потенциальные решения (соответствующий процесс называется *мутацией*).

Многие эволюционные алгоритмы используют возможность *скрещивания* потенциальных решений: получения нового решения на основе нескольких существующих. Использование скрещивания не является обязательным. Было доказано, что для нахождения эволюционным алгоритмом оптимального решения

достаточно применения мутации [57]. Однако из этого не следует, что использование оператора скрещивания не может позволить найти оптимальное решение быстрее. Существуют работы, в которых доказана эффективность применения оператора скрещивания при решении ряда задач [7, 48, 49, 58, 68, 106].

Функцию приспособленности естественно использовать в случае, когда стоит задача оптимизации. Однако существуют задачи, в которых напрямую не требуется оптимизация функции, а необходимо, например, сгенерировать сущность, эффективно действующую в определенных условиях. Задачи такого рода решаются с помощью *коэволюционных* алгоритмов [4, 44, 78]. Можно рассмотреть следующие их разновидности:

– «Все против всех». Приспособленность определяется как успешность особи в соперничестве с другими подобными особями.
– «Хищник-жертва». Имеется два типа особей: хищники и жертвы. Приспособленность хищника определяется как способность успешно уничтожать жертвы, а приспособленность жертвы как способность успешно противостоять хищникам.
– Кооперативные алгоритмы. Имеется несколько типов особей, при этом решение поставленной задачи складывается из особей разных типов как из составных частей. Приспособленность особи определяется как среднее качество решений, составленных при участии этой особи.

Возможным способом оценки эффективности программ может быть коэволюция вида «хищник-жертва», в которой в качестве жертв выступают программы, решающие некоторую задачу дискретной математики, а в качестве хищников — тесты, оценивающие эффективность программ. Однако для реализации этого подхода необходимо предложить осмысленные операторы мутации и скрещивания для программ, решающих рассматриваемую задачу дискретной математики. В частности, эти операторы должны быть устроены так, чтобы получающиеся в результате программы были корректными и по-прежнему решали рассматриваемую задачу. Это затруднительно, так как алгоритмы, лежа-

щие в основе программ, могут существенно различаться. Полное и всестороннее рассмотрение данного вопроса выходит за рамки настоящего исследования. Поэтому далее будем рассматривать только «классические» эволюционные алгоритмы, оптимизирующие функцию приспособленности.

1.2.1 Однокритериальные эволюционные алгоритмы

Однокритериальные эволюционные алгоритмы могут использоваться для решения задачи оптимизации некоторой функции, или *критерия* оптимизации. Как правило, необходимо найти точку глобального оптимума (максимума или минимума) оптимизируемой функции. Однако существуют и так называемые алгоритмы мультимодальной оптимизации, задачей которых является выявление точек локальных оптимумов [105, 115]. В настоящей диссертации рассматривается задача поиска точки глобального оптимума. Значение глобального оптимума при решении практических задач часто неизвестно. В модельных задачах искомое значение оптимума, как правило, известно, как и искомая точка оптимума. Однако эта информация не может быть применена в алгоритме и используется исключительно для анализа его эффективности.

Базовая схема работы эволюционного алгоритма может быть описана следующим образом [5, 15, 56].

– Кандидаты в искомое решение кодируются в виде *особей* эволюционного алгоритма. Распространенным способом кодирования являются битовые строки, используются также перестановки, деревья и многие другие способы представления в зависимости от решаемой задачи.
– Для оптимизации требуемого критерия необходимо определить *функцию приспособленности (ФП)* особей. В некоторых случаях критерий и функция приспособленности могут совпадать, однако существуют задачи, в которых критерий затруднительно представить в виде функции, и значение функции приспособленности является приблизительной оценкой эффективности особи с точки зрения требуемого критерия.

– Набор особей называется *поколением*. Первое поколение, как правило, генерируется случайным образом.
– Из текущего поколения отбираются особи, подвергаемые *скрещиванию* и *мутации*. Выбор особей для скрещивания, как правило, производится на основе их приспособленности.
– В зависимости от схемы отбора, из новых и, возможно, старых особей отбираются особи, формирующие следующее поколение. Отбор особей для следующего поколения тоже, как правило, осуществляется на основе их приспособленности.

Существуют различные условия останова эволюционного алгоритма. Наиболее распространенными являются условия, связанные с исчерпанием выделенного вычислительного бюджета или достижением точки оптимума функции приспособленности. Для использования второго условия останова необходимо знать значение оптимума. Оно часто применяется в теоретических исследованиях [88]. Вычислительный бюджет, используемый в первом условии, можно определить как общее время, выделяемое для работы эволюционного алгоритма. Также распространенным способом задания вычислительного бюджета является ограничение числа вычислений ФП. В случаях, когда вычисление ФП занимает примерно одинаковое время для любой особи, число вычислений ФП напрямую связано с общим временем работы алгоритма. Кроме того, для многих задач вычисление ФП является самой ресурсоемкой процедурой и вносит основной вклад в общее время работы [55].

Способы измерения эффективности эволюционных алгоритмов связаны с используемым условием останова. В случае запуска на фиксированном вычислительном бюджете эффективность алгоритма, как правило, определяется лучшим найденным значением ФП; в случае же достижения оптимального значения ФП — общим временем работы алгоритма или числом вычислений ФП, понадобившимся для достижения этого значения.

1.2.2 Многокритериальные эволюционные алгоритмы

Многие практические задачи оптимизации являются многокритериальными — это значит, что требуется минимизировать или максимизировать не одну, а несколько величин, которые часто конфликтуют друг с другом. При такой постановке задачи целесообразно находить не какое-то одно решение, а Парето-оптимальное множество решений — то есть такое множество, в котором никакие два решения не доминируют друг друга в смысле Парето, при этом никакое решение из этого множества не доминируется по Парето никаким другим возможным решением. Часто нахождение такого множества целиком либо невозможно (например, если оптимизируемые величины являются вещественнозначными), либо нецелесообразно по соображениям вычислительной сложности. В этом случае, как правило, ищут приближение Парето-оптимального множества решений. Рассмотрим основные принципы работы многокритериальных алгоритмов, которые позволяют найти это приближение.

1.2.2.1 Доминирование по Парето

В K-мерном пространстве, точка $A = (a_1, \ldots, a_K)$ *доминирует (в смысле Парето)* точку $B = (b_1, \ldots, b_K)$, когда для всех $1 \leqslant i \leqslant K$ выполняется неравенство $a_i \leqslant b_i$, и, кроме того, существует такое j, что $a_j < b_j$. Без ограничения общности, предположим, что решается многокритериальная задача минимизации с числом критериев, равным k. В этом случае, отношение доминирования по Парето определяется на двух k-мерных точках следующим образом [43]:

$$a \prec b \leftrightarrow \forall i \in [1; k]\ a_i \leqslant b_i \text{ и } \exists i \in [1; k]\ a_i < b_i;$$
$$a \preceq b \leftrightarrow \forall i \in [1; k]\ a_i \leqslant b_i,$$

где $a \prec b$ означает отношение *строгого доминирования*, а $a \preceq b$ означает отношение *слабого доминирования*.

Множество известных и широко применяемых многокритериальных эволюционных алгоритмов используют процедуру недоминирующей сортировки,

или процедуру определения множества недоминирующих решений, которая может быть сведена к недоминирующей сортировке. *Недоминирующая сортировка* [104] множества точек S в K-мерном пространстве — это процедура, которая назначает всем точкам из S, которые не доминируются ни одной точкой из множества S, *ранг*, равный нулю (обозначим множество точек с нулевым рангом как S_0), всем точкам из множества $S'_0 = S \setminus S_0$, которые не доминируются ни одной точкой из множества S'_0, ранг, равный единице (множество таких точек обозначим как S_1), аналогично, всем точкам из множества $S'_i = S \setminus \bigcup_{j=0}^{i-1} S_i$, которые не доминируются ни одной точкой из множества S'_i, ранг, равный i.

Примерами таких алгоритмов могут послужить NSGA-II [16], PESA [46], PESA-II [93], SPEA2 [117], PAES [75], PDE [19] и многие другие алгоритмы. Вычислительная сложность одной итерации этих алгоритмов часто определяется сложностью процедуры недоминирующей сортировки. Рассмотрим подробнее алгоритм NSGA-II [16], являющийся одним из самых популярных алгоритмов многокритериальной оптимизации на данный момент [43]. Этот алгоритм используется в главе 4 настоящей работы в экспериментальном сравнении с существующими методами.

1.2.2.2 Алгоритмы NSGA и NSGA-II

В работе [104] предложена вариация техники Гольдберга [104], названная Non-dominated Sorting Genetic Algorithm («генетический алгоритм с недоминирующей сортировкой»), или NSGA. Этот алгоритм содержит несколько уровней классификации особей. Перед тем, как осуществить отбор особей, особям назначается ранг в соответствии с уровнем доминирования. Более точно, всем недоминированным особям назначается один и тот же ранг, пропорциональный размеру популяции. Затем эти особи убираются из рассмотрения в процедуре назначения ранга, и процесс повторяется: все недоминированные особи среди оставшихся получают ранг, пропорциональный размеру оставшейся популяции, и так далее. Так как процедура отбора назначает вероятность вы-

бора особи, пропорциональную ее рангу, то недоминированные особи получают наибольшую вероятность быть выбранными, чем улучшают процесс поиска в Парето-оптимальных регионах пространства поиска. Для равномерного распределения популяции по таким регионам дополнительно используется процедура разделения приспособленности между особями с равным рангом.

Алгоритм NSGA обладал высокой вычислительной сложностью $O(N^3K)$, где N — число особей в популяции, а K — число критериев, по причине крайне неэффективного осуществления недоминирующей сортировки. Положение было исправлено в алгоритме NSGA-II [16]. В отличие от своего предшественника, алгоритма NSGA, данный алгоритм был с самого начала определен в обобщенной формулировке, допускающей различные операторы скрещивания и мутации. После недоминирующей сортировки, имеющей в данной вариации вычислительную сложность $O(N^2K)$, в качестве метода разбиения на ниши применяется так называемая плотность населенности (англ. *crowding distance*): чем на большее расстояние особь отстоит от своих ближайших соседей с тем же самым номером недоминирующего слоя, тем больше у нее вероятность быть выбранной. Данный алгоритм в настоящее время может быть названным самым популярным эволюционным алгоритмом многокритериальной оптимизации, в частности, утверждается, что он используется в большинстве сравнительных анализов многокритериальных алгоритмов [43, раздел 2.3.2].

1.3 Повышение эффективности ЭА за счет введения вспомогательных критериев

Первое исследование возможности использования вспомогательных критериев для повышения эффективности оптимизации, известное автору, относится к работе [6]. Однако в этой работе рассматривается оптимизация непрерывных функций, производящаяся классическими методами, предполагающими возможность вычисления производной. Рассмотрим современное состояние исследований в области эволюционных алгоритмов, с помощью которых про-

водится оптимизация дискретных функций, для которых невозможно вычислить производную.

Известны исследования по повышению эффективности эволюционных алгоритмов с помощью вспомогательных критериев [87, 89, 112]. В ряде существующих подходов вспомогательные критерии создаются вручную и оптимизируются одновременно с помощью многокритериальных эволюционных алгоритмов [66, 74, 81]. Также существует подход, в котором вспомогательные критерии оптимизируются не одновременно, а выбираются динамически в процессе работы эволюционного алгоритма [69]. Еще одной особенностью этого метода является то, что вспомогательные критерии оптимизируются одновременно с целевым критерием. Вспомогательные критерии, используемые для текущего этапа оптимизации, выбираются случайным образом из списка всех доступных вспомогательных критериев. Подобный подход не позволяет учесть особенности решаемой задачи оптимизации. В других работах предлагается использовать частные подходы: для определения порядка выбора вспомогательных критериев используется информация о решаемой задаче оптимизации [79, 80]. Такие подходы являются недостаточно общими и не могут быть модифицированы для выбора вспомогательных критериев при генерации тестов, оценивающих эффективность программ.

Общим недостатком перечисленных подходов повышения эффективности оптимизации целевого критерия с помощью вспомогательных является то, что в них используются многокритериальные эволюционные алгоритмы, основанные на сравнении по Парето. Итерации таких алгоритмов имеют большую вычислительную сложность, чем итерации однокритериальных эволюционных алгоритмов, которые используются при генерации тестов. Также в существующих подходах предполагается, что использование любого из вспомогательных критериев должно приводить к повышению эффективности оптимизации целевого критерия, по крайней мере, на некотором этапе оптимизации. Убедиться в выполнении этого требования достаточно сложно. Также сложно разрабаты-

вать соответствующие критерии [69, 74]. Таким образом, как отмечается в выводах к работе [69], необходим метод, позволяющий автоматически выбирать эффективные вспомогательные критерии и игнорировать неэффективные. Такой метод можно было бы применять, в том числе, к выбору критериев, сгенерированных автоматически, свойства которых заранее не известны.

Большинство существующих методов адаптивной настройки эволюционных алгоритмов предназначены для настройки числовых параметров [13, 50, 71]. Существуют также методы, позволяющие настраивать качественные параметры, а именно, выбирать эволюционные операторы во время работы алгоритма [100]. Эти методы основаны на применении обучения с подкреплением. Как показано в следующей главе, обучение с подкреплением достаточно естественно может быть применено для выбора элементов из конечного множества. Таким образом, задачу выбора вспомогательных критериев для генерации тестов предлагается решать с помощью обучения с подкреплением.

1.4 Обучение с подкреплением

В данном разделе описываются основные понятия и принципы, на которых основаны алгоритмы обучения с подкреплением.

1.4.1 Общие понятия обучения с подкреплением

Обучение с подкреплением используется для решения задач взаимодействия с динамической средой. Общая схема алгоритмов обучения с подкреплением такова: *агент* применяет *действие* к *среде*, после чего получает от среды *награду* и некоторое представление текущего *состояния* среды [107]. Агент использует полученную информацию для обучения, и цикл взаимодействия со средой повторяется. Задачей агента является максимизация суммарной награды. Схема взаимодействия агента со средой представлена на рисунке 1, где t — номер итерации алгоритма обучения с подкреплением. Далее введенные понятия будут описаны более подробно.

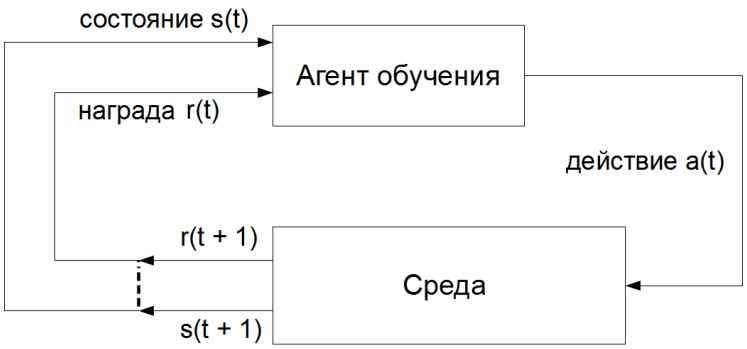

Рисунок 1 – Схема взаимодействия агента со средой в алгоритмах обучения с подкреплением [107]

Действие может как выбираться из некоторого конечного дискретного набора, так и, например, принадлежать множеству векторов вещественных чисел $\mathbb{R}^n$ [70, 100]. В настоящей диссертации в качестве действия рассматривается выбор критерия оптимизации из некоторого конечного заранее заданного набора. Поэтому будем рассматривать алгоритмы обучения с подкреплением, предназначенные для работы с конечным дискретным набором действий.

Состояние также может быть как элементом конечного дискретного набора, так и представляться вектором вещественных чисел [70]. В настоящей диссертации предлагаются способы определения состояния эволюционного алгоритма, позволяющие получить конечный дискретный набор состояний.

Награда обычно представляет собой целое или вещественное число [107]. Далее будем считать, что награда является вещественной. Положительная награда свидетельствует об эффективности действий агента, отрицательная — о неэффективности.

1.4.2 Способы оценки суммарной награды

Обозначим *мгновенную* награду, полученную на t-ой итерации алгоритма обучения с подкреплением, как r_t. Задача максимизации суммарной награды может быть формализована различными способами, соответствующими модели конечного горизонта ($\max E(\sum_{t=0}^{h} r_t)$, где h — число итераций алгоритма), бесконечного горизонта ($\max E(\sum_{t=0}^{\infty} \gamma^t r_t)$, где $0 < \gamma < 1$ — дис-

контный фактор) и достаточно редко использующейся модели средней награды ($\max \lim_{h\to 0} E(\frac{1}{h}\sum_{t=0}^{h} r_t)$) [11].

В настоящей диссертации рассматривается награда, формируемая эволюционным алгоритмом, время завершения которого не всегда известно заранее. Например, время завершения работы эволюционного алгоритма практически невозможно точно определить в случае, когда условием останова является достижение оптимального значения функции приспособленности [85]. Поэтому модель конечного горизонта неприменима, и предлагается использовать модель бесконечного горизонта.

1.4.3 Стратегии исследования среды

В алгоритмах обучения с подкреплением применение накопленного опыта совмещается с необходимостью исследовать еще не посещенные состояния среды. Эти задачи часто конфликтуют. Существуют различные стратегии совмещения применения опыта с исследованием среды.

Обозначим как $Q(s,a)$ ожидаемую эффективность применения действия a в состоянии s. Эффективность действия определяется его вкладом в суммарную награду. Одна из простейших стратегий заключается в том, чтобы не пытаться исследовать среду, а всегда выбирать действие, наиболее эффективное в текущем состоянии. Такая стратегия называется *жадной* [11, 107]. Этот подход может привести к тому, что будет найдена локально оптимальная стратегия поведения, не позволяющая в полной мере максимизировать суммарную награду.

Существует также ε-жадная стратегия исследования среды [11, 107], при которой с вероятностью ε агент выбирает случайное действие, а в остальных случаях придерживается жадной стратегии. Данная стратегия позволяет избежать остановки в локальном оптимуме. Однако выбор случайного действия теряет смысл, когда агент полностью исследовал среду.

Еще одним способом исследования среды является исследование по Больцману [11]. В соответствии с этим способом, вероятность выбора действия

a в состоянии s составляет $p(a) = \frac{e^{Q(s,a)/T}}{\sum_{a' \in A} e^{Q(s,a')/T}}$, где T — температура, параметр, убывающий со временем. Данный подход лишен недостатков описанных выше стратегий, однако его эффективность зависит от подбора параметра скорости убывания температуры.

Перечисленные стратегии могут применяться в различных алгоритмах обучения с подкреплением, которые не накладывают ограничения на то, какая стратегия используется. Также в некоторых алгоритмах могут использоваться специальные стратегии исследования среды [70, 90].

1.4.4 Q-обучение

Опишем Q-обучение (англ. *Q-learning*) — популярный алгоритм обучения с подкреплением, не строящий модель среды и относящийся к семейству алгоритмов итерации по значениям [11, 107]. В ходе работы этого алгоритма аппроксимируется функция $Q(s, a), s \in S, a \in A$ — ожидаемая оптимальная награда за действие a в состоянии s. В качестве стратегии исследования среды будем использовать ε-жадную стратегию. В листинге 1 приведен псевдокод алгоритма.

Отметим, что существует алгоритм отложенного Q-обучения, для которого доказана более высокая скорость сходимости [90]. Еще одним похожим алгоритмом является R-обучение, в котором используется модель средней награды [102].

1.4.5 Существующие теоретические результаты

Рассмотрим состояние теоретических исследований в области анализа обучения с подкреплением и эволюционных алгоритмов. Особенностью предлагаемого подхода является то, что в нем одновременно используются две сложные для анализа эвристики: эволюционные алгоритмы и обучение с подкреплением. Существуют и другие методы, в которых обучение с подкреплением используется для настройки эволюционных алгоритмов [17, 72, 86, 100]. Эти методы позволяют с помощью обучения с подкреплением настраивать некоторые

Листинг 1 – Алгоритм Q-обучения с ε-жадной стратегией исследования среды [11]

Require: ε — вероятность выбора случайного действия; α — скорость обучения; γ — дисконтный фактор.

1: Инициализировать $Q(s, a)$ для всех $s \in S$, $a \in A$
2: **while** (не достигнуто условие останова) **do**
3: Получить состояние среды s
4: $p \leftarrow$ случайное вещественное число $\in [0, 1]$
5: **if** ($p \leqslant \varepsilon$) **then**
6: $a \leftarrow \arg\max_a Q(s,a)$
7: **else**
8: $a \leftarrow$ случайное действие $\in A$
9: Применить действие a к среде
10: Получить от среды награду r и состояние s'
11: $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s, a))$

числовые параметры эволюционных алгоритмов, такие как вероятность применения генетических операторов. Однако, насколько известно автору, ни для одного из соответствующих алгоритмов не существует теоретических оценок времени их работы.

Существуют некоторые работы, в которых оценивается скорость сходимости алгоритмов обучения с подкреплением (в основном, скорость сходимости алгоритма Q-обучения при решении ряда модельных задач) [45, 59, 60, 73, 90]. Также существует большое число исследований, в которых оценивается время работы эволюционных алгоритмов при решении различных классов задач, многие из которых перечислены, например, в работах [67, 88]. Однако теоретические работы, анализирующие алгоритмы, совмещающие обучение с подкреплением и эволюционные вычисления, практически отсутствуют [23].

Насколько известно автору, единственной теоретической работой, анализирующей алгоритм, совмещающий обучение с подкреплением и поисковые эвристики, к которым относятся эволюционные алгоритмы, является работа [23]. В ней анализируется время работы поисковых эвристик, выбираемых с помощью обучения с подкреплением. При этом во всех эвристиках используется один и тот же критерий оптимизации, вопрос использования вспомогательных критериев не рассматривается. Этому исследованию предшествует работа [22], в которой используются другие методы обучения, основанные на случайном выборе эвристик, а также на жадном выборе без учета предыстории. Отметим, что первая работа автора настоящей диссертации, в которой приводятся оценки времени работы эволюционного алгоритма, настраиваемого с помощью обучения с подкреплением [29], была опубликована раньше, чем работы [22, 23]. Таким образом, асимптотический анализ времени работы алгоритма, совмещающего эволюционные вычисления и обучение с подкреплением, произведен автором настоящей диссертации впервые.

1.5 Задачи, решаемые в диссертационном исследовании

На основе выполненного обзора можно сформулировать следующие задачи, выполнение которых должно обеспечить сокращение общего времени, необходимого для осуществления автоматизированной оценки эффективности программ, решающих задачи дискретной математики:

1. Разработка и программная реализация метода адаптивного выбора вспомогательных критериев, используемых в эволюционных алгоритмах при генерации тестов для указанного класса программ. В разделе 1.1 показано, что для генерации тестов, позволяющих оценить эффективность работы программ, как правило, используются эволюционные алгоритмы. Из раздела 1.3 следует, что эффективность этих алгоритмов может быть повышена за счет использования вспомогательных критериев. Поэтому данная задача представляется актуальной. Выбор может быть осуществлен

с помощью обучения с подкреплением, которое рассматривается в разделе 1.4.

2. Получение асимптотических оценок времени работы алгоритма, реализующего предложенный метод, для определения его эффективности и корректировка алгоритма в случае необходимости. В разделе 1.4.5 показано, что теоретическая оценка алгоритмов, совмещающих обучение с подкреплением и эволюционные алгоритмы, является открытой задачей. При этом теоретический анализ способствует лучшему пониманию принципов работы алгоритмов и позволяет получить рекомендации по их эффективному применению [88].
3. Экспериментальное исследование предложенного метода и его сравнение с известными методами выбора вспомогательных критериев при генерации тестов, оценивающих эффективность программ решения задач дискретной математики. Выполнение этой задачи позволит оценить эффективность предложенного метода с практической точки зрения, а также степень достижения поставленной цели диссертационного исследования.

Выводы по главе 1

Приведен обзор методов генерации тестов для оценки эффективности программ и обоснована необходимость разработки адаптивных алгоритмов, повышающих эффективность существующего автоматизированного метода, основанного на применении эволюционных алгоритмов. Поставлена задача разработки метода выбора вспомогательных критериев оптимизации в эволюционных алгоритмах, приведен обзор существующих методов использования вспомогательных критериев. Показана целесообразность применения обучения с подкреплением для решения поставленной задачи.

ГЛАВА 2 МЕТОД EA + RL АДАПТИВНОГО ВЫБОРА ВСПОМОГАТЕЛЬНЫХ КРИТЕРИЕВ ОПТИМИЗАЦИИ В ЭВОЛЮЦИОННЫХ АЛГОРИТМАХ

В данной главе предлагается метод EA + RL, позволяющий динамически выбирать, какие критерии оптимизации следует использовать на текущем этапе генерации тестов, оценивающих эффективность программ решения задач дискретной математики. Критерии выбираются из множества, состоящего из вспомогательных и целевого критериев. Выбор производится в соответствии со стратегией, обновляемой с помощью алгоритма обучения с подкреплением. Предлагаемый метод описывается в разделе 2.1, в разделе 2.2 представлены способы реализации метода, в частности, способы вычисления награды и определения состояний ЭА.

2.1 Метод выбора вспомогательных критериев EA + RL

Опишем предлагаемый метод выбора вспомогательных критериев эволюционного алгоритма (англ. evolutionary algorithm, EA). Метод основан на обучении с подкреплением (англ. reinforcement learning, RL). Будем называть предлагаемый метод EA + RL [21, 36, 37].

Введем понятия, необходимые для описания предлагаемого метода. Имеется целевой критерий оптимизации $t : W \to \mathbb{R}$, где W — дискретное пространство поиска решений. Необходимо найти точку глобального оптимума целевого критерия из пространства W. В случае, когда существует несколько точек глобального оптимума, достаточно найти хотя бы одну из них. Далее, говоря о точках оптимума и оптимальных значениях критериев, будем иметь в виду точки глобальных оптимумов и глобальные оптимальные значения, если не указано иное. Оптимизация целевого критерия производится с помощью эволюционного алгоритма. Также имеется конечный набор H вспомогательных критериев $h_i \in H, h_i : W \to \mathbb{R}$. Будем рассматривать задачу максимизации целевого критерия.

Схема работы метода EA + RL представлена на рисунке 2. На каждой итерации i агент обучения выбирает оптимизируемый критерий f_i из множества $H \cup \{t\}$ и передает его ЭА. Выбранный критерий используется в ЭА в качестве функции приспособленности для создания из текущего поколения g_i следующего поколения g_{i+1}, после чего агенту возвращается награда r_i и некоторое представление состояния s_{i+1}, соответствующего следующему поколению. Целью агента является максимизация суммарной награды. Награда зависит от роста целевого критерия в текущем поколении по сравнению с предыдущим поколением. Чем лучше значение целевого критерия в новом поколении, тем больше награда. На основании полученной награды алгоритм обучается тому, какие критерии следует выбирать в дальнейшем.

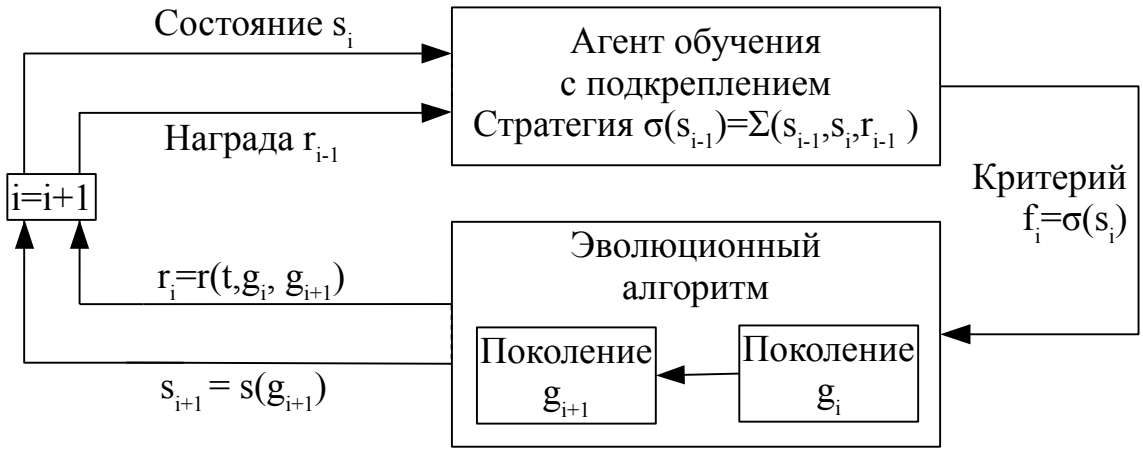

Рисунок 2 – Схема работы метода EA + RL, i – номер итерации, Σ –функция обновления стратегии выбора критериев σ

Подробное описание метода приведено в виде псевдокода в листинге 2. В методе возможно использование различных эволюционных алгоритмов и различных алгоритмов обучения с подкреплением. Поэтому описание является обобщенным, пример реализации метода с использованием конкретных алгоритмов приведен в разделе 2.2.3.

Для работы метода необходимо предоставить целевой критерий t, конечный набор вспомогательных критериев оптимизации H, функцию вычисления

Листинг 2 – Метод EA + RL

Require: t — целевой критерий; H — множество вспомогательных критериев; s — функция вычисления состояния ЭА; r — функция вычисления награды; σ — стратегия агента обучения; Σ — функция обновления стратегии агента обучения; эволюционные операторы.

1: Инициализировать номер итерации $i = 0$
2: Инициализировать стратегию агента обучения σ
3: Инициализировать текущее поколение ЭА g_0
4: **while** (не выполнено условие останова ЭА) **do**
5: Вычислить состояние ЭА: $s_i = s(g_i)$
6: Выбрать критерий $f_i \in H \cup \{t\}$ согласно стратегии агента обучения: $f_i = \sigma(s_i)$
7: Присвоить функции приспособленности критерий f_i
8: Применить эволюционные операторы
9: Сформировать следующее поколение g_{i+1}
10: Вычислить награду: $r_i = r(t, g_i, g_{i+1})$
11: Вычислить новое состояние: $s_{i+1} = s(g_{i+1})$
12: Используя r_i, s_i и s_{i+1}, обновить стратегию агента обучения: $\sigma(s_i)$
13: Обновить номер итерации: $i = i + 1$

состояния s, значение которой зависит от текущего поколения g_i, и функцию вычисления награды r, значение которой зависит от изменения значения целевого критерия между поколениями g_i и g_{i+1}. Подробно варианты реализации функций вычисления награды и состояния обсуждаются в разделах 2.2.1 и 2.2.2 соответственно. Также для работы метода необходимо определить стратегию σ агента обучения, функцию ее обновления Σ и эволюционные операторы. Эти параметры зависят от используемых алгоритмов обучения и ЭА.

В строках 1–3 производится инициализация номера итерации метода, а также инициализация алгоритма обучения и ЭА. Инициализация стратегии σ,

используемой агентом обучения с подкреплением, зависит от того, какой алгоритм обучения используется. Например, для алгоритма Q-обучения инициализация может заключаться в заполнении нулями всех значений в Q-таблице, предназначенной для хранения оценки эффективности выбора критериев в зависимости от текущего состояния. Текущее поколение ЭА, как правило, инициализируется путем генерации случайных особей.

Далее в строке 4 начинается основной цикл метода, который выполняется, пока не достигнуто условие останова ЭА. Как правило, используются такие условия, как нахождение оптимального значения целевого критерия или достижение ограничения на число вычислений функции приспособленности. Определяется состояние s_i, соответствующее текущему поколению, и в соответствии со стратегией агента обучения с подкреплением выбирается критерий f_i, который должен быть использован в этом состоянии на текущей итерации.

В строках 8–9 выполняется основная часть ЭА, в которой в качестве функции приспособленности используется выбранный критерий f_i. К текущему поколению применяются эволюционные операторы, отбираются особи для следующего поколения. Метод не накладывает ограничений на конкретную реализацию операторов и способ отбора особей, эти параметры зависят от используемого ЭА.

В строках 10–13 выполняются действия, необходимые для обновления стратегии агента обучения с подкреплением в соответствии с полученным на текущей итерации опытом. Вычисляется награда за выбор критерия f_i, величина которой зависит от изменения значения целевого критерия в поколении g_i и сформированном в текущей итерации новом поколении g_{i+1}. Чем больше изменилось это значение, тем выше должна быть награда. Благодаря описанному принципу, агент обучения с подкреплением выбирает критерии, ведущие к росту целевого критерия. Таким образом, выполняется максимизация целевого критерия, даже если он не оптимизируется явно.

Помимо награды, вычисляется новое состояние s_{i+1}, соответствующее новому поколению g_{i+1}. Награда, а также информация о новом и старом состояниях используются для обновления стратегии σ агента обучения с подкреплением. Конкретная реализация метода обновления стратегии зависит от используемого алгоритма обучения. Наконец, обновляется номер текущей итерации и цикл повторяется.

Заметим, что помимо выбора вспомогательных критериев, используемых при генерации тестов, оценивающих эффективность решения задач дискретной математики, предложенный метод успешно применялся автором для выбора вспомогательных критериев в задаче составления расписаний [95] и задаче коммивояжера [96]. Также подход, используемый в методе EA + RL, применялся для выбора эволюционных операторов [40].

2.2 Способы реализации метода EA + RL

В данном разделе предлагаются варианты конкретной релизации метода EA + RL, предложенного в разделе 2.1. Приводятся различные формулы вычисления награды и состояния, а также пример реализации метода EA + RL с использованием метода спуска со случайными мутациями и Q-обучения. Эта реализация подвергается теоретическому анализу на примере задач, обладающих свойствами задачи генерации тестов, в главе 3.

2.2.1 Формирование награды

Напомним, что в общем случае, согласно терминологии, принятой в обучении с подкреплением, функция награды имеет вид $R : S \times F \to \mathbb{R}$, где S — множество состояний среды, F — множество действий агента. Как правило, функция награды явно не доступна, однако ее можно оценить с помощью информации о значениях мгновенной награды r. Для краткости, будем называть функцию для вычисления мгновенной награды *функцией вычисления награды*.

Вычисление награды производится после выбора критерия f_i в состоянии s_i и формирования поколения g_{i+1} с использованием критерия f_i в качестве

функции приспособленности. Награда должна быть тем выше, чем выше значение целевого критерия, полученного в результате применения соответствующего критерия оптимизации. В методе EA + RL могут использоваться различные функции награды, удовлетворяющие этому требованию.

В работах [17, 98], посвященных настройке параметров и выбору операторов ЭА с помощью обучения с подкреплением, используется нормированная награда, основанная на значении приспособленности лучшей особи в поколении. В настоящей диссертации предлагается функция вычисления награды, основанная на средней приспособленности особей, составляющих поколение. Это связано с тем, что при генерации тестов число различных значений целевого критерия, как правило, невелико [2]. В данных условиях усредненная награда будет принимать больше различных значений, что позволит лучше различать близкие по эффективности действия. Пусть $B(f,g_i)$ — наибольшее значение критерия f в поколении g_i: $B(f, g_i) = \max_{x \in g_i} f(x)$, где x — особь ЭА. Тогда формула вычисления награды, основанная на работах [17, 98] и адаптированная для использования при генерации тестов, будет иметь следующий вид:

$$r(t, g_i, g_{i+1}) = \frac{\sum_{x \in g_{i+1}} t(x) - \sum_{x \in g_i} t(x)}{\sum_{x \in g_{i+1}} t(x)}. \quad (2.1)$$

Благодаря тому, что значение награды нормируется, она зависит не от абсолютной величины приспособленности по целевому критерию, а от величины относительного прироста целевого критерия. Однако в работе ЭА распространены случаи, когда с приближением к точке оптимума величина изменения приспособленности уменьшается, в то время как само значение приспособленности вблизи точки оптимума достаточно велико. Это может привести к тому, что нормированная награда будет сильно занижена в конце оптимизации и не будет существенно влиять на поведение агента обучения. Если при этом существует вспомогательный критерий, наиболее эффективный на последнем этапе оптимизации, он может быть не найден. Поэтому в настоящей диссертации

предлагается использовать вариант вычисления награды без нормирования:

$$r(t, g_i, g_{i+1}) = \sum_{x \in g_{i+1}} t(x) - \sum_{x \in g_i} t(x). \tag{2.2}$$

2.2.2 Определение состояния эволюционного алгоритма

Рассмотрим простейший случай, когда множество состояний состоит из единственного состояния: $S = \{s\}$. В методе EA + RL можно применять единственное состояние, иными словами, считать, что ЭА всегда находится в одном и том же состоянии с точки зрения взаимодействия с агентом обучения. Применение этого подхода может быть эффективным в случаях, когда во время всего процесса оптимизации наиболее выгодным является один и тот же критерий. Соответствующая формула вычисления состояния выглядит следующим образом:

$$s(g_i) = s. \tag{2.3}$$

Также состояние можно определить как значение целевого критерия, вычисленное на лучшей особи или усредненное для всех особей поколения. Приведем формулу состояния, соответствующего значению целевого критерия на лучшей особи, воспользовавшись обозначениями из раздела 2.4:

$$s(g_i) = B(t, g_i). \tag{2.4}$$

Эффективность предложенных типов состояний сравнивается на основе результатов экспериментов по генерации тестов для оценки эффективности программ решения задач дискретной математики в главе 4.

Аналогично определению награды, состояние также можно определить на основе средней приспособленности особей поколения. Однако в данной работе рассматриваются алгоритмы обучения с подкреплением для конечного дискретного множества состояний. Поэтому невозможно использовать значение средней приспособленности, которое не является целым числом, в качестве состояния. Можно использовать сумму приспособленностей, однако этот подход

приведет к большому числу состояний, что, в свою очередь, может замедлить процесс обучения, так как агент будет проводить меньше итераций в каждом из состояний. В связи с перечисленными причинами, состояние, основанное на приспособленности поколения, не рассматривается.

Как состояние, так и награду можно было бы определять на основе значений не только целевого, но и вспомогательных критериев. Однако при решении задачи оценки эффективности программ вспомогательные критерии генерируются автоматически и не существует гарантий по поводу их поведения. Некоторые критерии могут, например, оказаться константами. Поэтому для определения награды и состояния предлагается ограничиться целевым критерием — временем работы программы, значение которого заведомо релевантно, так как решается задача его максимизации.

2.2.3 Алгоритмы, реализующие метод EA + RL

В программной реализации метода EA + RL могут быть использованы различные эволюционные алгоритмы и алгоритмы обучения с подкреплением. В данной работе были выполнены реализации, основанные на методе спуска со случайными мутациями (англ. Random Local Search, RLS) и популяционном генетическом алгоритме. Эффективность первой реализации анализируется в главе 3. На основе этого анализа делаются выводы для более сложной реализации, основанной на популяционном алгоритме. Популяционный алгоритм используется в практической части работы в главе 4. Также подход, используемый в методе EA + RL, применялся автором для выбора операторов в алгоритмах на основе искусственных иммунных систем [35].

В качестве алгоритмов обучения с подкреплением использовались Q-обучение (англ. *Q-learning*), R-обучение (англ. *R-learning*), отложенное Q-обучение (англ. *Delayed Q-learning*). Эти алгоритмы сравниваются с точки зрения их эффективности для использования в методе EA + RL при решении задачи генерации тестов в главе 4. Также автором проводились исследования, в

которых в методе EA + RL использовалось многокритериальное обучение с подкреплением [41].

Будем пользоваться следующей нотацией. Пусть в методе EA + RL в качестве эволюционного алгоритма используется алгоритм A, а в качестве алгоритма обучения с подкреплением алгоритм B. Тогда конкретную реализацию метода EA + RL будем обозначать как A + B. Например, алгоритм, основанный на методе спуска со случайными мутациями RLS и Q-обучении будет называться *RLS + Q-learning*.

В листинге 3 представлен псевдокод алгоритма *RLS + Q-learning*, использующегося в теоретическом анализе EA + RL в главе 3.

Листинг 3 – Алгоритм *RLS + Q-learning*, реализующий метод EA + RL с использованием жадного Q-обучения и метода спуска со случайными мутациями

Require: t — целевой критерий; H — множество вспомогательных критериев; S — множество состояний; α — скорость обучения; γ — дисконтный фактор.

1: Инициализировать номер итерации $i = 0$
2: Инициализировать $Q(s, f)$ нулями для всех $s \in S$, $f \in H \cup \{t\}$
3: Инициализировать текущую особь: x_0
4: **while** ($t(x) \neq \max$) **do**
5: Вычислить состояние среды $s_i = t(x_i)$
6: Выбрать критерий: $f_i = \arg\max_f Q(s_i, f)$
7: Применить оператор мутации: $y = \text{mutate}(x_i)$
8: **if** $f(y) \geqslant f(x_i)$ **then**
9: $x_{i+1} \leftarrow y$
10: Вычислить награду: $r_i = t(x_{i+1}) - t(x_i)$
11: Вычислить новое состояние: $s_{i+1} = t(x_{i+1})$
12: $Q(s_i,f_i) = Q(s_i,f_i) + \alpha(r_i + \gamma \max_f Q(s_{i+1},f) - Q(s_i, f_i))$

В качестве функции вычисления награды в алгоритме *RLS* + *Q-learning* используется ненормированная функция (2.2), состояния основаны на значениях целевого критерия в соответствии с формулой (2.4). В качестве стратегии агента обучения применяется жадная стратегия из алгоритма Q-обучения: выбирается критерий с максимальной оценкой эффективности (строка 6 листинга 3). Формула обновления стратегии агента Σ определяется формулой обновления стратегии в Q-обучении (строка 12).

Поколение, в соответствии с алгоритмом RLS, состоит из одной особи. Из эволюционных операторов используется только оператор мутации. В алгоритме RLS это локальный оператор, производящий небольшие изменения в генотипе особи. Формирование следующего поколения производится на основе сравнения значения выбранного критерия f_i, вычисленного на особи до мутации и после (строка 8).

Реализация метода EA + RL зависит также и от решаемой задачи, в соответствии с которой определяются целевой и вспомогательные критерии, выбирается представление особей ЭА и эволюционные операторы. Примеры конкретных реализаций для модельных задач, определенных на битовых строках, и для задачи генерации тестов приведены в главах 3 и 4.

Выводы по главе 2

Предложен метод выбора вспомогательных критериев оптимизации, используемых в эволюционном алгоритме при генерации тестов, производимой с целью оценки эффективности программ решения задач дискретной математики. Метод основан на обучении с подкреплением. Также предложены конкретные варианты его реализации, включающие формулы вычисления награды и состояния. Приведен пример реализации, использующей конкретные алгоритмы оптимизации и обучения. Результаты, описанные в главе, опубликованы в статьях из списка ВАК [1, 3, 12], а также в трудах международных конференций, индексируемых в системе *Scopus* [20, 21, 36–38].

ГЛАВА 3 АСИМПТОТИЧЕСКИЕ ОЦЕНКИ ВРЕМЕНИ РАБОТЫ АЛГОРИТМА, РЕАЛИЗУЮЩЕГО МЕТОД EA + RL

В данной главе излагается теоретический анализ предложенного метода. В начале главы приводится краткий обзор ее содержания, включающий перечисление основных теоретических результатов и объяснение значения этих результатов с точки зрения применимости метода EA + RL к генерации тестов для оценки эффективности программ решения задач дискретной математики. В последующих разделах приведены подробные доказательства полученных результатов.

3.1 Краткое описание полученных теоретических результатов

В ходе теоретического анализа моделируются различные ситуации, возникающие при автоматической генерации тестов, используемых для оценки эффективности программ. Целью диссертации является уменьшение общего времени поиска сложного теста. В рассматриваемых в данной главе модельных задачах вычисление ФП для каждой особи занимает одно и то же время. Поэтому в данном разделе общее время оптимизации целевого критерия определяется числом вычислений ФП.

Для отражения свойств вспомогательных критериев, применяемых при генерации тестов, в данной главе используются функции вида $\{0,1\}^n \to \mathbb{Z}$, действующие из множества битовых строк длины n в множество целых чисел. Битовая строка z представляется как последовательность нулей и единиц длины n :

$$z = z_1 z_2 \dots z_n, \text{ где } z_i \in \{0, 1\}. \tag{3.1}$$

Определим используемые функции: OneMax — число бит, равных единице:

$$\text{OneMax}(z) = \sum_{i=1}^{n} z_i; \tag{3.2}$$

ZeroMax — число бит, равных нулю:

$$\text{ZeroMax}(z) = n - \text{OneMax}(z); \tag{3.3}$$

XdivK — «загрубленная» версия OneMax — функция, подсчитывающая, сколько раз по k единиц встречается в строке:

$$\text{XdivK}(z) = \left\lfloor \frac{\text{OneMax}(z)}{k} \right\rfloor, k \in \mathbb{Z}, n \bmod k = 0. \tag{3.4}$$

Описываемые в данной главе теоретические результаты получены на основе вычисления математического ожидания числа переходов в марковских цепях, моделирующих процесс оптимизации соответствующих функций алгоритмом *RLS + Q-learning*, представленном в листинге 3 главы 2. С целью получения верхних оценок текущая особь x инициализируется битовой строкой, состоящей из n нулей. В качестве оператора мутации применяется инвертирование одного бита, выбираемого случайным образом. Состояние определяется значением целевого критерия.

Чтобы получить общее представление о природе вспомогательных критериев, которые предлагается использовать при генерации тестов, приведем упрощенный пример программы, в код которой вставлены счетчики, используемые для вычисления значений этих критериев. В листинге 4 приведен псевдокод, отражающий один из возможных подходов к решению задачи «Ships. Version 2» [121], которая подробно рассматривается в главе 4. В листинге 4 имеется два счетчика: C_0 — число вызовов внешнего цикла **while** и C_1 — общее число вызовов рекурсивной процедуры внутри этого цикла. Целевым критерием является время работы программы, значения вспомогательных критериев соответствуют значениям счетчиков C_0 и C_1.

В ходе автоматического встраивания в код тестируемой программы счетчиков, значения которых используются в качестве вспомогательных критериев, могут возникать *мешающие критерии*: при оптимизации такого критерия получаемые тесты отдаляются в пространстве поиска от искомого сложного теста, что может вызывать увеличение общего времени, необходимого для его нахождения.

Листинг 4 – Упрощенный пример псевдокода программы, решающей задачу «Ships. Version 2», со вставленными счетчиками

$C_0 \leftarrow 0, C_1 \leftarrow 0, \text{last} \leftarrow 0$
Считать входные данные (тест)
while (пока решение не найдено) **do**
    Случайным образом перемешать входные данные
    $\text{last} \leftarrow 0$
    Вызвать рекурсивную процедуру поиска решений
    На каждом вызове этой процедуры увеличить счетчик: $\text{last} \leftarrow \text{last} + 1$
    if (решение найдено) **then**
        Вывести ответ
    else
        $C_0 \leftarrow C_0 + 1, C_1 \leftarrow C_1 + \text{last}, \text{last} \leftarrow 0$

Рассмотрим пример мешающего критерия, который может возникнуть при генерации тестов. Существуют задачи, время решения которых, при определенных конфигурациях входных данных, начинает уменьшаться с увеличением их размера. К таким задачам относится, например, задача о выполнимости (*SAT*) [84]. Счетчик, вставленный в цикл, обрабатывающий входные данные в программе, решающей такую задачу, будет соответствовать мешающему критерию: чем больше значение счетчика, тем больше входных данных было считано, а значит, тем быстрее будет работать программа. Иными словами, увеличение вспомогательного критерия, являющегося мешающим, в этом случае приводит к уменьшению целевого.

Для осуществления теоретического анализа влияния мешающего критерия используются функции OneMax и ZeroMax. Эти функции соотносятся как целевой и мешающий критерии в приведенном примере: функция ZeroMax является мешающим критерием по отношению к OneMax. В следующей теореме отраже-

на способность метода EA + RL уменьшать отрицательное влияние мешающего критерия:

Теорема 1. *Алгоритм* RLS + Q-learning *находит оптимальное значение целевого критерия* OneMax *при наличии мешающего критерия* ZeroMax *асимптотически за то же число вычислений функции приспособленности, что и метод спуска со случайными мутациями без мешающего критерия. Это число вычислений составляет* $\Theta(n \log n)$*, где* n *— размерность задачи.*

Доказательство теоремы приведено в разделе 3.2.

Рассмотрим теперь случай с *эффективным критерием*: использование такого критерия позволяет сократить число вычислений ФП, необходимое для нахождения сложного теста. Заметим, что, вследствие особенностей измерения времени на компьютере, время работы тестируемой программы принимает ограниченное число значений, пропорциональных определенному временному кванту [2]. При этом счетчики, встраиваемые в код тестируемой программы, могут принимать большее число различных значений. Вспомогательные критерии, основанные на этих счетчиках и коррелирующие со временем работы программы, являются эффективными, так как позволяют эволюционному алгоритму получить больше информации для сравнения особей.

В теоретическом анализе в качестве целевого критерия рассматривается функция XdivK, значения которой, подобно значениям времени работы тестируемой программы, пропорциональны определенной постоянной k. В качестве эффективного вспомогательного критерия рассматривается функция OneMax, коррелирующая с XdivK и принимающая большее число различных значений. В теореме 2 отражена способность метода EA + RL находить оптимум целевого критерия за асимптотически меньшее число вычислений ФП при наличии эффективного вспомогательного критерия:

Теорема 2. *Алгоритм* RLS + Q-learning *находит оптимальное значение целевого критерия* XdivK *при наличии эффективного критерия* OneMax *асимпто-*

тически быстрее в зависимости от k, чем метод спуска со случайными мутациями: $\frac{T_{RLS}}{T_{\text{RLS + Q-learning}}} \geqslant 2^{k-2}(1 - o(1))$.

Доказательство теоремы приведено в разделе 3.3.

Наконец, учтем возможность наличия как мешающего, так и эффективного критерия. Пусть целевым критерием, как и в предыдущем случае, является XdivK. В качестве вспомогательных критериев рассмотрим эффективный критерий OneMax и мешающий критерий ZeroMax.

В разделе 3.4 показано на основе результатов экспериментов, что алгоритм *RLS + Q-learning* с эффективным и мешающим критериями при определенных значениях параметров обучения не находит оптимум XdivK или находит его дольше, чем алгоритм *RLS + Q-learning* без вспомогательных критериев. В связи с этим предлагается *модификация алгоритма* RLS + Q-learning с сохранением особи с лучшим полученным значением целевого критерия. Способность модифицированного алгоритма снижать влияние мешающего критерия подтверждается теоремой 3.

Теорема 3. *Модификация алгоритма* RLS + Q-learning *находит оптимальное значение целевого критерия* XdivK *при наличии вспомогательных критериев* OneMax *и* ZeroMax *асимптотически за то же число вычислений ФП, что метод спуска со случайными мутациями без вспомогательных критериев.*

Доказательство теоремы приведено в разделе 3.4.

Способность модификации алгоритма *RLS + Q-learning* при определенных значениях ее параметров использовать эффективный критерий для снижения числа вычислений функции приспособленности подтверждается экспериментально. Результаты экспериментов, приведенные в разделе 3.4, показывают, что эта модификация работает наиболее эффективно с использованием состояния, равного значению целевого критерия, и нулевой вероятности исследования среды. Она позволяет найти оптимальное значение XdivK при наличии эффективного и мешающего критериев за в несколько раз меньшее число вычисле-

ний функции приспособленности, чем метод спуска со случайными мутациями. Таким образом, делается вывод о целесообразности использования метода EA + RL с сохранением лучшей особи для генерации тестов.

3.2 Задача с мешающим критерием

В данной главе рассматривается задача максимизации целевого критерия OneMax с мешающим вспомогательным критерием ZeroMax с помощью алгоритма *RLS* + *Q-learning*, реализующего метод EA + RL. Для удобства переобозначим целевой критерий как $f_1(x) = \text{OneMax}(x)$ и вспомогательный критерий как $f_0(x) = \text{ZeroMax}(x)$. Заметим, что при увеличении критерия f_0 на некоторую величину целевой критерий f_1 уменьшается на такую же величину. Таким образом, критерий f_0 является мешающим. Используется состояние, определяемое значением целевого критерия на текущей особи на i-ой итерации: $s = f_1(x_i)$. Награда равна разности значений целевого критерия на новой особи x_{i+1} и на исходной особи x_i: $r = f_1(x_{i+1}) - f_1(x_i)$.

3.2.1 Схема доказательства

В разделе 3.2.2 формулируется и доказывается вспомогательное утверждение, из которого следует, что агент, выбрав мешающий критерий, может обучиться не выбирать его в дальнейшем. Будем называть это утверждение *леммой об обучении*. На основе этой леммы в разделе 3.2.3 строится цепь Маркова, моделирующая процесс оптимизации, осуществляемый методом EA + RL. Получившаяся цепь, помимо основного пути, вдоль которого ФП возрастает от начального значения к оптимальному, содержит альтернативные пути, что усложняет ее анализ. В разделе 3.2.4 вычисляется математическое ожидание числа итераций метода EA + RL, необходимого для прохождения по фрагменту альтернативного пути. Полученное выражение позволяет построить упрощенную цепь, в которой фрагменты альтернативных путей заменены на ребра соответствующей длины, отличной от единицы.

На основе упрощенной цепи в разделе 3.2.5 вычисляется математическое ожидание числа итераций EA + RL, необходимое для максимизации целевого критерия. Полученное выражение сравнивается с аналогичным выражением для метода спуска со случайными мутациями при оптимизации OneMax. На основе сравнения делается вывод о том, что асимптотика времени работы метода EA + RL при решении рассматриваемой задачи с мешающим критерием совпадает с асимптотикой времени работы метода спуска со случайными мутациями при решении задачи OneMax без мешающего критерия. Таким образом, применение метода EA + RL позволяет успешно игнорировать мешающий критерий в рассмотренной задаче.

3.2.2 Лемма об обучении

Лемма 1. *Пусть агент обучения посещает состояние s и покидает его. Тогда при всех последующих посещениях состояния s агент будет выбирать эффективный критерий f_1.*

Рассмотрим случай, когда состояние s посещается в первый раз. В этом случае $Q(s, f_0) = Q(s, f_1) = 0$, так как изначально значения Q проинициализированы нулями и агент еще не посещал состояние s, а значит, не мог изменить начальные значения. Таким образом, равновероятно может быть выбран как критерий f_0, так и критерий f_1. В каждом из этих случаев оператор мутации может инвертировать либо единичный бит, либо нулевой. Следовательно, из состояния s возможны четыре вида переходов, как показано на рисунке 3.

Рассмотрим четыре описанных случая:

– Выбирается критерий f_0, после чего оператор мутации инвертирует ноль в единицу. Поскольку в полученной особи число нулей уменьшилось, она уступает исходной особи с точки зрения выбранного критерия. Полученная особь отвергается, алгоритм остается в состоянии s. Поскольку в итоге особь не изменилась, награда оказывается нулевой и значения Q не меняются.

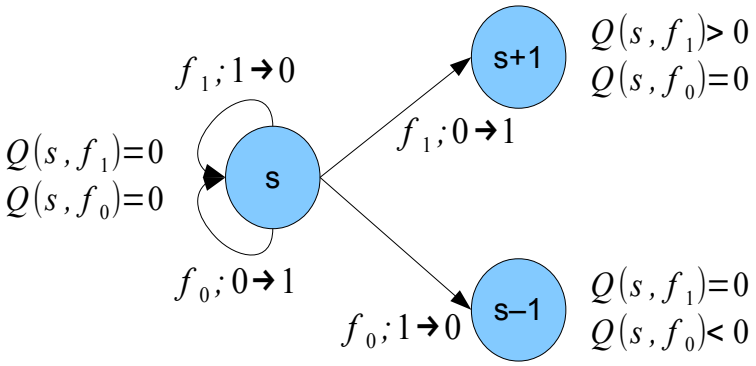

Рисунок 3 – Иллюстрация к доказательству леммы. Способ вычисления Q определен в листинге; s — состояние среды; $f_1 = \text{OneMax}$, $f_2 = \text{ZeroMax}$ — выбранные критерии; $a \to b$ — мутация бита a в бит b

– Выбирается критерий f_0, после чего оператор мутации инвертирует единицу в ноль. Поскольку полученная особь обладает большим значением f_0, чем исходная особь, полученная особь становится текущей, и алгоритм переходит в состояние $s - 1$. Так как значение целевого критерия f_1 при этом уменьшается на единицу, значение награды отрицательно. Таким образом, значение $Q(s, f_0)$ становится отрицательным после обновления, значение $Q(s, f_1)$ остается равным нулю.
– Выбирается критерий f_1, после чего оператор мутации инвертирует ноль в единицу. Поскольку полученной особи соответствует большее значение f_1, она становится текущей, и алгоритм переходит в состояние $s + 1$. Следовательно, награда положительна, и значение $Q(s, f_1)$ также становится положительным, а значение $Q(s, f_0)$ остается равным нулю.
– Выбирается критерий f_1, после чего оператор мутации инвертирует единицу в ноль. Поскольку полученной особи соответствует меньшее значение f_1, она будет отвергнута, текущая особь не изменится, и алгоритм останется в состоянии s. Аналогично первому случаю, значение награды будет нулевым, а значения Q не изменятся.

В двух из рассмотренных выше случаях алгоритм покидает состояние s, что соответствует формулировке леммы. В обоих случаях выполняется нера-

венство $Q(s, f_1) > Q(s, f_0)$, так как одно из значений Q нулевое, а другое положительное или отрицательное. При следующем посещении состояния s агент выберет эффективный критерий f_1, потому что ему соответствует большее значение Q. Неравенство $Q(s, f_1) > Q(s, f_0)$ останется истинным, так как агент при применении f_1 будет получать либо нулевую, либо положительную награду, которая может только увеличить разрыв между $Q(s, f_1)$ и $Q(s, f_0)$. Таким образом, эффективный критерий f_1 будет выбираться при всех последующих посещениях состояния s.

3.2.3 Цепь Маркова

Рассмотрим цепь Маркова, соответствующую процессу оптимизации, выполняемому с помощью описанного алгоритма (рисунок 4). Числа, написанные в вершинах цепи, соответствуют числу единиц в текущей особи. Первая особь полностью состоит из нулей, поэтому стартовая вершина содержит значение ноль.

Цепь состоит из трех групп вершин. Первая группа состоит из вершин A_i, которые посещаются впервые, переход в них осуществляется из вершин, расположенных ниже (соответствующих меньшему числу единиц). Вторая группа состоит из вершин B_i, посещаемых при выборе мешающего критерия f_0 и мутации, инвертирующей единицу в ноль. Переход в такие вершины соответствует переходу вниз. Третья группа состоит из вершин C_i, посещаемых при «восстановлении» после перехода вниз. Переход в такие вершины соответствует переходу вверх из вершин второй группы.

Отметим, что, согласно лемме 1, переходы вниз из вершин второй и третьей групп невозможны, так как ранее был осуществлен выход из соответствующих состояний. Поэтому в таких вершинах агент всегда будет выбирать эффективный критерий f_1 и перемещаться в вершины, соответствующие большему числу единиц.

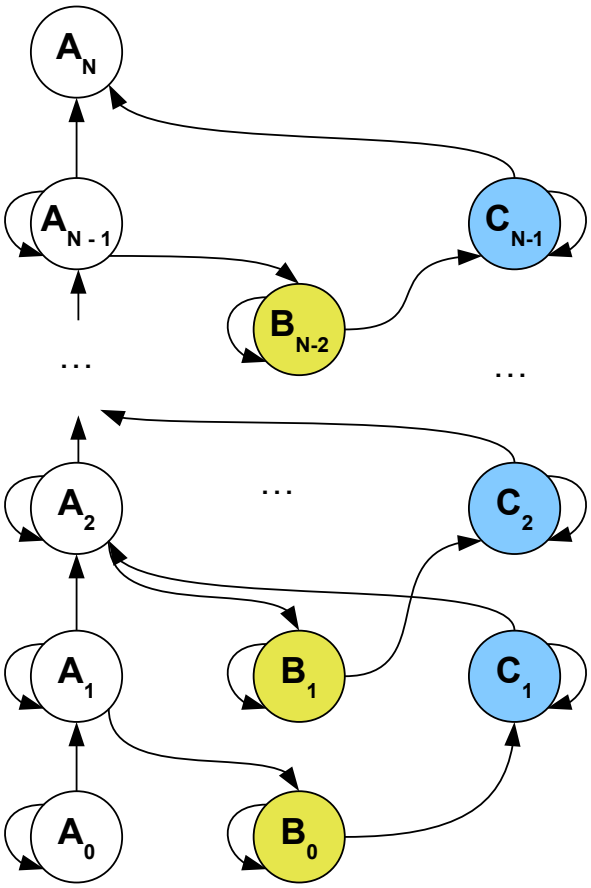

Рисунок 4 – Марковская цепь процесса оптимизации OneMax с мешающим критерием ZeroMax. Обозначения вершин: A_i — посещаемые впервые, B_i — посещаемые повторно в результате перехода вниз, C_i — посещаемые повторно в результате восстановления после перехода вниз; i — число единиц в особи

Следует отличать вершины цепи Маркова от состояний эволюционного алгоритма, введенных ранее. Состояние однозначно определяется числом единиц в текущей особи. Вершина цепи определяется не только числом единиц в текущей особи, но и принадлежностью к одной из трех описанных выше групп.

3.2.4 Упрощение цепи Маркова

В этом разделе производится приведение цепи Маркова к удобному для дальнейшего анализа виду. Необходимо заменить альтернативные пути, проходящие через вершины второй и третьей группы, на ребра, имеющие соответствующую длину и соединяющие вершины первой группы. Длина ребер будет

соответствовать математическому ожиданию числа вычислений функции приспособленности, необходимого для прохода по заменяемому пути.

Заметим, что все заменяемые пути имеют одинаковую структуру. Рассмотрим один такой путь, представленный на рисунке 5. Рядом с каждым переходом (ребром) на рисунке указан выбранный критерий оптимизации и результат применения оператора мутации.

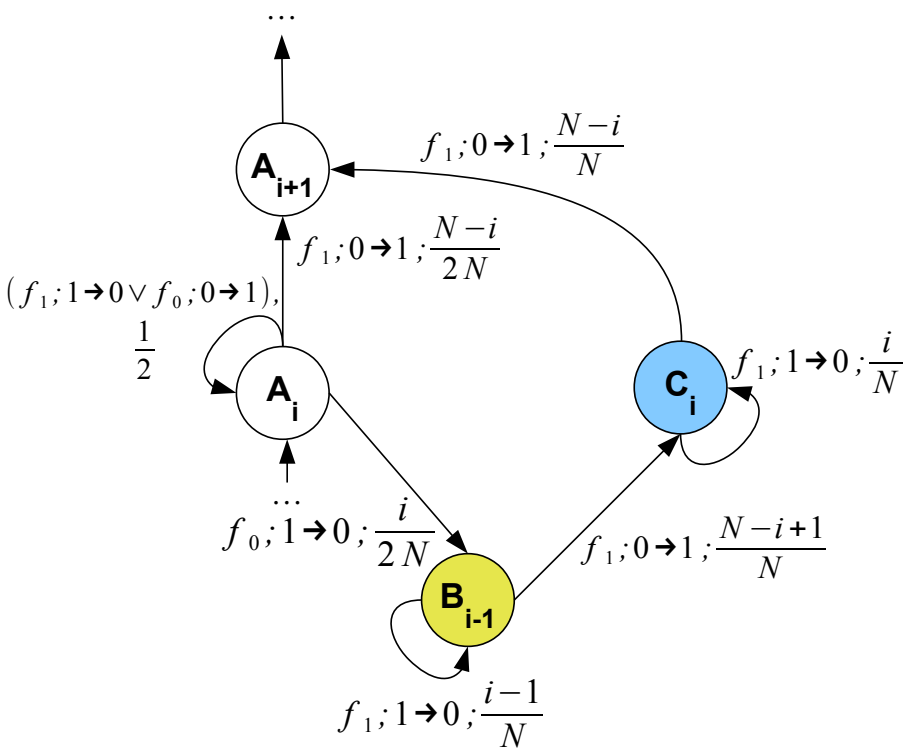

Рисунок 5 – Вероятности (дроби) переходов в марковской цепи. Значения условных обозначений такие же, как на рисунках 3 и 4

Вычислим математическое ожидание числа вычислений функции приспособленности, необходимого для перехода из вершины B_{i-1} в вершину A_{i+1} через промежуточную вершину C_i. Для этого сначала вычислим математическое ожидание числа вычислений функции приспособленности для перехода из B_{i-1} в C_i:

$$E(B_{i-1} \to C_i) = 1 \times \frac{n-i+1}{n} + (1 + E(B_{i-1} \to C_i)) \times \frac{i-1}{n} = \frac{n}{n-i+1}.$$

Затем вычислим математическое ожидание числа вычислений функции приспособленности для перехода из C_i в A_{i+1}:

$$E(C_i \to A_{i+1}) = 1 \times \frac{n-i}{n} + (1 + E(C_i \to A_{i+1})) \times \frac{i}{n} = \frac{n}{n-i}.$$

Наконец, получим искомую величину, просуммировав промежуточные результаты:

$$E(B_{i-1} \to A_{i+1}) = E(B_{i-1} \to C_i) + E(C_i \to A_{i+1}) = \frac{n}{n-i+1} + \frac{n}{n-i}.$$

Теперь стало возможным заменить рассмотренный альтернативный путь из вершины A_i в вершину A_{i+1} на ребро соответствующей длины. Вероятность прохода по этому ребру равна вероятности выбора ребра $A_i \to B_{i-1}$ из замененного альтернативного пути, которая составляет $p = \frac{i}{2n}$.

В предположении, что ребро $A_i \to B_{i-1}$ выбрано с вероятностью p, его длина равна единице. Соответственно, длина ребра между A_i и A_{i+1} составляет $1 + E(B_{i-1} \to A_{i+1})$, как показано на рисунке 6. Заменяя все альтернативные пути аналогичным образом, получаем упрощенную марковскую цепь.

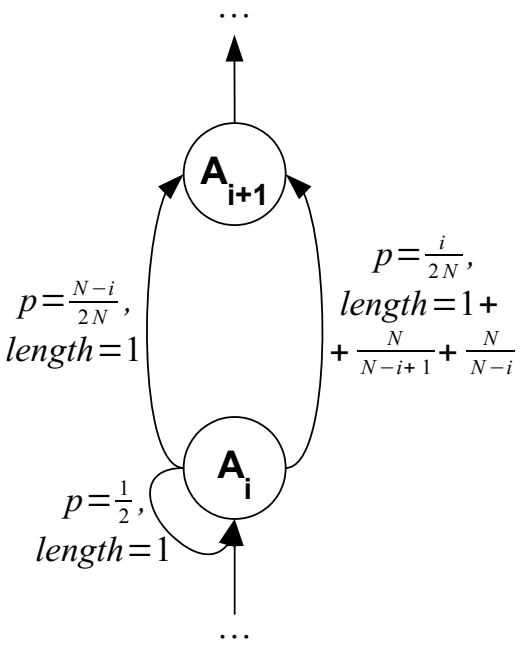

Рисунок 6 – Часть упрощенной марковской цепи

3.2.5 Асимптотическая оценка числа итераций метода EA + RL

В данном разделе вычисляется математическое ожидание числа вычислений функции приспособленности, необходимого для прохода по всем вершинам марковской цепи, начиная с вершины A_0 до вершины A_n. Другими словами, вычисляется математическое ожидание времени выполнения алгоритма *RLS* + *Q-learning*, реализующего метод EA + RL.

Пусть $Z(i)$ — математическое ожидание числа вычислений функции приспособленности, необходимого для перехода из вершины с номером i в вершину с номером $i+1$. Используя вероятности переходов и длины ребер, указанные на рисунке 6, получаем следующее уравнение:

$$Z(i) = \frac{1 + Z(i)}{2} + \frac{n-i}{2n} + \left(1 + \frac{n}{n-i+1} + \frac{n}{n-i}\right) \times \frac{i}{2n}.$$

Решая приведенное выше уравнение, получаем выражение для $Z(i)$ в замкнутой форме:

$$Z(i) = 2 + \frac{i}{n-i+1} + \frac{i}{n-i}. \tag{3.5}$$

Отметим, что начальная вершина имеет вид, отличный от остальных. Рассмотрим соответствующий случай, изображенный на рисунке 7, отдельно.

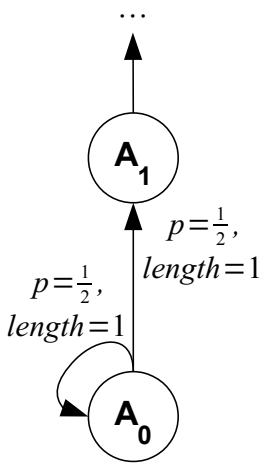

Рисунок 7 – Первые две вершины упрощенной марковской цепи

Аналогично, решим уравнение для $Z(0)$, чтобы получить выражение для $Z(0)$ в замкнутой форме:

$$Z(0) = 1 \times \frac{1}{2} + (1 + Z(0)) \times \frac{1}{2} = 2.$$

Заметим, что подставляя $i = 0$ в выражение 3.5, также получим $Z(0) = 2$. Таким образом, выражение 3.5 верно и для вершины $Z(0)$.

Вычислим математическое ожидание общего числа вычислений функции приспособленности, необходимого для перехода из вершины с номером 0 в вершину с номером n как сумму $Z(i)$:

$$T_{EA+RL}(n) = \sum_{i=0}^{n-1} Z(i) = \sum_{i=0}^{n-1} \left(2 + \frac{i}{n-i+1} + \frac{i}{n-i}\right). \tag{3.6}$$

Приведем асимптотическую оценку полученного результата. Рассмотрим математическое ожидание числа вычислений функции приспособленности, необходимое для решения задачи OneMax в традиционной постановке (без вспомогательных критериев) с помощью метода спуска со случайными мутациями [53]:

$$T_{EA}(n) = \sum_{i=0}^{n-1} \left(1 + \frac{i}{n-i}\right); \tag{3.7}$$

$$T_{EA}(n) = \Theta(n \log n). \tag{3.8}$$

Проанализируем аргумент суммы из выражения 3.6, соответствующего времени работы метода EA + RL:

$$T_{EA+RL}(n) = \sum_{i=0}^{n-1} \left(2 + \frac{i}{n-i+1} + \frac{i}{n-i}\right);$$

$$1 + \frac{i}{n-i} < 2 + \frac{i}{n-i+1} + \frac{i}{n-i} < 2 + 2 \cdot \frac{i}{n-i} = 2 \cdot (1 + \frac{i}{n-i}).$$

Сравнивая неравенства, приведенные выше, со временем решения задачи OneMax в традиционной постановке (выражения 3.7, 3.8), получаем верхнюю и нижнюю границы на время работы метода EA + RL:

$$T_{EA}(n) < T_{EA+RL}(n) < 2 \cdot T_{EA}(n);$$

$$T_{EA+RL}(n) = \Theta(n \log n).$$

Таким образом, удалось показать, что время, необходимое для решения модифицированной задачи OneMax с мешающим критерием с помощью алгоритма *RLS* + *Q-learning*, реализующего метод EA + RL, составляет $\Theta(n \log n)$, что асимптотически совпадает со временем решения традиционной задачи OneMax без вспомогательных критериев. Теорема 1 доказана, полученный результат подтверждает, что метод EA + RL позволяет успешно игнорировать мешающий критерий.

3.3 Задача с эффективным критерием

В данной главе рассматривается задача максимизации целевого критерия XdivK с эффективным вспомогательным критерием OneMax с помощью алгоритма *RLS* + *Q-learning*, реализующего метод EA + RL. Для удобства введем краткие обозначения часто использующихся в анализе объектов: пусть $x = \text{OneMax}(z)$ — число единиц в битовой строке z. Определим OneMax как функцию от числа единиц: $\text{OneMax}(x) = x$. Таким образом, значение вспомогательного критерия OneMax совпадает с x — далее в разделе будем обозначать вспомогательный критерий этой буквой. Обозначим длину битовой строки как n. Требуется максимизировать целевой критерий t, $t(x) = \text{XdivK}(x) = \lfloor \frac{x}{k} \rfloor$, где k — целочисленный параметр, $n \bmod k = 0$.

Заметим, что оптимизация по вспомогательному критерию позволяет достичь оптимума целевого критерия за меньшее число вычислений функции приспособленности. Пусть есть две битовые строки y и z с числом единиц a и b соответственно, $a < b$ (рисунок 8). Пусть также с точки зрения целевого критерия эти строки не различаются. Однако строка z находится ближе к следующему значению XdivK, чем строка y. Вспомогательный критерий x ускоряет

процесс оптимизации целевого критерия за счет того, что он позволяет выбрать строку z. Таким образом, критерий x является эффективным.

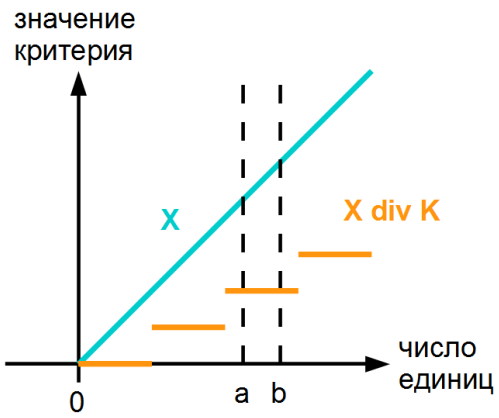

Рисунок 8 – Целевой и вспомогательный критерии в задаче XdivK

Далее на примере описанной модельной задачи будет произведено сравнение времени работы метода EA + RL и метода спуска со случайными мутациями, не использующего вспомогательные критерии. Требуется показать, что метод EA + RL позволит использовать преимущества эффективного вспомогательного критерия и найти оптимальное значение целевого критерия за меньшее число итераций.

Напомним, что в качестве обучения с подкреплением используется алгоритм ε-жадного Q-обучения [107], поддерживающий текущую оценку ожидаемой награды, которая обновляется в соответствии с формулой: $Q(s,f) = Q(s,f)+\alpha(r+\gamma \max_{f'} Q(s',f')-Q(f,a))$, где f — выбранный критерий, состояние s — значение целевого критерия, награда r — разность значений целевого критерия на двух последовательных итерациях. Изначально Q инициализируется нулями.

3.3.1 Схема доказательства

Представим процесс оптимизации как цепь Маркова (рисунок 9). Общий вид цепи одинаков как для метода EA + RL, так и для метода спуска со случайными мутациями. Вершины цепи соответствуют числу единиц в строке. Цветом

выделены вершины, в которых x нацело делится на k. Заметим, что из этих вершин отсутствуют переходы в вершины с меньшим значением x, так как эволюционный алгоритм выбирает строки с большим или равным значением текущего критерия, и оба используемых критерия в этих вершинах отвергают мутации, инвертирующие единицы в нули.

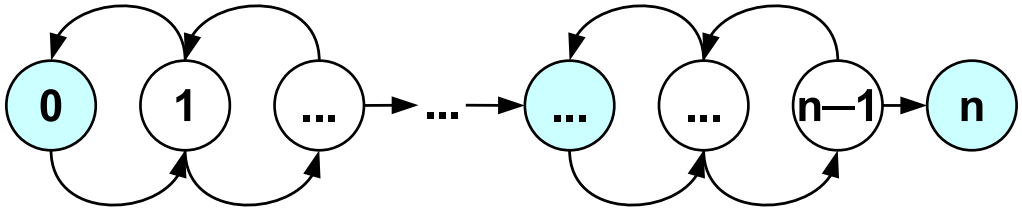

Рисунок 9 – Цепь Маркова, соответствующая процессу оптимизации в задаче XdivK

Рассмотрим число итераций, необходимое для перехода из вершины x в вершину $x+1$. Пусть $Z_E(x)$ — математическое ожидание числа итераций метода спуска, $Z_R(x)$ — математическое ожидание числа итераций метода EA + RL. На рисунках 10 и 11 для метода спуска и метода EA + RL рядом с каждым переходом цепи Маркова подписаны выбранный критерий, мутация и вероятность соответствующего перехода.

Оценим вероятности переходов в случае использования метода спуска (рисунок 10). Для состояний, в которых x делится нацело на k ($x \bmod k = 0$), выполняется соотношение:

$$Z_E(x) = \frac{n-x}{n} + \frac{x}{n} \cdot (1 + Z_E(x)) = \frac{n}{n-x}, \tag{3.9}$$

в то время как для остальных состояний выполняется:

$$\begin{aligned} Z_E(x) &= \frac{n-x}{n} + \frac{x}{n} \cdot (1 + Z_E(x-1) + Z_E(x)) = \\ &= \frac{n}{n-x} + Z_E(x-1) \cdot \frac{x}{n-x}. \end{aligned} \tag{3.10}$$

Рассмотрим теперь вероятности переходов в случае использования метода EA + RL, выбирающего на каждой итерации вспомогательный или целевой

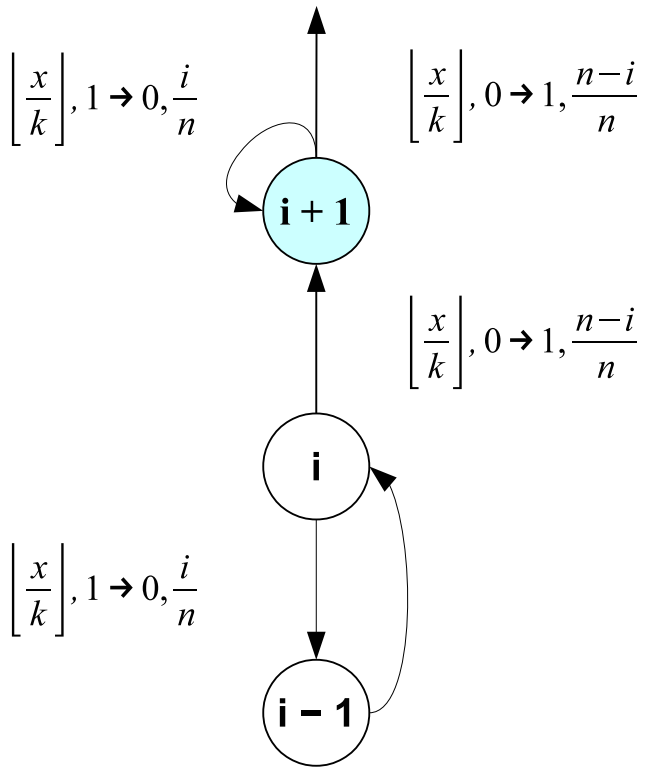

Рисунок 10 – Вероятности переходов при решении задачи XdivK без вспомогательных критериев

критерий и передающий этот критерий в метод спуска со случайными мутациями (рисунок 11). В этом случае вероятность перехода в состояние с меньшим числом единиц ниже, так как при выборе вспомогательного критерия выбор строки с меньшим числом единиц осуществляться не будет.

Для состояний, в которых x делится нацело на k ($x \bmod k = 0$), выполняется соотношение:

$$Z_R(x) = \frac{n-x}{n} + \frac{x}{n} \cdot (1 + Z_R(x)) = \frac{n}{n-x}, \tag{3.11}$$

в то время как для остальных состояний выполняется:

$$\begin{aligned} Z_R(x) &= \frac{n-x}{n} + \frac{x}{2n} \cdot (1 + Z_R(x)) + \frac{x}{2n} \cdot (1 + Z_R(x-1) + Z_R(x)) = \\ &= \frac{n}{n-x} + Z_R(x-1) \cdot \frac{x}{2(n-x)}. \end{aligned} \tag{3.12}$$

Приведенные выражения представляются как:

$$Z_E(x) = \sum_{i=0}^{x \bmod k} \frac{\binom{n}{x-i}}{\binom{n-1}{x}}, \; Z_R(x) = \sum_{i=0}^{x \bmod k} 2^{-i} \frac{\binom{n}{x-i}}{\binom{n-1}{x}},$$

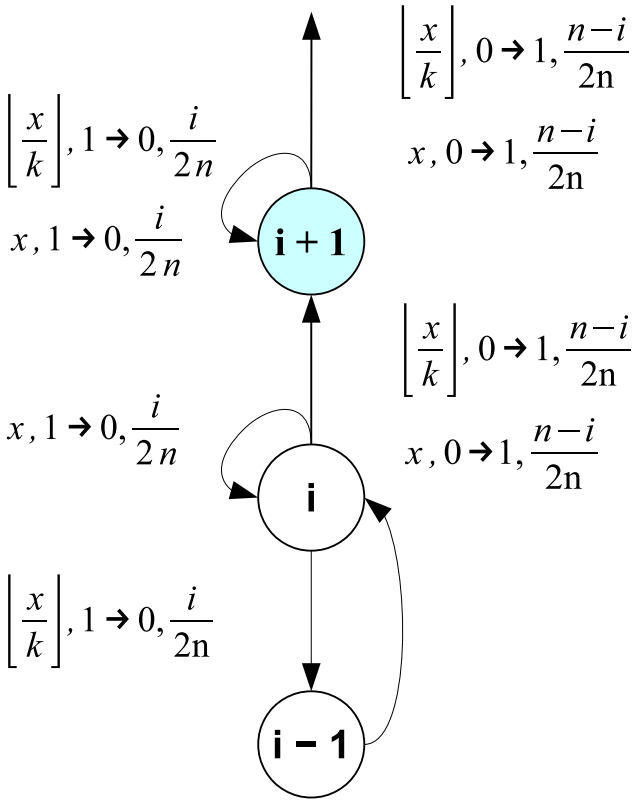

Рисунок 11 – Вероятности переходов при решении задачи XdivK со вспомогательным критерием OneMax

что можно доказать по индукции. Доказательство приводится в разделе 3.3.2.

Для вычисления общего числа итераций, необходимого для достижения максимального значения целевого критерия, следует просуммировать полученные выражения вдоль рассмотренной цепи Маркова: $T_E(x) = \sum_{x=0}^{n-1} Z_E(x)$ и $T_R(x) = \sum_{x=0}^{n-1} Z_R(x)$. В разделе 3.3.3 эти суммы модифицируются для проведения дальнейшего анализа.

Будет показано, что $T_E(n,k) = T_R(n,k) = \Omega(n^k) = O(n^{k+1})$, где k — константа (раздел 3.3.4). В разделе 3.3.5 показывается, что если $n \to \infty$ при фиксированном k, то $\frac{T_E(n,k)}{T_R(n,k)} \geqslant 2^{k-2}(1 - o(1))$. Последнее выражение означает, что метод EA + RL позволяет решить модельную задачу XdivK не менее, чем в 2^{k-2} раза быстрее, чем метод спуска со случайными мутациями. Был проведен численный эксперимент, который показал, что данную оценку можно попытаться улучшить до 2^{k-1}.

3.3.2 Устранение рекурсии

Устраним рекурсию в выражениях для Z_E и Z_R.

Теорема 4. *Следующее равенство выполняется для всех x:*

$$Z_E(x) = \sum_{i=0}^{x \bmod k} \frac{\binom{n}{x-i}}{\binom{n-1}{x}}. \tag{3.13}$$

Доказательство. Докажем по индукции. Для всех x, таких что $x \bmod k = 0$, база индукции следует из выражения (3.9):

$$Z_E(x) = \frac{n}{n-x} = \frac{\binom{n}{x}}{\binom{n-1}{x}} = \sum_{i=0}^{x \bmod k} \frac{\binom{n}{x-i}}{\binom{n-1}{x}}.$$

Предположим, что $v > 0$ и теорема доказана для всех x', таких что $x' \bmod k = v - 1$. Тогда для всех x, удовлетворяющих равенству $x \bmod k = v$, выполняем шаг индукции, используя выражение 3.10:

$$\begin{aligned}
Z_E(x) &= \frac{n}{n-x} + Z_E(x-1) \cdot \frac{x}{n-x} = \\
&= \frac{n}{n-x} + \frac{x}{n-x} \left(\sum_{i=0}^{(x-1) \bmod k} \frac{\binom{n}{x-1-i}}{\binom{n-1}{x-1}} \right) = \\
&= \frac{n}{n-x} + \frac{x}{n-x} \left(\sum_{i=1}^{x \bmod k} \frac{\binom{n}{x-i}}{\binom{n-1}{x-1}} \right) = \\
&= \frac{n}{n-x} + \sum_{i=1}^{x \bmod k} \frac{\binom{n}{x-i}}{\binom{n-1}{x}} = \\
&= \frac{\binom{n}{x}}{\binom{n-1}{x}} + \sum_{i=1}^{x \bmod k} \frac{\binom{n}{x-i}}{\binom{n-1}{x}} = \sum_{i=0}^{x \bmod k} \frac{\binom{n}{x-i}}{\binom{n-1}{x}}. \quad \square
\end{aligned}$$

Теорема 5. *Следующее равенство выполняется для всех x:*

$$Z_R(x) = \sum_{i=0}^{x \bmod k} 2^{-i} \frac{\binom{n}{x-i}}{\binom{n-1}{x}}. \tag{3.14}$$

Доказательство. Теорема доказывается по индукции аналогично Теореме 4 с использованием выражений (3.11) и (3.12). $\square$

3.3.3 Модификация выражений

Пусть используется метод спуска со случайными мутациями, и начальная особь состоит из всех нулей. Тогда суммарное число вычислений функции приспособленности, необходимое для нахождения оптимума целевой функции в задаче XdivK, составляет:

$$T_E(n,k) = \sum_{x=0}^{n-1} Z_E(x) = \sum_{x=0}^{n-1} \sum_{i=0}^{x \bmod k} \frac{\binom{n}{x-i}}{\binom{n-1}{x}}. \tag{3.15}$$

Аналогично, пусть применяется метод EA+RL и начальная особь состоит из всех нулей. Тогда суммарное число вычислений функции приспособленности, необходимое для нахождения оптимума целевой функции в задаче XdivK, составляет:

$$T_R(n,k) = \sum_{x=0}^{n-1} Z_R(x) = \sum_{x=0}^{n-1} \sum_{i=0}^{x \bmod k} 2^{-i} \frac{\binom{n}{x-i}}{\binom{n-1}{x}}. \tag{3.16}$$

Эти выражения достаточно сложно анализировать, особенно если требуется сравнить асимптотическое поведение T_E и T_R. Перепишем $T_E(n,k)$ в следующем виде:

$$\begin{aligned} T_E(n,k) &= \sum_{x=0}^{n-1} \sum_{i=0}^{x \bmod k} \frac{\binom{n}{x-i}}{\binom{n-1}{x}} = \\ &= \sum_{m=0}^{\frac{n}{k}-1} \sum_{j=0}^{k-1} \sum_{i=0}^{j} \frac{\binom{n}{mk+j-i}}{\binom{n-1}{mk+j}} = \\ &= \sum_{m=0}^{\frac{n}{k}-1} \sum_{i=0}^{k-1} \sum_{j=i}^{k-1} \frac{\binom{n}{mk+j-i}}{\binom{n-1}{mk+j}} = \\ &= \sum_{i=0}^{k-1} \sum_{j=i}^{k-1} \sum_{m=0}^{\frac{n}{k}-1} \frac{\binom{n}{mk+j-i}}{\binom{n-1}{mk+j}}. \end{aligned} \tag{3.17}$$

Аналогично, переписав $T_R(n,k)$, получаем:

$$T_R(n,k) = \sum_{x=0}^{n-1} \sum_{i=0}^{x \bmod k} 2^{-i} \frac{\binom{n}{x-i}}{\binom{n-1}{x}} = \\ = \sum_{i=0}^{k-1} 2^{-i} \sum_{j=i}^{k-1} \sum_{m=0}^{\frac{n}{k}-1} \frac{\binom{n}{mk+j-i}}{\binom{n-1}{mk+j}}. \tag{3.18}$$

В результате выполнения приведенных выше преобразований удалось вынести 2^{-i}, насколько это было возможно, наружу, что позволяет упростить оценку асимптотического поведения T_E и T_R.

3.3.4 Асимптотическая оценка числа вычислений функции приспособленности

В этом разделе приводится оценка асимптотического поведения T_E и T_R.

Определим $V(n,k,i)$ следующим образом:

$$V(n,k,i) = \sum_{j=i}^{k-1} \sum_{m=0}^{\frac{n}{k}-1} \frac{\binom{n}{mk+j-i}}{\binom{n-1}{mk+j}}. \tag{3.19}$$

Используя выражения (3.17) и (3.19), получаем:

$$T_E(n,k) = \sum_{i=0}^{k-1} V(n,k,i). \tag{3.20}$$

Аналогично, используя выражения (3.18) и (3.19), получаем:

$$T_R(n,k) = \sum_{i=0}^{k-1} 2^{-i} V(n,k,i). \tag{3.21}$$

Лемма 2. *Для всех i, x, таких что $0 \leqslant i < k$, $0 \leqslant x < n-i-1$, выполняется следующее:*

$$\frac{\binom{n}{x}}{\binom{n-1}{x+i}} < \frac{\binom{n}{x+1}}{\binom{n-1}{x+i+1}}. \tag{3.22}$$

Доказательство.

$$\frac{\frac{\binom{n}{x}}{\binom{n-1}{x+i}}}{\frac{\binom{n}{x+1}}{\binom{n-1}{x+i+1}}} = \frac{(x+1)(n-x-i-1)}{(n-x)(x+i+1)} =$$

$$= \left(1 - \frac{i}{x+i+1}\right)\left(1 - \frac{i+1}{n-x}\right) < 1. \quad \square$$

Из выражения (3.22) следует, что

$$\frac{\binom{n}{x}}{\binom{n-1}{x+i}} \leqslant \binom{n}{n-i-1}. \tag{3.23}$$

Теорема 6. *Если k является константой в зависимости от n, то*

$$V(n,k,i) = \Omega(n^{i+1}) = O(n^{i+2}). \tag{3.24}$$

Доказательство. Рассмотрим в выражении (3.19) случай, когда $j = k-1$ и $m = n/k-1$, тогда отношение биномиальных коэффициентов составляет $\binom{n}{n-i-1}$, так что:

$$\binom{n}{n-i-1} \leqslant V(n,k,i).$$

Из (3.23) следует, что

$$V(n,k,i) \leqslant \binom{n}{n-i-1}(k-i)\frac{n}{k}.$$

Если k зависит от n как константа, тогда из $i < k$ следует, что $\binom{n}{n-i-1} = \Theta(n^{i+1})$. Из этого факта и предыдущего выражения следует доказываемая асимптотическая оценка. $\square$

Теорема 7. *Асимптотическая оценка сложности $T_E(n,k)$ и $T_R(n,k)$ для фиксированного k составляет $\Omega(n^k)$ и $O(n^{k+1})$ соответственно.*

Доказательство. Следует из (3.20), (3.21) и (3.24). $\square$

3.3.5 Отношение числа вычислений функции приспособленности

Получим отношение числа вычислений функций приспособленности, необходимого для достижения оптимума целевого критерия в задаче XdivK при использовании метода спуска со случайными мутациями и метода EA + RL.

Теорема 8. *Для достаточно больших n и фиксированного k*

$$T_E(n,k) \geqslant 2^{k-2} \cdot (1 - o(1)) \cdot T_R(n,k).$$

Доказательство.

$$T_E(n,k) = V(n,k,k-1) + V(n,k,k-2) + o(n^k);$$
$$T_R(n,k) = \frac{V(n,k,k-1)}{2^{k-1}} + \frac{V(n,k,k-2)}{2^{k-2}} + o(n^k).$$

Рассмотрим три случая:

1. $V(n,k,k-1) = \Theta(n^k)$, $V(n,k,k-2) = \Theta(n^k)$. В этом случае оценим отношение следующим образом:

$$\frac{T_E(n,k)}{T_R(n,k)} = \frac{V(n,k,k-1) + V(n,k,k-2) + o(n^k)}{\frac{V(n,k,k-1)}{2^{k-1}} + \frac{V(n,k,k-2)}{2^{k-2}} + o(n^k)} \geqslant$$
$$\geqslant \frac{V(n,k,k-1) + V(n,k,k-2) + o(n^k)}{\frac{V(n,k,k-1)+V(n,k,k-2)}{2^{k-2}} + o(n^k)} =$$
$$= \frac{2^{k-2} + \frac{o(n^k)}{V(n,k,k-1)+V(n,k,k-2)}}{1 + \frac{o(n^k)}{V(n,k,k-1)+V(n,k,k-2)}} =$$
$$= \frac{2^{k-2} + o(1)}{1 + o(1)} \geqslant \frac{2^{k-2}}{1 + o(1)} = 2^{k-2}(1 - o(1)).$$

2. $V(n,k,k-1) = \Theta(n^k)$, $V(n,k,k-2) = o(n^k)$. В этом случае оценим отношение как

$$\frac{T_E(n,k)}{T_R(n,k)} = \frac{V(n,k,k-1) + o(n^k)}{\frac{V(n,k,k-1)}{2^{k-1}} + o(n^k)} \geqslant$$
$$\geqslant \frac{2^{k-1} + \frac{o(n^k)}{V(n,k,k-1)}}{1 + \frac{o(n^k)}{V(n,k,k-1)}} =$$
$$= \frac{2^{k-1} + o(1)}{1 + o(1)} \geqslant \frac{2^{k-1}}{1 + o(1)} = 2^{k-1}(1 - o(1)).$$

3. $V(n,k,k-1) = \omega(n^k)$. В этом случае оценим отношение следующим образом:

$$\frac{T_E(n,k)}{T_R(n,k)} = \frac{V(n,k,k-1) + O(n^k)}{\frac{V(n,k,k-1)}{2^{k-1}} + O(n^k)} \geqslant$$
$$\geqslant \frac{2^{k-1} + \frac{O(n^k)}{V(n,k,k-1)}}{1 + \frac{O(n^k)}{V(n,k,k-1)}} =$$
$$= \frac{2^{k-1} + o(1)}{1 + o(1)} \geqslant \frac{2^{k-1}}{1 + o(1)} = 2^{k-1}(1 - o(1)).$$

Таким образом, в первом случае получаем оценку отношения $2^{k-2}(1 - o(1))$, в двух оставшихся случаях получаем $2^{k-1}(1 - o(1))$. □

3.3.6 Оценка асимптотики времени оптимизации XdivK

Значения $T_E(n, k)$ и $T_R(n, k)$ для различных n и k представлены в таблице 1. Можно видеть, что значение отношения T_E/T_R медленно растет с ростом n при постоянном значении k. Отношение ведет себя очень похоже на 2^{k-1}.

Для осуществления оценки асимптотики времени оптимизации XdivK методом спуска со случайными мутациями без вспомогательных критериев была составлена таблица значений $T_E(n, k)$ и $T_E(n/2, k)$ для различных n и k. Результаты представлены в таблице 2. Можно видеть, что значение отношения $T_E(n, k)/T_E(n/2, k)$ времени оптимизации XdivK для n и $n/2$ достигает 2^k с ростом n, что позволяет выдвинуть гипотезу о значении точной оценки на время оптимизации XdivK: $T_E(n, k) = \Theta(n^k)$.

Из представленных наблюдений следует, что второй случай в теореме 8 ($V(n, k, k-1) = \Theta(n^k)$, $V(n, k, k-2) = o(n^k)$) отражает поведение анализируемых алгоритмов на практике.

3.3.7 Выводы по результатам анализа задачи XdivK

В результате проведенного теоретического анализа были получены точные выражения для следующих функций:

- $T_E(n, k)$ — математическое ожидание времени работы метода спуска со случайными мутациями при оптимизации XdivK;
- $T_R(n, k)$ — математическое ожидание времени работы алгоритма *RLS* + *Q-learning* при оптимизации XdivK со вспомогательным критерием OneMax, состоянием, вычисляемым как значение XdivK, и наградой, равной разности значений XdivK на двух последовательных итерациях.

С использованием полученных выражений были проведены теоретические доказательства следующих фактов:

- $T_E(n, k) = \Omega(n^k) = O(n^{k+1})$ для постоянного k;

Таблица 1 – Экспериментальное определение отношения T_E/T_R

n	k	$T_E(n,k)$	$T_R(n,k)$	$T_E(n,k)/T_R(n,k)$
40	2	$1.19 \cdot 10^3$	$6.81 \cdot 10^2$	1.75
48	2	$1.70 \cdot 10^3$	$9.56 \cdot 10^2$	1.78
56	2	$2.30 \cdot 10^3$	$1.28 \cdot 10^3$	1.80
64	2	$2.98 \cdot 10^3$	$1.64 \cdot 10^3$	1.81
72	2	$3.76 \cdot 10^3$	$2.05 \cdot 10^3$	1.83
80	2	$4.62 \cdot 10^3$	$2.51 \cdot 10^3$	1.84
60	3	$3.94 \cdot 10^4$	$1.08 \cdot 10^4$	3.66
72	3	$6.79 \cdot 10^4$	$1.83 \cdot 10^4$	3.71
84	3	$1.08 \cdot 10^5$	$2.86 \cdot 10^4$	3.76
96	3	$1.60 \cdot 10^5$	$4.23 \cdot 10^4$	3.79
108	3	$2.28 \cdot 10^5$	$5.97 \cdot 10^4$	3.81
120	3	$3.12 \cdot 10^5$	$8.14 \cdot 10^4$	3.83
80	4	$1.72 \cdot 10^6$	$2.30 \cdot 10^5$	7.45
96	4	$3.57 \cdot 10^6$	$4.72 \cdot 10^5$	7.55
112	4	$6.61 \cdot 10^6$	$8.68 \cdot 10^5$	7.62
128	4	$1.13 \cdot 10^7$	$1.47 \cdot 10^6$	7.66
144	4	$1.81 \cdot 10^7$	$2.35 \cdot 10^6$	7.70
160	4	$2.76 \cdot 10^7$	$3.57 \cdot 10^6$	7.73
100	5	$8.05 \cdot 10^7$	$5.37 \cdot 10^6$	14.98
120	5	$2.02 \cdot 10^8$	$1.33 \cdot 10^7$	15.16
140	5	$4.38 \cdot 10^8$	$2.86 \cdot 10^7$	15.28
160	5	$8.56 \cdot 10^8$	$5.57 \cdot 10^7$	15.38
180	5	$1.55 \cdot 10^9$	$1.00 \cdot 10^8$	15.45
200	5	$2.63 \cdot 10^9$	$1.69 \cdot 10^8$	15.50

Таблица 2 – Экспериментальное определение асимптотического поведения T_E

n	k	$T_E(n,k)$	$T_E(n/2,k)$	$T_E(n,k)/T_E(n/2,k)$
40	2	$1.19 \cdot 10^3$	$3.11 \cdot 10^2$	3.82
48	2	$1.70 \cdot 10^3$	$4.42 \cdot 10^2$	3.84
56	2	$2.30 \cdot 10^3$	$5.95 \cdot 10^2$	3.86
64	2	$2.98 \cdot 10^3$	$7.71 \cdot 10^2$	3.87
72	2	$3.76 \cdot 10^3$	$9.69 \cdot 10^2$	3.88
80	2	$4.62 \cdot 10^3$	$1.19 \cdot 10^3$	3.89
60	3	$3.94 \cdot 10^4$	$5.07 \cdot 10^3$	7.78
72	3	$6.79 \cdot 10^4$	$8.67 \cdot 10^3$	7.83
84	3	$1.08 \cdot 10^5$	$1.37 \cdot 10^4$	7.86
96	3	$1.60 \cdot 10^5$	$2.03 \cdot 10^4$	7.88
108	3	$2.28 \cdot 10^5$	$2.88 \cdot 10^4$	7.90
120	3	$3.12 \cdot 10^5$	$3.94 \cdot 10^4$	7.91
80	4	$1.72 \cdot 10^6$	$1.07 \cdot 10^5$	16.10
96	4	$3.57 \cdot 10^6$	$2.21 \cdot 10^5$	16.10
112	4	$6.61 \cdot 10^6$	$4.11 \cdot 10^5$	16.10
128	4	$1.13 \cdot 10^7$	$7.02 \cdot 10^5$	16.09
144	4	$1.81 \cdot 10^7$	$1.13 \cdot 10^6$	16.09
160	4	$2.76 \cdot 10^7$	$1.72 \cdot 10^6$	16.08
100	5	$8.05 \cdot 10^7$	$2.43 \cdot 10^6$	33.14
120	5	$2.02 \cdot 10^8$	$6.11 \cdot 10^6$	32.98
140	5	$4.38 \cdot 10^8$	$1.33 \cdot 10^7$	32.85
160	5	$8.56 \cdot 10^8$	$2.61 \cdot 10^7$	32.75
180	5	$1.55 \cdot 10^9$	$4.73 \cdot 10^7$	32.68
200	5	$2.63 \cdot 10^9$	$8.05 \cdot 10^7$	32.61

– $T_R(n,k) = \Omega(n^k) = O(n^{k+1})$ для постоянного k;
– отношение $T_X(n,k)/T_R(n,k)$ достигает по крайней мере 2^{k-2} для постоянного k и достаточно большого n.

Также были представлены таблицы значений $T_E(n,k)$ и $T_R(n,k)$ для некоторых значений n и k, которые позволяют предположить, что выполняются следующие точные оценки:

– $T_E(n,k) = \Theta(n^k)$ для постоянного k;
– $T_R(n,k) = \Theta(n^k)$ для постоянного k;
– отношение $T_E(n,k)/T_R(n,k)$ достигает 2^{k-1} для постоянного k и достаточно большого n.

3.4 Задача с эффективным и мешающим критериями

Рассмотрим наиболее общую постановку задачи выбора вспомогательных критериев: учтем возможность наличия как мешающего, так и эффективного критерия. Пусть целевым критерием, как и в предыдущем случае, является XdivK. В качестве вспомогательных критериев рассмотрим эффективный критерий OneMax и мешающий критерий ZeroMax.

В разделе 3.4.1 показано на основе результатов эксперимента, что метод EA + RL решает рассматриваемую задачу недостаточно эффективно, в связи с чем в разделе 3.4.2 предлагается его модификация. В разделе 3.4.3 производится теоретический анализ времени работы этой модификации, в результате которого в разделе 3.4.4 предлагается усовершенствованная модификация и экспериментально подтверждается ее эффективность.

3.4.1 Экспериментальная оценка времени работы EA + RL при наличии эффективного и мешающего критериев

Проанализируем на основе результатов эксперимента, как настройки параметров обучения с подкреплением влияют на эффективность выбора вспомогательных критериев. В таблице 3 показано среднее число вычислений функции приспособленности, которое понадобилось для нахождения оптимума XdivK

методу спуска со случайными мутациями RLS и методу EA + RL при различных значениях параметров обучения. В качестве реализации метода EA + RL использовался алгоритм *RLS + Q-learning*. Каждая конфигурация алгоритма *RLS + Q-learning* запускалась по 10^3 раз, затем результаты усреднялись. В случае, если оптимум не был найден за 10^9 вычислений ФП, число вычислений условно считалось равным бесконечности (в таблице обозначена как ∞). Первая особь генерировалась случайным образом.

Таблица 3 – Число вычислений ФП, понадобившееся для нахождения оптимума XdivK с критериями OneMax и ZeroMax; ε — вероятность исследования среды, ss — единственное состояние, ts — состояние, равное целевому критерию

n, k	RLS	*RLS + Q-learning*		
		ss, $\varepsilon = 0{,}1$	ts, $\varepsilon = 0$	ts, $\varepsilon = 0{,}1$
60, 3	$3{,}94 \cdot 10^4$	$9{,}01 \cdot 10^4$	$1{,}18 \cdot 10^4$	$1{,}45 \cdot 10^6$
72, 3	$6{,}79 \cdot 10^4$	$3{,}84 \cdot 10^5$	$1{,}99 \cdot 10^4$	$1{,}50 \cdot 10^7$
84, 3	$1{,}08 \cdot 10^5$	$1{,}52 \cdot 10^6$	$3{,}15 \cdot 10^4$	∞
96, 3	$1{,}60 \cdot 10^5$	$6{,}86 \cdot 10^6$	$4{,}72 \cdot 10^4$	∞
108, 3	$2{,}28 \cdot 10^5$	$2{,}89 \cdot 10^7$	$6{,}75 \cdot 10^4$	∞
120, 3	$3{,}12 \cdot 10^5$	$1{,}16 \cdot 10^8$	$9{,}36 \cdot 10^4$	∞

Рассматривались следующие определения состояния ЭА: одно состояние (ss) и состояние, равное значению XdivK (ts), а также две вероятности исследования среды: $\varepsilon = 0$ (жадное обучение) и $\varepsilon = 0{,}1$. Одновременное использование одного состояния и жадного обучения не рассматривалось. В этом случае после первого получения положительной награды алгоритм жадно обучается одному критерию и не имеет возможности впоследствии выбрать другой. Впервые положительная награда может быть получена при генерации особи, содержащей kd единичных бит (d целое) и использовании XdivK или OneMax. Таким обра-

зом, EA + RL обучится выбирать XdivK с вероятностью $0{,}5$, а значит, заведомо не удастся получить существенного ускорения по сравнению с RLS.

Из таблицы 3 можно видеть, что метод EA + RL не всегда эффективно справляется с поставленной задачей: случае использования одного состояния и $\varepsilon = 0{,}1$ результаты хуже, чем у RLS. В случае использования состояния, равного значению XdivK, и $\varepsilon = 0{,}1$, метод EA + RL для большинства рассмотренных nи k не успевает найти оптимальное значение в рамках выделенного вычислительного бюджета. И только при использовании $\varepsilon = 0$ можно наблюдать ускорение по отношению к RLS. Это объясняется тем, что при ненулевой вероятности исследования среды существует возможность выбора критерия OneMax, который может повлечь за собой «потерю» особи с большим числом единиц и выбор особи с меньшим числом единиц. В следующем разделе предлагается модификация метода EA + RL, сохраняющая особь с большим числом единиц.

3.4.2 Модификация EA + RL, сохраняющая лучшую особь

Предлагается следующая модификация метода EA + RL, сохраняющая лучшую особь. В методе EA + RL в случае, когда новая особь лучше текущей по выбранному критерию, в качестве особи для следующего поколения выбирается новая особь. Однако если выбран мешающий критерий, то новая особь может быть хуже текущей по целевому критерию. В таком случае ЭА теряет особь с лучшим значением целевого критерия, а для того чтобы снова найти хорошее решение, требуется много итераций ЭА.

В предложенной модификации метода EA + RL в случае, когда новая особь лучше текущей по выбранному критерию, но хуже по целевому критерию, в следующее поколение переходит текущая особь. Как и при исследовании метода EA + RL, в качестве ЭА используется метод спуска со случайными мутациями, а алгоритмом обучения с подкреплением является Q-обучение.

Псевдокод предложенной модификации RLS + Q-learning представлен в листинге 5. Он отличается от псевдокода немодифицированного *RLS + Q-*

learning (листинг 3) в строке 8: при выборе особи для следующего поколения учитывается не только значение выбранного критерия, вычисленное на этой особи, но и значение целевого критерия. Если значение целевого критерия на новой особи меньше, в качестве текущей остается старая особь.

Листинг 5 – Модифицированный алгоритм *RLS* + *Q-learning*, сохраняющий особь с лучшим значением целевого критерия

Require: t — целевой критерий; H — множество вспомогательных критериев; S — множество состояний; α — скорость обучения; γ — дисконтный фактор.

1: Инициализировать $Q(s, f)$ нулями для всех $s \in S$, $f \in H \cup \{t\}$
2: Инициализировать текущую особь: x
3: **while** ($t(x) \neq$ max) **do**
4: Запомнить текущую особь: prev $= x$
5: Вычислить состояние среды $s = t(x)$
6: Выбрать критерий: $f = \arg\max_f Q(s, f)$
7: Применить инверсию случайного бита: $y =$ flip-one-bit(x)
8: **if** $f(y) \geqslant f(x)$ и $t(y) \geqslant t(x)$ **then**
9: $x \leftarrow y$
10: Вычислить награду: $r = t(x) - t($prev$)$
11: Вычислить новое состояние: $s' = t(x)$
12: $Q(s,f) = Q(s,f) + \alpha(r + \gamma \max_{f'} Q(s',f') - Q(s, f))$

Оценим эффективность предложенной модификации. В таблице 4 показано среднее число вычислений функции приспособленности, которое понадобилось для нахождения оптимума XdivK методу спуска со случайными мутациями RLS и модифицированному методу EA + RL при различных значениях параметров обучения. Использованы те же обозначения, что и в таблице 4, эксперимент проводился с такими же параметрами.

Таблица 4 – Число вычислений ФП, понадобившееся для нахождения оптимума XdivK с критериями OneMax и ZeroMax; ε — вероятность исследования среды, ss — единственное состояние, ts — состояние, равное целевому критерию

n, k	RLS	Модиф. *RLS* + *Q-learning*		
		ss, $\varepsilon = 0{,}1$	ts, $\varepsilon = 0$	ts, $\varepsilon = 0{,}1$
60, 3	$3{,}94 \cdot 10^4$	$7{,}18 \cdot 10^4$	$5{,}92 \cdot 10^4$	$5{,}75 \cdot 10^4$
72, 3	$6{,}79 \cdot 10^4$	$1{,}17 \cdot 10^5$	$1{,}01 \cdot 10^5$	$1{,}05 \cdot 10^5$
84, 3	$1{,}08 \cdot 10^5$	$1{,}77 \cdot 10^5$	$1{,}64 \cdot 10^5$	$1{,}52 \cdot 10^5$
96, 3	$1{,}60 \cdot 10^5$	$2{,}61 \cdot 10^5$	$2{,}32 \cdot 10^5$	$2{,}49 \cdot 10^5$
108, 3	$2{,}28 \cdot 10^5$	$3{,}76 \cdot 10^5$	$3{,}43 \cdot 10^5$	$3{,}53 \cdot 10^5$
120, 3	$3{,}12 \cdot 10^5$	$4{,}64 \cdot 10^5$	$4{,}86 \cdot 10^5$	$4{,}84 \cdot 10^5$

Сравнивая результаты в таблицах 3 и 4, можно видеть, что модифицированный метод EA + RL, в отличие от исходной версии EA + RL, позволяет найти оптимум XdivK при использовании всех рассмотренных конфигураций алгоритма и для всех рассмотренных длин особей. Однако среднее число вычислений ФП, необходимое для вычисления оптимума, у модифицированного метода больше, чем у эффективной конфигурации EA + RL с использованием состояния, определяемого значением целевого критерия, и нулевой вероятности исследования среды. В следующем разделе приводится теоретический анализ модифицированной версии EA + RL с этими параметрами и выявляются причины повышения числа вычислений ФП по сравнению с немодифицированной версией.

3.4.3 Анализ времени работы модификации EA + RL при наличии эффективного и мешающего критериев

Для анализа времени работы предложенной модификации метода EA + RL была построена марковская цепь, описывающая соответствующий процесс оптимизации. Данная цепь представлена на рисунке 12.

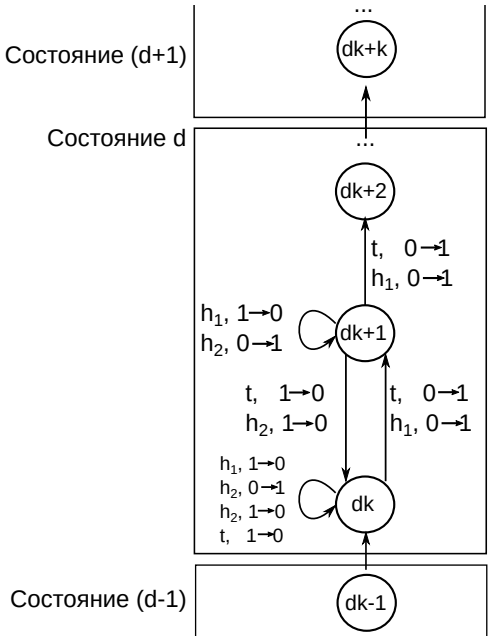

Рисунок 12 – Марковская цепь процесса оптимизации XdivK, проводимого модификацией EA + RL без обучения на ошибках
($t = \text{XdivK}, h_1 = \text{OneMax}, h_2 = \text{ZeroMax}$)

Поясним значения меток на переходах в марковской цепи на рисунке 12. В случае, когда число единиц равно dk, где d — константа, агент находится в состоянии обучения, равном d, и марковское состояние равно dk. У агента нет опыта в состоянии d, поэтому критерии выбираются равновероятно. В случае, когда выбран целевой критерий или критерий, который на данном этапе оптимизации совпадает с OneMax, и оператор мутации инвертировал единичный бит, новая особь содержит $dk-1$ единичный бит. Таким образом, новая особь хуже текущей по выбранному критерию, и она не будет выбрана ЭА для следующего поколения. Та же ситуация возникает, когда выбранный критерий равен ZeroMax и инвертирован нулевой бит. В случае выбора критерия, равного ZeroMax, и инверсии единичного бита, новая особь лучше текущей по выбранному критерию. Однако значение целевого критерия, вычисленное на новой особи, меньше, чем у текущей особи. Таким образом, в качестве особи для следующего поколения выбирается текущая особь. В случае выбора целевого кри-

терия или критерия, равного OneMax, и инверсии нулевого бита новая особь переходит в следующее поколение.

Переходы в марковских состояниях dk и $dk + 1$ отличаются в случае выбора целевого критерия или критерия, равного ZeroMax, и инверсии единичного бита. В этом случае значение выбранного критерия новой особи лучше или совпадает со значением выбранного критерия текущей особи. Поэтому новая особь переходит в следующее поколение. Однако новая особь содержит меньшее число единичных бит, чем текущая, поэтому алгоритм переходит в марковское состояние dk. Переходы в состояниях $dk + 2, \ldots, dk + k - 1$ определяются аналогично переходам в состоянии $dk + 1$.

Для анализа времени работы предложенной модификации также необходимо построить марковскую цепь для метода спуска со случайными мутациями без вспомогательных критериев. Данная марковская цепь представлена на рисунке 13.

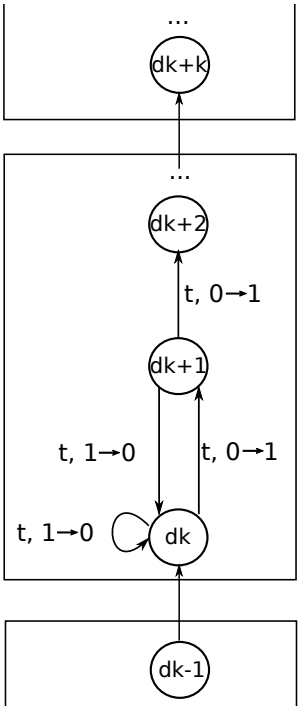

Рисунок 13 – Марковская цепь для RLS без вспомогательных критериев на задаче XdivK

Математическое ожидание времени работы предложенной модификации равно числу вычислений ФП, необходимому для достижения марковского состояния n из состояния 0. Каждый переход в марковской цепи соответствует вычислению ФП новой особи, полученной в результате мутации. Поэтому математическое ожидание времени работы алгоритма равно числу сделанных алгоритмом переходов в марковской цепи. Обозначим математическое ожидание времени работы предложенной модификации как $T(n)$:

$$T(n) = \sum_{i=0}^{n-1} E(i \to i+1), \tag{3.25}$$

где $E(i \to i+1)$ — математическое ожидание числа переходов, необходимое для достижения состояния $i+1$ из состояния i.

Значение $E(i \to i+1)$ зависит от номера состояния i. Рассмотрим два случая с разными значениями i. В первом случае $i = dk$, где d — константа. Математическое ожидание числа переходов, необходимое для достижения состояния $dk+1$ из состояния dk, вычисляется как $z_{dk} = E(dk \to dk+1)$:

$$z_{dk} = \frac{2}{3} \cdot \frac{(n-dk)}{n} \cdot 1 + \left(\frac{2}{3} \cdot \frac{dk}{n} + \frac{1}{3}\right) \cdot (1 + z_{dk}) \tag{3.26}$$

Из (3.26) получаем, что z_{dk} вычисляется как:

$$z_{dk} = \frac{3n}{2(n-dk)} \tag{3.27}$$

Во втором случае $i = dk + t$, где $1 \leqslant t \leqslant k-1$. Математическое ожидание числа переходов, необходимое для достижения состояния $dk+t+1$ из состояния $dk+t$ вычисляется как $z_{dk+t} = E(dk+t \to dk+t+1)$:

$$\begin{aligned} z_{dk+t} = \frac{2(n-dk-t)}{3n} + \frac{2(dk+t)}{3n} \cdot (1 + z_{dk+t-1} + z_{dk+t}) + \\ + \left(\frac{dk+t}{3n} + \frac{n-dk-t}{3n}\right) \cdot (1 + z_{dk+t}) \end{aligned} \tag{3.28}$$

Из (3.28) получаем, что z_{dk+t} вычисляется как:

$$z_{dk+t} = z_{dk+t-1} \cdot \frac{dk+t}{n-dk-t} + \frac{3n}{2(n-dk-t)} \tag{3.29}$$

Для оценки времени работы предложенной модификации необходимо вычислить математическое ожидание времени работы метода спуска со случайными мутациями без вспомогательных критериев. Методика вычисления аналогична использованной выше для предложенной модификации. Общее время выполнения также вычисляется по формуле (3.25). Имеются два случая: $i = dk$ и $i = dk + t$. Математическое ожидание числа переходов, необходимого для достижения состояния $dk + 1$ из состояния dk, вычисляется как $a_{dk} = E(dk \to dk + 1)$:

$$a_{dk} = \frac{(n - dk)}{n} \cdot 1 + \frac{dk}{n} \cdot (1 + a_{dk}) \tag{3.30}$$

Из (3.30) получаем, что a_{dk} вычисляется как:

$$a_{dk} = \frac{n}{(n - dk)} \tag{3.31}$$

Математическое ожидание числа переходов, необходимое для достижения состояния $dk + t + 1$ из состояния $dk + t$ вычисляется как $a_{dk+t} = E(dk + t \to dk + t + 1)$:

$$a_{dk+t} = \frac{(n - dk - t)}{n} \cdot 1 + \frac{dk + t}{n} \cdot (1 + a_{dk+t-1} + a_{dk+t}) \tag{3.32}$$

Из (3.32) получаем, что a_{dk+t} вычисляется как:

$$a_{dk+t} = a_{dk+t-1} \cdot \frac{dk + t}{n - dk - t} + \frac{n}{(n - dk - t)} \tag{3.33}$$

Из (3.27) и (3.31) получаем, что $z_{dk} = \frac{3}{2} a_{dk}$. Из выражений (3.25), (3.29), (3.33) при помощи математической индукции получаем, что время работы предложенной модификации метода EA + RL на задаче XdivK с переключающимися критериями в 1,5 раза больше времени работы метода спуска со случайными мутациями. Таким образом, верхняя и нижняя асимптотические оценки времени работы предложенной модификации на задаче XdivK совпадают с оценками времени работы RLS, которые равны $O(n^{k+1})$ и $\Omega(n^k)$ [32]. Данный результат показывает, что предложенная модификация

метода EA + RL успешно игнорирует мешающий вспомогательный критерий в асимптотическом смысле.

Рассмотрим причину того, почему точное время работы модифицированного EA + RL больше времени работы RLS без мешающих критериев в постоянное число раз. Модифицированный EA + RL гарантированно не выбирает особь с худшим значением целевого критерия XdivK. Однако при работе этого метода существует вероятность отвергнуть лучшую особь в случае, когда выбран мешающий вспомогательный критерий ZeroMax, и оставить старый критерий, из-за чего процесс оптимизации замедляется. Для того, чтобы этого не происходило, следует в соответствующем случае присваивать отрицательную награду, несмотря на то, что особь фактически не ухудшается. Соответствующее улучшение модифицированного EA + RL предлагается и экспериментально анализируется в следующем разделе.

3.4.4 Модификация EA + RL с сохранением лучшей особи и обучением на ошибках

Награду в строке 10 листинга 5 можно вычислять двумя способами: первый способ указан в листинге, вычисляется разность значений целевого критерия между выбранной особью и особью из предыдущего поколения. Однако в случае, когда особь может быть выбрана по значению выбранного критерия, но не выбирается по значению целевого критерия, ее также можно использовать для вычисления награды. Первый способ будем называть вычислением награды без обучения на ошибках, второй — с обучением на ошибках.

В листинге 6 приведен псевдокод улучшенной модификации метода EA + RL, обучающейся на ошибках. Основные содержательные отличия от псевдокода первой версии модификации EA + RL, приведенной в листинге 5, содержатся в строках 7–12. Теперь награда вычисляется всегда, когда новая особь подходит по выбранному критерию f (строка 8), независимо от того, выбирается ли она в качестве новой текущей особи в строке 10 на основании

значения целевого критерия t, или нет. В случае, если особь обладает худшим значением выбранного критерия f, текущая особь не меняется, как и в первой версии модификации EA + RL, и награда оказывается нулевой (строка 12). Также в новом псевдокоде исчезла необходимость в запоминании текущей особи и использовании переменной prev, присутствующей в псевдокоде первой версии.

Листинг 6 – Модифицированный алгоритм *RLS* + *Q-learning*, сохраняющий особь с лучшим значением целевого критерия

Require: t — целевой критерий; H — множество вспомогательных критериев; S — множество состояний; α — скорость обучения; γ — дисконтный фактор.

1: Инициализировать $Q(s, f)$ нулями для всех $s \in S$, $f \in H \cup \{t\}$
2: Инициализировать текущую особь: x
3: **while** ($t(x) \neq$ max) **do**
4: Вычислить состояние среды $s = t(x)$
5: Выбрать критерий: $f = \arg\max_f Q(s, f)$
6: Применить инверсию случайного бита: $y =$ flip-one-bit(x)
7: **if** $f(y) \geqslant f(x)$ **then**
8: Вычислить награду: $r = t(y) - t(x)$
9: **if** $t(y) \geqslant t(x)$ **then**
10: Выбрать новую особь как текущую: $x = y$
11: **else**
12: Обнулить награду (особь не изменилась): $r = 0$
13: Вычислить новое состояние: $s' = t(x)$
14: $Q(s,f) = Q(s,f) + \alpha(r + \gamma \max_{f'} Q(s',f') - Q(s, f))$

В таблице 5 приводятся результаты, полученные с использованием обучения на ошибках. Сравнивая таблицы 3–5 можно видеть, что при использовании нулевой вероятности исследования среды и состояния, равного целевому кри-

терию, предложенной модификации с обучением на ошибках требуется меньше всего вычислений ФП по сравнению со всеми методами, рассмотренными при решении задачи XdivK с эффективным и мешающим вспомогательными критериями. В частности, удается достичь примерно четырехкратного ускорения по сравнению с методом RLS без вспомогательных критериев и примерно шестикратного по сравнению с наиболее эффективной конфигурацией первой версии без обучения на ошибках. Также заметим, что модификация EA + RL с обучением на ошибках во всех случаях находит оптимальное значение XdivK, в отличие от метода EA + RL.

Таблица 5 – Число вычислений ФП, понадобившееся для нахождения оптимума XdivK с критериями OneMax и ZeroMax; ε — вероятность исследования среды, ss — единственное состояние, ts — состояние, равное целевому критерию

n, k	RLS	Модиф. *RLS* + *Q-learning*, обуч. на ош.		
		ss, $\varepsilon = 0{,}1$	ts, $\varepsilon = 0$	ts, $\varepsilon = 0{,}1$
60, 3	$3{,}94 \cdot 10^4$	$1{,}39 \cdot 10^4$	$1{,}06 \cdot 10^4$	$1{,}27 \cdot 10^4$
72, 3	$6{,}79 \cdot 10^4$	$2{,}28 \cdot 10^4$	$1{,}91 \cdot 10^4$	$2{,}17 \cdot 10^4$
84, 3	$1{,}08 \cdot 10^5$	$3{,}61 \cdot 10^4$	$2{,}71 \cdot 10^4$	$3{,}28 \cdot 10^4$
96, 3	$1{,}60 \cdot 10^5$	$5{,}25 \cdot 10^4$	$4{,}30 \cdot 10^4$	$4{,}85 \cdot 10^4$
108, 3	$2{,}28 \cdot 10^5$	$7{,}38 \cdot 10^4$	$5{,}72 \cdot 10^4$	$6{,}67 \cdot 10^4$
120, 3	$3{,}12 \cdot 10^5$	$9{,}92 \cdot 10^4$	$8{,}14 \cdot 10^4$	$9{,}72 \cdot 10^4$

На основании результатов данного раздела можно заключить, что генерацию тестов наиболее перспективно производить с помощью модификации метода EA + RL с обучением на ошибках. В следующей главе модификация метода EA + RL с обучением на ошибках, а также его исходная версия, описанная в главе 2, применяются к практической задаче оценки эффективности программ решения задачи дискретной математики; производится сравнение этих версий,

а также сравнение с другими существующими методами выбора вспомогательных критериев.

Выводы по главе 3

Определены свойства вспомогательных критериев, используемых при генерации тестов, и получены асимптотические оценки времени работы алгоритма, реализующего предложенный метод, для мешающих и эффективных вспомогательных критериев, обладающих этими свойствами. На основе выполненного анализа предложена модификация разработанного алгоритма. Проведенный анализ основан на представлении процесса оптимизации в виде марковской цепи. Подход к анализу предложенного метода и полученные результаты опубликованы в трудах ряда международных конференций, индексируемых в системе *Scopus* [29, 32, 33, 42, 94, 97].

ГЛАВА 4 ПРИМЕНЕНИЕ МЕТОДА EA + RL ДЛЯ ОЦЕНКИ ЭФФЕКТИВНОСТИ ПРОГРАММ РЕШЕНИЯ ЗАДАЧИ «SHIPS. VERSION 2»

В данной главе проводится экспериментальная оценка эффективности метода выбора вспомогательных критериев EA + RL, предложенного в главе 2. Метод применяется для повышения эффективности генерации тестов, выявляющих неэффективные решения олимпиадной задачи по программированию «Ships. Version 2» [121]. Производится сравнение с другими существующими методами выбора вспомогательных критериев.

4.1 Описание экспериментального исследования

В ходе экспериментального исследования тесты для программ, решающих задачу «Ships. Version 2», генерируются с помощью предложенного метода EA + RL и других существующих методов, позволяющих использовать вспомогательные критерии в ЭА. Параметры и структура ЭА, описанные в подразделах 4.1.2 и 4.1.3 являются общими для всех рассматриваемых подходов, и соответствуют описанию ЭА из работы [2], в которой также решается задача «Ships. Version 2»и предлагается один из методов выбора вспомогательных критериев SaRA, с которым производится сравнение.

4.1.1 Задача, решаемая тестируемыми программами

Задача «Ships. Version 2»описана в архиве задач с проверяющей системой *Timus* Уральского федерального университета под номером 1394 [121].

Условие задачи формулируется следующим образом. Дано N кораблей (англ. *ships*, отсюда название задачи), каждый длиной s_i, и M гаваней, каждая длиной h_j. Все корабли необходимо разместить в гаванях, причем таким образом, чтобы суммарная длина кораблей, размещенных в j-ой гавани, не превосходила h_j.

Гарантируется, что для всех предлагаемых наборов входных данных существует размещение, удовлетворяющее требованиям задачи. На входные данные

и характеристики работы программы, решающей задачу, налагаются следующие ограничения:

- $N \leqslant 99, 2 \leqslant M \leqslant 9, 1 \leqslant s_i \leqslant 100$;
- $\sum s_i = \sum h_j$;
- предельное значение времени работы программы составляет одну секунду;
- предельное значение объема оперативной памяти, которую может использовать программа, составляет 64 мегабайта.

Эта задача является частным случаем задачи о мультирюкзаке, которая, в свою очередь, является NP-трудной в сильном смысле [99]. Это означает, что неизвестно о существовании программы, которая решала бы задачу за полиномиальное время для любых возможных значений входных данных. По этой причине крайне маловероятно, что любой возможный экземпляр задачи «Ships. Version 2»может быть решен в рамках ограничений на память и время, заданных в условии. Однако для самых эффективных программ решения этой задачи достаточно сложно составить тест, на котором время работы программы превышает заданное ограничение.

4.1.2 Представление тестов в эволюционных алгоритмах

Тесты кодируются в виде особей эволюционного алгоритма, представляющих собой списки целых чисел от нуля до 100, аналогично работе [2]. Каждое ненулевое число в этом списке задает длину соответствующего корабля, а сумма каждой последовательности ненулевых чисел задает длину соответствующей гавани, последовательности разделяются нулями. Пример теста, закодированного таким образом, представлен на рисунке 14.

Заметим, что такое представление теста содержит в себе решение задачи. Для применения теста в проверяющей системе необходимо поменять места гаваней и кораблей, чтобы исключить возможность использования этого факта.

Особь: $0, \underbrace{3, 7, 5}_{15}, 0, \underbrace{1, 2}_{3}, 0, \underbrace{4, 7, 6}_{17}, 0, 0, \underbrace{1, 3, 8}_{12}$

Корабли: $3, 7, 5, 1, 2, 4, 7, 6, 1, 3, 8$

Гавани: $15, 3, 17, 12$

Рисунок 14 – Пример теста, представленного в качестве особи ЭА

Например, можно отсортировать корабли и гавани в порядке возрастания их длин.

Использование описанного представления позволяет выполнить требования, накладываемые на входные данные, которые приведены в разделе 4.1.1: сумма длин кораблей автоматически оказывается равной сумме длин гаваней, и для любого такого теста существует решение, удовлетворяющее условию задачи. Однако ограничения на число кораблей и гаваней могут не выполняться. В этом случае соответствующей особи присваивается нулевое значение приспособленности.

4.1.3 Эволюционные операторы

Новая особь формируется из L случайно сгенерированных чисел, помещаемых в представляющий особь список в порядке генерации. Числа генерируются следующим образом: ноль выбирается с вероятностью $1/5$, в противном случае равновероятно выбирается положительное значение из промежутка $[1; 100]$.

Оператор мутации заменяет каждое число в особи с вероятностью $1/L$ на число, случайно сгенерированное описанным выше способом. Используется следующий вариант двухточечного оператора скрещивания. Пусть числа, входящие в особь, проиндексированы в диапазоне от 1 до L. В операторе скрещивания случайным образом выбирается длина обмена X из интервала $[1; L]$. Затем случайным образом выбирается сдвиг F_1 в первой особи из интервала

$[1; L - X + 1]$. Наконец, сдвиг F_2 во второй особи случайно выбирается из того же интервала независимо от F_1. Подстроки с индексами $[F_1, F_1 + X - 1]$ из первой особи и $[F_2, F_2 + X - 1]$ из второй особи обмениваются. Таким образом, формируются две новые особи.

4.1.4 Вспомогательные критерии

В качестве вспомогательных критериев используются значения счетчиков числа вызовов элементов тестируемых программ: процедур и итераций циклов. Счетчики добавляются в тестируемую программу автоматически [116]. Полные коды тестируемых программ со вставленными счетчиками приведены в приложении. Число счетчиков разное для разных программ и зависит от результатов работы утилиты, автоматически вставляющей счетчики в исходный код. В тестируемых в данной работе программах использовалось от 16 до 94 счетчиков. Для демонстрации принципов работы утилиты вставки счетчиков в листинге 7 приведен пример кода программы, сортирующей массив случайных чисел, до и после вставки четырех счетчиков.

Для того, чтобы иметь возможность вызывать тестируемые программы непосредственно из кода, реализующего ЭА, рассматриваются программы, написанные на языке *Java*. Данный подход позволяет не тратить лишнее время на вызов внешней программы для вычисления приспособленности особей ЭА. В код тестируемых программ с помощью специально написанной утилиты вставляются объявления глобальных статических переменных, предназначенных для хранения значений счетчиков (строки 7–10 в примере из листинга 7 (б)), а также статические методы для доступа к этим значениям и их обнуления (строки 18 и 11 соответственно). Увеличение значений счетчиков производится в коде тестируемых программ в следующих местах:

– начало каждого метода (кроме метода `main`, строка 28 листинга 7 (б));
– начало каждого цикла (строки 35, 39, 41 листинга 7 (б)).

Листинг 7 – Пример кода до (а) и после (б) вставки счетчиков

(а)

```
1  import java.util.Random;
2
3  public class Sorting {
4    private static Random random = new Random();
5
6    private static int number() {
7      return random.nextInt(1000);
8    }
9
10   public static void main(String[] args) {
11     int N = 500;
12     int[] arr = new int[N];
13     for (int i = 0; i < N; ++i) {
14       arr[i] = number();
15     }
16     for (int i = 0; i < N; ++i) {
17       for (int j = 1; j < N; ++j) {
18         if (arr[j - 1] > arr[j]) {
19           int tmp = arr[j - 1];
20           arr[j - 1] = arr[j];
21           arr[j] = tmp;
22         }
23       }
24     }
25   }
26 }
```

(б)

```
1  import java.util.Random;
2
3  import java.util.Map;
4  import java.util.HashMap;
5  import static util.TimeoutChecker.check;
6  public class SortingCounted {
7    private static long counter$0 = 0;
8    private static long counter$1 = 0;
9    private static long counter$2 = 0;
10   private static long counter$3 = 0;
11   public static void profilerCleanup() {
12     counter$0 = 0;
13     counter$1 = 0;
14     counter$2 = 0;
15     counter$3 = 0;
16     counter$4 = 0;
17   }
18   public static Map<String, Long> profilerData() {
19     Map<String, Long> rv = new HashMap<>();
20     rv.put("counter$0", counter$0);
21     rv.put("counter$1", counter$1);
22     rv.put("counter$2", counter$2);
23     rv.put("counter$3", counter$3);
24     return rv;
25   }
26   private static Random random = new Random();
27   private static int number() {
28 if ((++counter$0 & 262143) == 0) check();
29     return random.nextInt(1000);
30  }
31   public static void main(String[] args) {
32     int N = 500;
33     int[] arr = new int[N];
34     for (int i = 0; i < N; ++i) {
35 if ((++counter$1 & 262143) == 0) check();
36       arr[i] = number();
37     }
38     for (int i = 0; i < N; ++i) {
39 if ((++counter$2 & 262143) == 0) check();
40       for (int j = 1; j < N; ++j) {
41 if ((++counter$3 & 262143) == 0) check();
42         if (arr[j - 1] > arr[j]) {
43           int tmp = arr[j - 1];
44           arr[j - 1] = arr[j];
45           arr[j] = tmp;
46         }
47       }
48     }
49   }
50 }
```

Для обнаружения этих мест и вставки в них инструкций по увеличению счетчиков использовалось дерево разбора кода, построенное с помощью *ANTLR* [91].

Тестируемые программы могут работать достаточно долго, поэтому необходимо установить ограничение на время их работы и прерывать программу, когда ограничение превышено. В виртуальной машине *Java* не предусмотрена возможность остановки потока исполнения по причине превышения ограничения на время выполнения. Поэтому данная возможность была специально реализована следующим образом. Время, прошедшее с начала запуска текущего потока, проверяется в специальной процедуре `check` в тех же фрагментах кода, где происходит увеличение значений счетчиков (строки 28, 35, 39, 41 листинга 7 (б)). В случае, если ограничение превышено, вызывается исключение, которое обрабатывается в процедуре вычисления функции приспособленности и приводит к завершению выполнения тестируемой программы. Чтобы снизить число системных вызовов, необходимых для выполнения упомянутой проверки, процедура `check` вызывается через каждые $2^{18} = 262144$ увеличений значения счетчика.

Вычисление значений вспомогательных критериев производится на основе значений счетчиков следующим образом. ЭА формирует тесты для конкретной программы решения задачи «Ships. Version 2». В случае, если тест не удовлетворяет ограничениям на число кораблей или гаваней, значения всех критериев считаются нулевыми. В противном случае программа запускается на тесте. В процессе работы программы автоматически вычисляются значения вставленных в нее счетчиков, также определяется время, в течение которого программа работала. Таким образом, одного запуска программы достаточно для вычисления значений всех вспомогательных критериев (значений счетчиков) и целевого критерия (времени работы). Из этого следует, что наличие вспомогательных критериев не увеличивает время вычисления функции приспособленности, в частности, это время не зависит от числа критериев.

4.1.5 Порядок проведения экспериментов

В ходе экспериментов тесты для задачи «Ships. Version 2»генерировались с помощью различных реализаций метода EA + RL и его модификации, а также с помощью других существующих алгоритмов выбора критериев. Параметры реализаций и использованные алгоритмы перечислены в разделах 4.2 и 4.3, описывающих результаты соответствующих экспериментов.

Реализации всех методов генерации тестов, а также всех программ, предназначенных для тестирования, выполнены на языке программирования *Java* и исполнялись под 64-битной виртуальной машиной OpenJDK версии `1.8.0_91`. Все эксперименты выполнялись на сервере с четырьмя шестнадцатиядерными процессорами AMD Opteron™ 6378, работающими на частоте 2,4 ГГц. На сервере установлена операционная система Ubuntu Server 14.04 семейства GNU/Linux. Запуск каждой конфигурации метода генерации тестов на каждой программе производился в отдельном экземпляре виртуальной машины с отведенными для этого двумя логическими процессорами и двумя гигабайтами оперативной памяти.

Для каждого алгоритма выделялся одинаковый вычислительный бюджет. Условием останова для каждого алгоритма являлось нахождение сложного теста или, в случае если сложный тест не был найден, исчерпание вычислительного бюджета. Предельное допустимое значение времени работы программы было увеличено с одной секунды (условие задачи) до пяти секунд для обеспечения нахождения гарантированно сложного теста.

По результатам всех запусков каждого алгоритма вычислялось два показателя: процент запусков, завершившихся нахождением сложного теста, и медианное значение времени работы алгоритма, прошедшее с момента запуска до выполнения условия останова. В случаях, когда более 50 % запусков завершились нахождением сложного теста, это медианное значение отражает время, необходимое для его нахождения.

4.2 Подбор параметров для метода EA + RL и его модификации

В данном разделе описываются результаты предварительного эксперимента, проведенного на ограниченном числе программ, и предназначенного для первоначальной настройки метода EA + RL и его модификации. Вычислительный бюджет для каждой рассматриваемой реализации метода составлял 10^6 вычислений функции приспособленности. Время работы алгоритмов измерялось как число вычислений ФП. Каждый алгоритм запускался на каждой из тестируемых программ 100 раз.

Далее производится выбор алгоритма обучения с подкреплением путем сравнения реализаций EA + RL и его модификации, использующих различные алгоритмы. Для выбранного алгоритма обучения сравнивается эффективность определений состояния, которые были описаны в разделе 2.2.2.

4.2.1 Выбор алгоритма обучения

Рассмотрим следующие алгоритмы обучения с подкреплением (параметры были подобраны в ходе предварительного эксперимента):

- Q-обучение с параметрами $\varepsilon = 0{,}01, \alpha = 0{,}8, \gamma = 0{,}2$ [107];
- отложенное Q-обучение с параметрами $\varepsilon = 0{,}001, m = 5, \gamma = 0{,}1$ [90];
- R-обучение с параметрами $\varepsilon = 0{,}01, \alpha = 0{,}2, \beta = 0{,}3$ [90, 102];
- алгоритмы решения задачи о многоруком бандите *UCB1* и *UCB2* с параметром $\alpha = 0{,}001$ [26, 27].

В таблице 6 для реализаций метода EA + RL, использующих различные алгоритмы обучения с подкреплением, приведены значения процента запусков, в которых были найдены сложные тесты, а также значения ожидаемого числа вычислений ФП, необходимого для нахождения сложного теста. В случае, если реализация ни разу не нашла сложный тест, на месте ожидаемого значения числа вычислений ФП ставится прочерк. В таблице 7 приведены аналогичные значения для модификации метода EA + RL с сохранением лучшей особи и обучением на ошибках.

Таблица 6 – Результаты генерации тестов с помощью метода EA + RL при использовании различных алгоритмов обучения (верхняя часть ячейки — доля успешных запусков, нижняя — оценка числа вычислений ФП)

ID программы	Q-обучение	Отлож. Q-обучение	R-обучение	*UCB1*	*UCB2*
2208365	98% $2{,}54 \cdot 10^5$	0% –	66% $7{,}56 \cdot 10^5$	77% $6{,}02 \cdot 10^5$	70% $7{,}22 \cdot 10^5$
1700736	3% $3{,}31 \cdot 10^7$	0% –	60% $1{,}00 \cdot 10^6$	2% $4{,}94 \cdot 10^7$	10% $9{,}24 \cdot 10^6$
3600147	74% $6{,}65 \cdot 10^5$	56% $1{,}16 \cdot 10^6$	74% $7{,}25 \cdot 10^5$	25% $3{,}23 \cdot 10^6$	24% $3{,}37 \cdot 10^6$
3589819	0,00% –	0,00% –	1,00% $9{,}97 \cdot 10^7$	4,00% $2{,}46 \cdot 10^7$	3,00% $3{,}31 \cdot 10^7$

Программы, для которых генерировались тесты, обозначены с помощью идентификаторов, под которыми они значатся в проверяющей системе *Timus*. Всего было рассмотрено четыре программы. Лучшие достигнутые значения процентов успешных запусков и числа вычислений ФП для каждой программы выделены серым фоном. Чтобы поставить все алгоритмы в одинаковые условия, использовалось единственное состояние, так как в алгоритмах *UCB1* и *UCB2* не предусмотрено использование различных состояний.

На основании сопоставления данных из таблиц можно сделать вывод, что для каждой программы наиболее эффективной оказалась реализация либо метода EA + RL, либо его модификации, основанная на Q-обучении. Алгоритм Q-обучения оказался наиболее эффективным алгоритмом для двух программ в случае использования исходной версии EA + RL и для трех программ в случае использования его модификации. Каждый из остальных рассмотренных алго-

Таблица 7 – Результаты генерации тестов с помощью **модификации** метода EA + RL при использовании различных алгоритмов обучения (верхняя часть ячейки — доля успешных запусков, нижняя — оценка числа вычислений ФП)

ID программы	Q-обучение	Отлож. Q-обучение	R-обучение	*UCB1*	*UCB2*
2208365	94% $2{,}65 \cdot 10^5$	1% $9{,}93 \cdot 10^7$	75% $4{,}98 \cdot 10^5$	82% $4{,}61 \cdot 10^5$	80% $5{,}21 \cdot 10^5$
1700736	94% $4{,}43 \cdot 10^5$	0% –	56% $1{,}09 \cdot 10^6$	2% $4{,}91 \cdot 10^7$	3% $3{,}26 \cdot 10^7$
3600147	26% $3{,}19 \cdot 10^6$	38% $1{,}90 \cdot 10^6$	22% $3{,}76 \cdot 10^6$	18% $4{,}68 \cdot 10^6$	12% $7{,}50 \cdot 10^6$
3589819	6% $1{,}64 \cdot 10^7$	0% –	5% $1{,}96 \cdot 10^7$	1% $9{,}95 \cdot 10^7$	4% $2{,}45 \cdot 10^7$

ритмов оказался эффективным не более одного раза. Таким образом, для дальнейших экспериментов будет использоваться алгоритм Q-обучения.

Также отметим, что модификация EA + RL оказалась в целом эффективнее исходной версии метода, однако для программы с идентификатором 3600147 ситуация обратная. Поэтому продолжим рассматривать как модификацию метода, так и исходную версию.

4.2.2 Выбор определения состояния

Сравним эффективность реализаций метода EA + RL и его модификации, использующих единственное состояние и состояние, равное значению целевого критерия. В таблице 8 приведены результаты соответствующего эксперимента для метода EA + RL и его модификации.

Из представленных результатов следует, что эффективность реализаций метода EA + RL и его модификации существенно не меняется при использо-

Таблица 8 – Результаты генерации тестов с помощью метода EA + RL и его модификации при использовании одного состояния (s) и состояния, равного целевому критерию (t); верхняя часть ячейки — доля успешных запусков, нижняя — оценка числа вычислений ФП

ID программы	EA + RL, s	EA + RL, t	Модиф., s	Модиф., t
2208365	98% $2{,}54 \cdot 10^5$	98% $2{,}78 \cdot 10^5$	94% $2{,}65 \cdot 10^5$	97% $2{,}53 \cdot 10^5$
1700736	3% $3{,}31 \cdot 10^7$	0% –	94% $4{,}43 \cdot 10^5$	90% $4{,}82 \cdot 10^5$
3600147	74% $6{,}65 \cdot 10^5$	84% $4{,}63 \cdot 10^5$	26% $3{,}19 \cdot 10^6$	34% $2{,}23 \cdot 10^6$
3589819	0% –	0% –	6% $1{,}64 \cdot 10^7$	1% $9{,}93 \cdot 10^7$

вании различных определений состояния. В дальнейшем будем использовать единственное состояние.

4.3 Сравнение метода EA + RL и его модификации с другими методами выбора критериев

В данном разделе производится сравнение различных алгоритмов генерации тестов и предложенного метода, а также его модификации. Рассматриваются следующие алгоритмы:

1. Генетический алгоритм с одним фиксированным критерием: в качестве функции приспособленности используется целевой критерий (время работы программы) или один из вспомогательных, применяются операторы мутации и скрещивания, описанные в разделе 4.1.3, размер популяции составляет 100 особей.

2. EA + RL: генетический алгоритм из пункта 1, в котором в качестве функции приспособленности используются критерии, выбираемые во время работы алгоритма Q-обучением [107].
3. Модификация EA + RL с сохранением лучшей особи и обучением на ошибках, описанная в разделе 3.4.4. Используются те же параметры, что и для EA + RL.
4. Случайный выбор и случайный выбор с сохранением лучшей особи: генетический алгоритм из пункта 1, в котором в качестве функции приспособленности используется критерий, случайно выбираемый в каждом поколении из множества всех критериев (целевого и вспомогательных);
5. SaRA: генетический алгоритм из пункта 1, в котором в качестве функции приспособленности поочередно используются все критерии с увеличением вычислительного бюджета, при этом при смене критерия происходит перезапуск генетического алгоритма [30].
6. Метод Jensen'a [69]: используется многокритериальный эволюционный алгоритм NSGA-II с теми же операторами и размером популяции, что и алгоритм из пункта 1. Одновременно оптимизируются два критерия: целевой критерий и вспомогательный критерий, выбираемый случайным образом.
7. Случайный поиск по целевому критерию: пока не достигнуто условие останова, генерируются случайные особи.

Каждый алгоритм запускался на каждой из тестируемых программ 20 раз. Вычислительный бюджет составлял один час. В таблице 9 указаны результаты генерации тестов, время измеряется в секундах. Отметим, что результаты для задач, использованных в предыдущих разделах, отличаются от результатов в данном разделе, так как используются разные вычислительные бюджеты и разные способы измерения времени. Использование астрономического времени для представления основных результатов обусловлено тем, что вычисление ФП для разных особей занимает разное время, в силу того, что именно это вре-

мя и является оптимизируемой величиной. По указанной причине число вычислений ФП не вполне адекватно отражает время, требуемое алгоритму для генерации тестов.

4.3.1 Результаты генерации тестов с помощью предложенного метода, его модификации и других методов выбора критериев

Рассмотрим результаты, представленные в таблице 9. Градации серого в порядке по убыванию от темного к светлому соответствуют первому, второму и третьему по эффективности результатам для каждой задачи. Можно видеть, что для каждой из рассмотренных программ наиболее эффективными алгоритмами как по проценту успешных запусков, так и по медианному времени работы (за исключением одной программы), является реализация метода EA + RL либо его модификации.

Из рассмотренных реализаций метода EA + RL наиболее эффективной является реализация его модификации. Она оказывается лучшим алгоритмом для пяти программ из семи по проценту запусков, в которых был найден сложный тест. Также этот алгоритм позволяет добиться существенного ускорения нахождения сложного теста по сравнению с ранее использовавшимся алгоритмом SaRA в четырех случаях.

В целом, предложенный метод и его модификация позволяют добиться более стабильной генерации сложных тестов за меньшее время, чем ранее использовавшийся алгоритм и существующие аналоги.

4.3.2 Проверка статистической значимости результатов

На основании рассмотрения результатов, представленных в таблице 9, можно выдвинуть гипотезу о том, что модификация метода EA + RL является предпочтительным методом для оценки эффективности программ решения задач дискретной математики. В данном разделе выполняется проверка статистической значимости этой гипотезы на основе имеющихся экспериментальных данных.

Таблица 9 – Результаты генерации тестов с помощью предложенного метода, его модификации и других методов выбора критериев

Метод	0937951	1700736	2208365	2558302	3589819	3600147	7294221
Модиф. EA + RL	100 % 172 с	100 % 492 с	75 % 1153 с	100 % 692 с	25 % > 3600 с	75 % 234 с	100 % 636 с
EA + RL	85 % 1404 с	0 % > 3600 с	100 % 626 с	100 % 364 с	0 % > 3600 с	100 % 177 с	40 % > 3600 с
Случайный выбор с сохр.	100 % 636 с	85 % 1114 с	5 % > 3600 с	85 % 744 с	0 % > 3600 с	80 % 1749 с	5 % > 3600 с
Случайный выбор	0 % > 3600 с	0 % > 3600 с	0 % > 3600 с	30 % > 3600 с	0 % > 3600 с	100 % 242 с	0 % > 3600 с
SaRA с перезапусками	100 % 690 с	100 % 840 с	100 % 1384 с	95 % 1721 с	0 % > 3600 с	100 % 143 с	20 % > 3600 с
Jensen	100 % 437 с	100 % 585 с	70 % 965 с	100 % 428 с	5 % > 3600 с	25 % > 3600 с	25 % > 3600 с
ГА, целевой критерий	0 % > 3600 с	10 % > 3600 с	20 % > 3600 с	95 % 474 с	0 % > 3600 с	10 % > 3600 с	5 % > 3600 с
Случайная генерация	0 % > 3600 с	0 % > 3600 с	0 % > 3600 с	0 % > 3600 с	0 % > 3600 с	0 % > 3600 с	0 % > 3600 с

Согласно исследованию [18], в общем случае для анализа результатов работы эволюционных алгоритмов следует использовать непараметрические тесты. В качестве непараметрического теста в настоящей диссертации используется непарный тест сумм рангов Уилкоксона, называемый также тестом Манна-Уитни [83], используется реализация теста из библиотеки языка программирования R [120]. Данный тест применяется, как правило, для выявления совпадения или различия медиан двух случайных величин X и Y на основе использования независимых выборок этих случайных величин. В рассматриваемом случае указанные случайные величины — это результаты работы двух различных алгоритмов при генерации сложных тестов для одной и той же программы. Более подробное описание этих случайных величин приводится далее.

Для функционирования теста требуется выполнение следующих предусловий:

– Все выборки из обеих случайных величин являются независимыми друг от друга. Это условие выполняется, так как каждый запуск каждого алгоритма на каждой программе использовал генератор случайных чисел, инициализированный и функционирующий независимо от других таких генераторов.
– На используемых значениях определено отношение полного порядка. В рассматриваемых экспериментах существует два различных типа результатов запуска алгоритма на программе:
  – был найден сложный тест (*успех*) — в этом случае результат определяется тем, за какое время был найден сложный тест;
  – сложный тест найден не был (*неудача*) — в этом случае результат определяется максимальным значением целевого критерия среди всех сгенерированных алгоритмом тестов.

  На этих исходах можно ввести отношение полного порядка, которое выглядит так:
  – любой успех лучше (больше) любой неудачи;
  – успех за время T_1 считается лучше (больше) успеха за время T_2, если $T_1 < T_2$ — иными словами, лучше находить сложные тесты раньше;
  – неудача с максимальным значением целевого критерия F_1 считается лучше (больше) неудачи со значением F_2, если $F_1 > F_2$ — лучше найти более качественный тест.

Следовательно, тест Уилкоксона применим для работы с результатами генерации сложных тестов.

Сравнение методов для оценки эффективности программ решения задач дискретной математики выполнялось путем последовательного запуска тестов Уилкоксона для результатов этих методов на каждой из семи рассматриваемых программ, решающих задачу «Ships. Version 2», после чего к полученным ре-

зультатам применялась коррекция Бонферрони [54], заключающаяся в домножении каждого из p-значений на число измерений (то есть, на семь; при превышении результата значения $1{,}0$ итоговое значение считается равным $1{,}0$). Пороговый уровень статистической значимости принимался равным $0{,}05$.

В таблицах 10 приводятся результаты определения статистической значимости различий между результатами модифицированного метода EA + RL и результатами остальных методов. В ячейках таблицы указываются p-значения с учетом коррекции Бонферрони. Зеленый цвет ячейки означает, что различие в соответствующем случае в пользу модифицированного метода EA + RL и оно статистически значимо. Красный цвет означает, что различие не в пользу модифицированного метода EA + RL и является статистически значимым. Статистически незначимым различиям соответствует белый цвет.

Таблица 10 – Статистическая значимость различий между модифицированным методом EA + RL и остальными методами

Метод	0937951	1700736	2208365	2558302	3589819	3600147	7294221
EA + RL	$5{,}8 \cdot 10^{-7}$	$1{,}0 \cdot 10^{-10}$	$6{,}4 \cdot 10^{-1}$	$5{,}7 \cdot 10^{-1}$	$2{,}6 \cdot 10^{-3}$	$1{,}0$	$3{,}4 \cdot 10^{-5}$
Случ. выбор с сохр.	$2{,}8 \cdot 10^{-6}$	$3{,}3 \cdot 10^{-2}$	$3{,}6 \cdot 10^{-7}$	$1{,}0$	$6{,}9 \cdot 10^{-8}$	$1{,}8 \cdot 10^{-1}$	$3{,}0 \cdot 10^{-9}$
Случайный выбор	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}9 \cdot 10^{-6}$	$1{,}0 \cdot 10^{-10}$	$1{,}0$	$1{,}0 \cdot 10^{-10}$
SaRA	$5{,}3 \cdot 10^{-6}$	$5{,}7 \cdot 10^{-1}$	$1{,}0$	$4{,}8 \cdot 10^{-5}$	$5{,}2 \cdot 10^{-8}$	$1{,}0$	$2{,}0 \cdot 10^{-10}$
Jensen	$1{,}5 \cdot 10^{-2}$	$1{,}0$	$1{,}0$	$3{,}7 \cdot 10^{-1}$	$4{,}7 \cdot 10^{-2}$	$8{,}5 \cdot 10^{-2}$	$1{,}4 \cdot 10^{-3}$
ГА, целевой критерий	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$4{,}3 \cdot 10^{-6}$	$1{,}0$	$1{,}1 \cdot 10^{-4}$	$1{,}2 \cdot 10^{-2}$	$1{,}0 \cdot 10^{-10}$
Случайная генерация	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$6{,}8 \cdot 10^{-9}$	$1{,}0 \cdot 10^{-10}$

Так как в таблице 10 отсутствует красный цвет, можно сделать вывод о том, что модификация метода EA + RL ни в одном случае не проигрывает другим методам статистически значимым образом. Однако, имеются методы, в некото-

рых случаях демонстрирующие сравнимые результаты, а именно: немодифицированный метод EA + RL, метод SaRA, использовавшийся ранее для генерации тестов [2], а также метод Jensen'a [69]. Для этих методов были построены аналогичные таблицы 11, wilcoxon-sara и 13 соответственно.

Таблица 11 – Статистическая значимость различий между методом EA + RL и остальными методами

Метод	0937951	1700736	2208365	2558302	3589819	3600147	7294221
Модиф. EA + RL	$5{,}8 \cdot 10^{-7}$	$1{,}0 \cdot 10^{-10}$	$6{,}4 \cdot 10^{-1}$	$5{,}7 \cdot 10^{-1}$	$2{,}6 \cdot 10^{-3}$	1,0	$3{,}4 \cdot 10^{-5}$
Случ. выбор с сохр.	$2{,}1 \cdot 10^{-1}$	$3{,}0 \cdot 10^{-9}$	$1{,}0 \cdot 10^{-10}$	$8{,}4 \cdot 10^{-1}$	$3{,}6 \cdot 10^{-4}$	$4{,}8 \cdot 10^{-4}$	$1{,}6 \cdot 10^{-7}$
Случайный выбор	$1{,}0 \cdot 10^{-10}$	$5{,}3 \cdot 10^{-6}$	$1{,}0 \cdot 10^{-10}$	$2{,}8 \cdot 10^{-7}$	$1{,}0 \cdot 10^{-10}$	1,0	$2{,}0 \cdot 10^{-10}$
SaRA	$3{,}4 \cdot 10^{-1}$	$1{,}0 \cdot 10^{-10}$	$3{,}3 \cdot 10^{-2}$	$6{,}8 \cdot 10^{-9}$	$3{,}3 \cdot 10^{-3}$	1,0	$4{,}8 \cdot 10^{-4}$
Jensen	$6{,}1 \cdot 10^{-2}$	$1{,}0 \cdot 10^{-10}$	$5{,}0 \cdot 10^{-1}$	1,0	1,0	$4{,}1 \cdot 10^{-5}$	1,0
ГА, целевой критерий	$1{,}0 \cdot 10^{-10}$	$2{,}0 \cdot 10^{-5}$	$4{,}6 \cdot 10^{-9}$	1,0	1,0	$1{,}9 \cdot 10^{-6}$	$2{,}8 \cdot 10^{-8}$
Случайная генерация	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$

Из приведенных таблиц видно, что перечисленные выше методы проигрывают модификации метода EA + RL всякий раз, когда различия являются статистически значимыми. Также можно отметить, что метод SaRA чаще проигрывает немодифицированному методу EA + RL и методу Jensen'a, немодифицированный метод EA + RL демонстрирует слабые результаты в отношении программы 1700736, а метод Jensen'a — в отношении программы 3600147.

Таким образом, исходя из анализа результатов экспериментов, в том числе из анализа статистической значимости различий, можно заключить, что модификация метода EA + RL действительно демонстрирует лучшие результаты по сравнению с остальными рассмотренными методами.

Таблица 12 – Статистическая значимость различий между методом SaRA и остальными методами

Метод	0937951	1700736	2208365	2558302	3589819	3600147	7294221
Модиф. EA + RL	$5{,}3 \cdot 10^{-6}$	$5{,}7 \cdot 10^{-1}$	1,0	$4{,}8 \cdot 10^{-5}$	$5{,}2 \cdot 10^{-8}$	1,0	$2{,}0 \cdot 10^{-10}$
EA + RL	$3{,}4 \cdot 10^{-1}$	$1{,}0 \cdot 10^{-10}$	$3{,}3 \cdot 10^{-2}$	$6{,}8 \cdot 10^{-9}$	$3{,}3 \cdot 10^{-3}$	1,0	$4{,}8 \cdot 10^{-4}$
Случ. выбор с сохр.	1,0	$8{,}0 \cdot 10^{-1}$	$1{,}0 \cdot 10^{-10}$	$7{,}2 \cdot 10^{-2}$	1,0	$9{,}4 \cdot 10^{-7}$	$2{,}8 \cdot 10^{-7}$
Случайный выбор	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$7{,}3 \cdot 10^{-4}$	$1{,}0 \cdot 10^{-10}$	1,0	$1{,}9 \cdot 10^{-6}$
Jensen	$8{,}4 \cdot 10^{-1}$	1,0	1,0	$9{,}4 \cdot 10^{-7}$	$2{,}2 \cdot 10^{-2}$	$1{,}4 \cdot 10^{-5}$	$2{,}5 \cdot 10^{-2}$
ГА, целевой критерий	$1{,}0 \cdot 10^{-10}$	$4{,}1 \cdot 10^{-10}$	$9{,}3 \cdot 10^{-8}$	$1{,}7 \cdot 10^{-4}$	$6{,}0 \cdot 10^{-1}$	$7{,}4 \cdot 10^{-7}$	$9{,}3 \cdot 10^{-8}$
Случайная генерация	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$4{,}1 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}2 \cdot 10^{-9}$

Таблица 13 – Статистическая значимость различий между методом Jensen’a и остальными методами

Метод	0937951	1700736	2208365	2558302	3589819	3600147	7294221
Модиф. EA + RL	$1{,}5 \cdot 10^{-2}$	1,0	1,0	$3{,}7 \cdot 10^{-1}$	$4{,}7 \cdot 10^{-2}$	$8{,}5 \cdot 10^{-2}$	$1{,}4 \cdot 10^{-3}$
EA + RL	$6{,}1 \cdot 10^{-2}$	$1{,}0 \cdot 10^{-10}$	$5{,}0 \cdot 10^{-1}$	1,0	1,0	$4{,}1 \cdot 10^{-5}$	1,0
Случ. выбор с сохр.	1,0	$1{,}7 \cdot 10^{-1}$	$6{,}8 \cdot 10^{-9}$	$3{,}4 \cdot 10^{-1}$	$3{,}7 \cdot 10^{-3}$	$1{,}7 \cdot 10^{-1}$	$6{,}9 \cdot 10^{-8}$
Случайный выбор	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$5{,}8 \cdot 10^{-7}$	$1{,}2 \cdot 10^{-9}$	$3{,}4 \cdot 10^{-5}$	$4{,}1 \cdot 10^{-10}$
SaRA	$8{,}4 \cdot 10^{-1}$	1,0	1,0	$9{,}4 \cdot 10^{-7}$	$2{,}2 \cdot 10^{-2}$	$1{,}4 \cdot 10^{-5}$	$2{,}5 \cdot 10^{-2}$
ГА, целевой критерий	$1{,}0 \cdot 10^{-10}$	$7{,}1 \cdot 10^{-10}$	$1{,}2 \cdot 10^{-6}$	1,0	1,0	$2{,}6 \cdot 10^{-1}$	$6{,}9 \cdot 10^{-8}$
Случайная генерация	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$1{,}0 \cdot 10^{-10}$	$9{,}3 \cdot 10^{-8}$	$1{,}0 \cdot 10^{-10}$

4.4 Итоги внедрения результатов диссертационной работы в систему *Timus Online Judge*

Приведенные в данной главе результаты внедрены при разработке тестов для архива задач с проверяющей системой *Timus Online Judge*, функционирующего на базе Уральского федерального университета имени первого Президента России Б. Н. Ельцина, г. Екатеринбург, используемого для подготовки к олимпиадам по программированию. По состоянию на первое сентября 2017 года по задаче «Ships. Version 2»была принята 31 программа, решающая эту задачу. Согласно используемым на тот момент тестам, в частности, тестам, сгенерированным с помощью ранее использовавшегося подхода [2], все программы были признаны эффективными. В результате применения предложенного в диссертации метода было получено три сложных теста, которые выявили неэффективность всех считавшихся эффективными программ. Тесты получили номера 80–82.

Отметим, что предложенный подход оказался не только более эффективным, чем ранее использовавшийся, но и более автоматизированным. В предыдущем подходе счетчики размещались в коде тестируемых программ вручную, что требовало непосредственного интеллектуального участия человека. В предложенном подходе счетчики вставлялись автоматически, как описано в разделе 4.1.4. Эффективный автоматический выбор и использование вставленных счетчиков оказались возможными благодаря предложенному методу выбора вспомогательных критериев. Таким образом, участие человека в оценке эффективности программ, решающих рассматриваемую задачу, свелось к минимуму, связанному с размещением тестов в системе. Автоматизация оставшейся стадии процесса тестирования является стандартной инженерной задачей и не входит в круг научных проблем, рассматриваемых в диссертации.

Выводы по главе 4

Описана методология применения предложенного метода выбора вспомогательных критериев к генерации тестов, позволяющих оценить эффективность программ решения задач дискретной математики. Также приведены результаты генерации тестов на примере задачи «Ships. Version 2» [121], полученные с помощью предложенного метода и существующих аналогов. Показаны преимущества предложенного метода по сравнению с аналогами, полученные результаты обладают статистической значимостью.

Описанный способ генерации тестов опубликован в трудах ряда международных конференций, индексируемых в системе *Scopus* [31, 34, 39, 116]. Приведенные в данной главе результаты внедрены при разработке тестов для архива задач с проверяющей системой *Timus Online Judge*, функционирующего на базе Уральского федерального университета имени первого Президента России Б. Н. Ельцина, г. Екатеринбург, используемого для подготовки к олимпиадам по программированию.

ЗАКЛЮЧЕНИЕ

В результате диссертационного исследования получены следующие результаты:

1. Разработаны метод и программная реализация адаптивного выбора вспомогательных критериев, используемых в эволюционных алгоритмах при генерации тестов, оценивающих эффективность программ решения задач дискретной математики.
2. Определены свойства вспомогательных критериев, используемых при генерации тестов, и получены асимптотические оценки времени работы алгоритма, реализующего предложенный метод, для мешающих и эффективных вспомогательных критериев, обладающих этими свойствами.На основе выполненного анализа предложена модификация разработанного алгоритма.
3. Предложенный метод позволил добиться за меньшее время более стабильной генерации тестов, оценивающих эффективность программ решения задач дискретной математики, на примере NP-трудной олимпиадной задачи по программированию «Ships. Version 2», по сравнению с другими подходами.

Таким образом, выполнены все задачи диссертации, что позволило достичь поставленную цель сокращения общего времени, необходимого для осуществления автоматизированной оценки эффективности программ решения задач дискретной математики.

Возможным направлением дальнейших исследований является разработка методов коэволюции программ и тестов, что может позволить не только оценивать эффективность существующих программ, но и повышать ее путем изменения их кода.

Результаты диссертации использовались в учебном процессе кафедры «Компьютерные технологии» Университета ИТМО при руководстве следующими бакалаврскими работами и магистерскими диссертациями:

1. В. С. Кононов. Исследование влияния характеристик обучения с подкреплением на эффективность выбора операторов в эволюционных алгоритмах. Бакалаврская работа, 2013 г.
2. И. А. Петрова. Повышение эффективности алгоритмов многокритериальной оптимизации с помощью машинного обучения для решения задач составления расписаний. Бакалаврская работа, 2013 г.
3. Д. С. Антипов. Асимптотическая оценка времени работы (1+1) эволюционной стратегии, управляемой алгоритмом Q-learning, на примере задачи OneMax+ZeroMax. Бакалаврская работа, 2014 г.
4. Н. С. Буланова. Исследование эффективности применения вспомогательных оптимизируемых величин при использовании методов оптимизации на основе искусственных иммунных систем. Магистерская диссертация, 2015 г.
5. А. А. Матвеева. Выбор вспомогательных оптимизируемых величин в эволюционных алгоритмах при помощи многокритериального обучения с подкреплением. Магистерская диссертация, 2015 г.
6. И. А. Петрова. Выбор вспомогательных оптимизируемых величин для повышения эффективности эволюционных алгоритмов в нестационарных условиях. Магистерская диссертация, 2015 г.
7. А. Ю. Рост. Адаптивная настройка параметров эволюционных алгоритмов с помощью обучения с подкреплением. Магистерская диссертация, 2015 г.
8. А. О. Басин. Выбор вспомогательных оптимизируемых величин на основе информации о ландшафте приспособленности особей эволюционного алгоритма. Магистерская диссертация, 2016 г.

СПИСОК ЛИТЕРАТУРЫ

1 *Афанасьева А. С.*, *Буздалов М. В.* Выбор функции приспособленности особей генетического алгоритма с помощью обучения с подкреплением // Научно-технический вестник информационных технологий, механики и оптики. — 2012. — 1(77). — С. 77–81.

2 *Буздалов М. В.* Генерация тестов для определения неэффективных решений олимпиадных задач по программированию с использованием эволюционных алгоритмов : дис. … канд. / Буздалов Максим Викторович. — СПбНИУ ИТМО, 12.2014.

3 *Буздалова А. С.*, *Буздалов М. В.* Метод повышения эффективности эволюционных алгоритмов с помощью обучения с подкреплением // Научно-технический вестник информационных технологий, механики и оптики. — 2012. — 5 (81). — С. 115–119.

4 *Воронцов К. В.*, *Каневский Д. Ю.* Коэволюционный метод обучения алгоритмических композиций // Таврический вестник информатики и математики. — 2005. — Т. 2. — С. 51–64.

5 *Гладков Л. А.*, *Курейчик В. В.*, *Курейчик В. М.* Генетические алгоритмы. — Физматлит, 2010. — 368 с.

6 *Евтушенко Ю. Г.*, *Жадан В. Г.* Точные вспомогательные функции в задачах оптимизации // Журнал вычислительной математики и математической физики. — 1990. — Т. 30, № 1. — С. 43–57.

7 *Еремеев А. В.* Исследование эволюционных методов решения задач комбинаторной оптимизации : дис. … д-ра / Еремеев Антон Валентинович. — Институт математики им. Соболева Сибирского отделения РАН, 12.2013.

8 *Кормен Т.*, *Лейзерсон Ч.*, *Ривест Р.* Алгоритмы. Построение и анализ. Второе издание. — М. : Издательский дом «Вильямс», 2005. — 1296 с.

9 *Курейчик В. М.*, *Кажаров А. А.*, *Ляпунова И. А.* Определение зависимости параметров муравьиного алгоритма от исходных данных // Вестник Ростовского государственного университета путей сообщения. — 2014. — Т. 56, № 4. — С. 63–70.

10 *Курейчик В. М.*, *Каланчук С. А.* Обзор и состояние проблемы роевых методов оптимизации // Информатика, вычислительная техника и инженерное образование. — 2016. — Т. 25, № 1. — С. 1–13.

11 *Николенко С. И.*, *Тулупьев А. Л.* Самообучающиеся системы. — М., 2009.

12 *Петрова И. А.*, *Буздалова А. С.*, *Шалыто А. А.* Метод динамического выбора вспомогательных критериев в многокритериальных эволюционных алгоритмах // Научно-технический вестник информационных технологий, механики и оптики. — 2016. — 3(16). — С. 460–466.

13 *Хритоненко Д. И.*, *Семенкин Е. С.* Адаптивная мутация в самоконфигурируемых эволюционных алгоритмах // Системы управления и информационные технологии. — 2017. — Т. 69, № 3. — С. 37–42.

14 *Чивилихин Д. С.* Генерация конечных автоматов на основе муравьиных алгоритмов : дис. … канд. / Чивилихин Даниил Сергеевич. — СПбНИУ ИТМО, 12.2015.

15 *Скобцов Ю. А.* Основы эволюционных вычислений. — Донецк : ДонНТУ, 2008.

16 A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II / K. Deb, A. Pratap, S. Agarwal, T. Meyarivan // IEEE Transactions on Evolutionary Computation. — 2002. — Vol. 6, no. 2. — P. 182–197.

17 A Method to Control Parameters of Evolutionary Algorithms by Using Reinforcement Learning / Y. Sakurai, K. Takada, T. Kawabe, S. Tsuruta // Proceedings of 2010 Sixth International Conference on Signal-Image Technology and Internet-Based Systems (SITIS). — 2010. — P. 74–79.

18 A Practical Tutorial on the Use of Nonparametric Statistical Tests as a Methodology for Comparing Evolutionary and Swarm Intelligence Algorithms / J. Derrac, S. Garcia, D. Molina, F. Herrera // Swarm and Evolutionary Computation. — 2011. — Vol. 1, no. 1. — P. 3–18.

19 *Abbass H. A.*, *Sarker R.*, *Newton C.* PDE: A Pareto Frontier Differential Evolution Approach for Multiobjective Optimization Problems // Proceedings of the Congress on Evolutionary Computation. — IEEE Press, 2001. — P. 971–978.

20 *Afanasyeva A.*, *Buzdalov M.* Choosing Best Fitness Function with Reinforcement Learning // Proceedings of the Tenth International Conference on Machine Learning and Applications. Vol. 2. — Honolulu, HI, USA : IEEE Computer Society, 2011. — P. 354–357.

21 *Afanasyeva A.*, *Buzdalov M.* Optimization with Auxiliary Criteria using Evolutionary Algorithms and Reinforcement Learning // Proceedings of 18th International Conference on Soft Computing MENDEL 2012. — Brno, Czech Republic, 2012. — P. 58–63.

22 *Alanazi F.*, *Lehre P. K.* Runtime analysis of selection hyper-heuristics with classical learning mechanisms // 2014 IEEE Congress on Evolutionary Computation (CEC). — 2014. — P. 2515–2523.

23 *Alanazi F.*, *Lehre P. K.* Limits to Learning in Reinforcement Learning Hyper-heuristics // Evolutionary Computation in Combinatorial Optimization: 16th European Conference, EvoCOP Proceedings. — Springer, 2016. — P. 170–185.

24 *Alander J. T.*, *Mantere T.*, *Moghadampour G.* Testing Software Response Times Using a Genetic Algorithm // Proceedings of the 3rd Nordic Workshop on Genetic Algorithms and their Applications. — 1997. — P. 293–298.

25 *Alander J. T.*, *Mantere T.*, *Turunen P.* Genetic Algorithm Based Software Testing // Artificial Neural Nets and Genetic Algorithms. — Springer-Verlag, 1998. — P. 325–328.

26 Analyzing Bandit-based Adaptive Operator Selection Mechanisms / Á. Fialho, L. Da Costa, M. Schoenauer, M. Sebag // Annals of Mathematics and Artificial Intelligence. — 2010. — Vol. 60, no. 1/2. — P. 25–64.

27 *Auer P.*, *Cesa-Bianchi N.*, *Fischer P.* Finite-time Analysis of the Multiarmed Bandit Problem // Machine Learning. — 2002. — Vol. 47, no. 2. — P. 235–256.

28 *Burnim J.*, *Sen K.* Heuristics for Scalable Dynamic Test Generation // Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering. — 2008. — P. 443–446.

29 *Buzdalov M.*, *Buzdalova A.*, *Shalyto A.* A First Step towards the Runtime Analysis of Evolutionary Algorithm Adjusted with Reinforcement Learning // Proceedings of the International Conference on Machine Learning and Applications. Vol. 1. — IEEE Computer Society, 2013. — P. 203–208.

30 *Buzdalov M.* A Switch-and-Restart Algorithm with Exponential Restart Strategy for Objective Selection and its Runtime Analysis // Proceedings of the International Conference on Machine Learning and Applications. — IEEE Computer Society, 2014. — P. 141–146.

31 *Buzdalov M.*, *Buzdalova A.* Adaptive Selection of Helper-Objectives for Test Case Generation // 2013 IEEE Congress on Evolutionary Computation. Vol. 1. — 2013. — P. 2245–2250.

32 *Buzdalov M.*, *Buzdalova A.* OneMax Helps Optimizing XdivK: Theoretical Runtime Analysis for RLS and EA+RL // Proceedings of Genetic and Evolutionary Computation Conference Companion. — ACM, 2014. — P. 201–202.

33 *Buzdalov M.*, *Buzdalova A.* Analysis of Q-Learning with Random Exploration for Selection of Auxiliary Objectives in Random Local Search // Proceedings of IEEE Congress on Evolutionary Computation. — 2015. — P. 1776–1783.

34 *Buzdalov M.*, *Buzdalova A.*, *Petrova I.* Generation of Tests for Programming Challenge Tasks Using Multi-Objective Optimization // Proceedings of Genetic and Evolutionary Computation Conference Companion. — ACM, 2013. — P. 1655–1658.

35 *Buzdalova A.*, *Bulanova N.* Selection of Auxiliary Objectives in Artificial Immune Systems: Initial Explorations // Proceedings of International Conference on Soft Computing MENDEL. — 2015. — P. 47–52.

36 *Buzdalova A.*, *Buzdalov M.* Adaptive Selection of Helper-Objectives with Reinforcement Learning // Proceedings of the International Conference on Machine Learning and Applications. Vol. 2. — IEEE Computer Society, 2012. — P. 66–67.

37 *Buzdalova A.*, *Buzdalov M.* Increasing Efficiency of Evolutionary Algorithms by Choosing between Auxiliary Fitness Functions with Reinforcement Learning // Proceedings of the International Conference on Machine Learning and Applications. Vol. 1. — 2012. — P. 150–155.

38 *Buzdalova A.*, *Buzdalov M.* A New Algorithm for Adaptive Online Selection of Auxiliary Objectives // Proceedings of International Conference on Machine Learning and Applications. — 2014. — P. 584–587.

39 *Buzdalova A.*, *Buzdalov M.*, *Parfenov V.* Generation of Tests for Programming Challenge Tasks Using Helper-Objectives // 5th International Symposium on Search-Based Software Engineering. — Springer, 2013. — P. 300–305. — (Lecture Notes in Computer Science ; 8084).

40 *Buzdalova A.*, *Kononov V.*, *Buzdalov M.* Selecting Evolutionary Operators using Reinforcement Learning: Initial Explorations // Proceedings of Genetic and Evolutionary Computation Conference Companion. — 2014. — P. 1033–1036.

41 *Buzdalova A.*, *Matveeva A.*, *Korneev G.* Selection of Auxiliary Objectives with Multi-Objective Reinforcement Learning // Proceedings of Genetic and Evolutionary Computation Conference Companion. — 2015. — P. 1177–1180.

42 *Buzdalova A.*, *Petrova I.*, *Buzdalov M.* Runtime Analysis of Different Approaches to Select Conflicting Auxiliary Objectives in the Generalized One-Max Problem // Proceedings of IEEE Symposium Series on Computational Intelligence. — 2016. — P. 280–286.

43 *Coello C. A. C.*, *Lamont G. B.*, *Veldhuizen D. A. V.* Evolutionary Algorithms for Solving Multi-Objective Problems. — Second. — Springer-Verlag, 2006. — 800 p.

44 Coevolutionary Principles / E. Popovici, A. Bucci, R. P. Wiegand, E. de Jong // Handbook of Natural Computing. — Springer. — P. 987–1033.

45 Convergence Results for Single-Step On-Policy Reinforcement Learning Algorithms / S. Singh, T. Jaakkola, M. L. Littman, C. Szepesvári // Machine Learning. — Hingham, MA, USA, 2000. — Vol. 38, no. 3. — P. 287–308.

46 *Corne D. W.*, *Knowles J. D.*, *Oates M. J.* The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization // Parallel Problem Solving from Nature – PPSN VI. — Springer, 2000. — P. 839–848. — (Lecture Notes in Computer Science ; 1917).

47 *Danielsson N. A.* Lightweight Semiformal Time Complexity Analysis for Purely Functional Data Structures // SIGPLAN Not. — 2008. — Vol. 43, no. 1. — P. 133–144.

48 *Doerr B.*, *Doerr C.* Optimal Parameter Choices Through Self-Adjustment: Applying the 1/5-th Rule in Discrete Settings // Proceedings of Genetic and Evolutionary Computation Conference. — 2015. — P. 1335–1342.

49 *Doerr B.*, *Doerr C.*, *Ebel F.* From black-box complexity to designing new genetic algorithms // Theoretical Computer Science. — 2015. — Vol. 567. — P. 87–104.

50 *Doerr C.* Non-Static Parameter Choices in Evolutionary Computation // Proceedings of Genetic and Evolutionary Computation Companion. — 2017. — P. 736–761.

51 *Dorigo M.* The ant system: Optimization by colony of cooperating agents // IEEE Transactions on Systems, Man and Cybernetics. — 1996. — Vol. 26, no. 1. — P. 1–13.

52 *Dorigo M.*, *Gambardella L. M.* 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.

53 *Droste S.*, *Jansen T.*, *Wegener I.* On the analysis of the (1+1) evolutionary algorithm // Theor. Comput. Sci. — 2002. — Vol. 276, no. 1/2. — P. 51–81.

54 *Dunn O. J.* Multiple Comparisons Among Means // Journal of the American Statistical Association. — 1961. — Vol. 56, no. 293. — P. 52–64.

55 *Eiben A. E.*, *Smith J. E.* Introduction to Evolutionary Computing. — Berlin, Heidelberg, New York : Springer-Verlag, 2015. — 287 p.

56 *Eiben A. E.*, *Smith J. E.* Introduction to Evolutionary Computing. — Springer, 2015. — 287 p.

57 *Eiben A.*, *Rudolph G.* Theory of evolutionary algorithms: a bird's eye view // Theoretical Computer Science. — 1999. — Vol. 229, no. 1. — P. 3–9.

58 Escaping Local Optima with Diversity Mechanisms and Crossover / D.-C. Dang, T. Friedrich, T. Kötzing, M. S. Krejca, P. K. Lehre, P. S. Oliveto, D. Sudholt, A. M. Sutton // Proceedings of Genetic and Evolutionary Computation Conference. — 2016. — P. 645–652.

59 *Even-Dar E.*, *Mansour Y.* Learning Rates for Q-learning // Journal of Machine Learning Research. — 2004. — Vol. 5. — P. 1–25.

60 *Gosavi A.* Reinforcement Learning: A Tutorial. Survey and Recent Advances // INFORMS Journal on Computing. — 2009. — Vol. 21, no. 2. — P. 178–192.

61 *Gross H.-G.* Measuring Evolutionary Testability of Real-Time Software : PhD thesis / Gross Hans-Gerhard. — University of Glamorgan, 06/2000.

62 *Gross H.-G.* A Prediction System for Evolutionary Testability Applied to Dynamic Execution Time Analysis // Information and Software Technology. — 2001. — Vol. 43, no. 14. — P. 855–862.

63 *Gross H.-G.*, *Jones B. F.*, *Eyres D. E.* Structural Performance Measure of Evolutionary Testing Applied to Worst-Case Timing of Real-Time Systems // IEEE Proceedings – Software. — 2000. — Vol. 147, no. 2. — P. 25–30.

64 *Gross H.-G.*, *Mayer N.* Evolutionary Testing in Component-based Real-Time System Construction // Proceedings of Genetic and Evolutionary Computation Conference. — 2002. — P. 207–214.

65 *Gross H.-G.*, *Mayer N.* Search-based Execution-Time Verification in Object-Oriented and Component-Based Real-Time System Development // Proceedings of the 8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. — 2003. — P. 113–120.

66 *Handl J.*, *Lovell S. C.*, *Knowles J. D.* Multiobjectivization by Decomposition of Scalar Cost Functions // Parallel Problem Solving from Nature – PPSN X. — Springer, 2008. — P. 31–40. — (Lecture Notes in Computer Science ; 5199).

67 *Jansen T.* Analyzing Evolutionary Algorithms. — Springer, 2013. — 258 p.

68 *Jansen T.*, *Wegener I.* The Analysis of Evolutionary Algorithms–A Proof that Crossover Really Can Help // Algorithmica. — 2002. — Vol. 34. — P. 47–66.

69 *Jensen M. T.* Helper-Objectives: Using Multi-Objective Evolutionary Algorithms for Single-Objective Optimisation: Evolutionary Computation Combinatorial Optimization // Journal of Mathematical Modelling and Algorithms. — 2004. — Vol. 3, no. 4. — P. 323–347.

70 *Kaelbling L. P.*, *Littman M. L.*, *Moore A. W.* Reinforcement Learning: A Survey // Journal of Artificial Intelligence Research. — 1996. — Vol. 4. — P. 237–285.

71 *Karafotias G.*, *Hoogendoorn M.*, *Eiben A.* Parameter Control in Evolutionary Algorithms: Trends and Challenges // IEEE Transactions on Evolutionary Computation. — 2014. — Vol. 18, no. 2. — P. 167–187.

72 *Karafotias G.*, *Eiben Á. E.*, *Hoogendoorn M.* Generic parameter control with reinforcement learning // Proceedings of Genetic and Evolutionary Computation Conference. — 2014. — P. 1319–1326.

73 *Kearns M.*, *Singh S.* Finite-Sample Convergence Rates for Q-learning and Indirect Algorithms // Proceedings of the 1998 conference on Advances in neural information processing systems II. — Cambridge, MA, USA : MIT Press, 1999. — P. 996–1002.

74 *Knowles J. D.*, *Watson R. A.*, *Corne D.* Reducing Local Optima in Single-Objective Problems by Multi-objectivization // Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization. — Springer-Verlag, 2001. — P. 269–283.

75 *Knowles J. D.*, *Corne D. W.* Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy // Evolutionary Computation. — 2000. — Vol. 8, no. 2. — P. 149–172.

76 *Langdon W. B.*, *Harman M.* Genetically Improving 50000 Lines of C++: Research Note RN/12/09: tech. rep. — 2012.

77 *Lehre P.*, *Yao X.* Runtime analysis of (1+1) EA on computing unique input output sequences // 2007 IEEE Congress on Evolutionary Computation, CEC 2007. — 09/2007. — P. 1882–1889.

78 *Liskowski P.*, *Krawiec K.* Online Discovery of Search Objectives for Test-Based Problems // Evolutionary Computation. — 2017. — Vol. 25, no. 3. — P. 375–406.

79 *Lochtefeld D. F.*, *Ciarallo F. W.* Deterministic Helper-Objective Sequences Applied to Job-Shop Scheduling // Proceedings of Genetic and Evolutionary Computation Conference. — ACM, 2010. — P. 431–438.

80 *Lochtefeld D. F.*, *Ciarallo F. W.* Helper-Objective Optimization Strategies for the Job-Shop Scheduling Problem // Applied Soft Computing. — 2011. — Vol. 11, no. 6. — P. 4161–4174.

81 *Lochtefeld D. F.*, *Ciarallo F. W.* Multiobjectivization via helper objectives with the tunable objectives problem // IEEE Trans. Evolutionary Computation. — 2012. — Vol. 16, no. 3. — P. 373–390.

82 *Luke S.* Essentials of Metaheuristics. — Lulu, 2009. — 253 p.

83 *Mann H. B.*, *Whitney D. R.* On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other // Annals of Mathematical Statistics. — 1947. — Vol. 18, no. 1. — P. 50–60.

84 *Mitchell D.*, *Selman B.*, *Levesque H.* Hard and Easy Distributions of SAT Problems // Proceedings of AAAI Conference on Artificial Intelligence. — 1992. — P. 459–465.

85 *Mitchell M.* An Introduction to Genetic Algorithms. — Cambridge, MA : MIT Press, 1996. — 221 p.

86 *Mueller S.*, *Schraudolph N. N.*, *Koumoutsakos P. D.* Step Size Adaptation in Evolution Strategies using Reinforcement Learning // Proceedings of the Congress on Evolutionary Computation. — IEEE, 2002. — P. 151–156.

87 *Neumann F.*, *Wegener I.* Can Single-Objective Optimization Profit from Multiobjective Optimization? // Multiobjective Problem Solving from Nature. — Springer Berlin Heidelberg, 2008. — P. 115–130. — (Natural Computing Series).

88 *Oliveto P. S.*, *He J.*, *Yao X.* Time Complexity of Evolutionary Algorithms for Combinatorial Optimization: A Decade of Results // International Journal of Automation and Computing. — 2007. — Vol. 4, no. 3. — P. 281–293.

89 On the Effects of Adding Objectives to Plateau Functions / D. Brockhoff, T. Friedrich, N. Hebbinghaus, C. Klein, F. Neumann, E. Zitzler // IEEE Transactions on Evolutionary Computation. — 2009. — Vol. 13, no. 3. — P. 591–603.

90 PAC Model-free Reinforcement Learning / A. L. Strehl, L. Li, E. Wiewiora, J. Langford, M. L. Littman // Proceedings of the 23rd International Conference on Machine Learning. — 2006. — P. 881–888.

91 *Parr T.* The Definitive ANTLR Reference: Building Domain-Specific Languages. — Pragmatic Bookshelf, 2007.

92 PathCrawler: Automatic Generation of Path Tests by Combining Static and Dynamic Analysis / N. Williams, B. Marre, P. Mouy, M. Roger // Dependable Computing – EDCC 5. — 2005. — P. 281–292. — (Lecture Notes in Computer Science ; 3463).

93 PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization / D. W. Corne, N. R. Jerram, J. D. Knowles, M. J. Oates // Proceedings of Genetic and Evolutionary Computation Conference. — Morgan Kaufmann Publishers, 2001. — P. 283–290.

94 *Petrova I.*, *Buzdalova A.* Reinforcement learning based dynamic selection of auxiliary objectives with preservation of the best found solution // Proceedings of the Genetic and Evolutionary Computation Conference Companion. — 2017. — P. 1435–1438.

95 *Petrova I.*, *Buzdalova A.*, *Buzdalov M.* Improved Helper-Objective Optimization Strategy for Job-Shop Scheduling Problem // Proceedings of the International Conference on Machine Learning and Applications. Vol. 2. — IEEE Computer Society, 2013. — P. 374–377.

96 *Petrova I.*, *Buzdalova A.* Selection of Auxiliary Objectives in the Travelling Salesman Problem using Reinforcement Learning // Proceedings of Genetic and Evolutionary Computation Conference Companion. — 2015. — P. 1455–1456.

97 *Petrova I.*, *Buzdalova A.*, *Korneev G.* Runtime analysis of random local search with reinforcement based selection of non-stationary auxiliary objectives: initial study // Proceedings of 22nd International Conference on Soft Computing MENDEL 2016. — Czech Republic, 2016. — P. 95–102.

98 *Pettinger J. E.*, *Everson R. M.* Controlling Genetic Algorithms with Reinforcement Learning // Proceedings of Genetic and Evolutionary Computation Conference. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2002. — P. 692.

99 *Pisinger D.* Algorithms for Knapsack Problems : PhD thesis / Pisinger David. — University of Copenhagen, 02/1995.

100 Reinforcement Learning for Online Control of Evolutionary Algorithms / A. E. Eiben, M. Horvath, W. Kowalczyk, M. C. Schut // Proceedings of the 4th international conference on Engineering self-organising systems. — Springer-Verlag, Berlin, Heidelberg, 2006. — P. 151–160.

101 *Rice H. G.* Classes of Recursively Enumerable Sets and Their Decision Problems // Transactions of the American Mathematical Society. — 1953. — Vol. 74, no. 2. — P. 358–366.

102 *Schwartz A.* A Reinforcement Learning Method for Maximizing Undiscounted Rewards // Proceedings of the Tenth International Conference on Machine Learning. — 1993. — P. 298–305.

103 *Sen K.*, *Marinov D.*, *Agha G.* CUTE: A Concolic Unit Testing Engine for C // SIGSOFT Software Engineering Notes. — New York, NY, USA, 2005. — Vol. 30, no. 5. — P. 263–272.

104 *Srinivas N.*, *Deb K.* Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms // Evolutionary Computation. — 1994. — Vol. 2, no. 3. — P. 221–248.

105 *Stanovov V. V.*, *Sopov E. A.*, *Semenkin E. S.* Multi-Strategy Multimodal Genetic Algorithm for Designing Fuzzy Rule Based Classifiers // Proceedings of IEEE Symposium Series on Computational Intelligence. — 2015. — P. 167–173.

106 *Sudholt D.* Crossover speeds up building-block assembly // Proceedings of Genetic and Evolutionary Computation Conference. — 2012. — P. 689–696.

107 *Sutton R. S.*, *Barto A. G.* Reinforcement Learning: An Introduction. — Cambridge, MA, USA : MIT Press, 1998.

108 Testing Real-Time Systems using Genetic Algorithms / J. Wegener, H. Sthamer, B. F. Jones, D. E. Eyres // Software Quality Journal. — 1997. — Vol. 6, no. 2. — P. 127–135.

109 *Tillmann N.*, *Halleux J. de.* Pex – White Box Test Generation for .NET // Tests And Proofs. Second International Conference. — 2008. — P. 134–153.

110 *Tlili M.*, *Wappler S.*, *Sthamer H.* Improving Evolutionary Real-Time Testing // Proceedings of Genetic and Evolutionary Computation Conference. — ACM, 2006. — P. 1917–1924.

111 Using multi-objective evolutionary algorithms for single-objective constrained and unconstrained optimization / C. Segura, C. A. C. Coello, G. Miranda, C. León // Annals of Operations Research. — 2016. — Vol. 240, no. 1. — P. 217–250.

112 Using multi-objective evolutionary algorithms for single-objective optimization / C. Segura, C. A. C. Coello, G. Miranda, C. Léon // 4OR. — 2013. — Vol. 3, no. 11. — P. 201–228.

113 *Weegen E. van der*, *McKinna J.* A Machine-Checked Proof of the Average-Case Complexity of Quicksort in Coq // Types for Proofs and Programs: International Conference, TYPES 2008 Torino, Italy, March 26-29, 2008 Revised Selected Papers. — Springer Berlin Heidelberg, 2009. — P. 256–271.

114 *Wegener J.*, *Mueller F.* A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints // Real-Time Systems. — 2001. — Vol. 21, no. 3. — P. 241–268.

115 *Wessing S.*, *Preuss M.*, *Trautmann H.* Stopping Criteria for Multimodal Optimization // Parallel Problem Solving from Nature – PPSN XIII. — Springer, 2014. — P. 141–150. — (Lecture Notes in Computer Science ; 8672).

116 Worst-Case Execution Time Test Generation using Genetic Algorithms with Automated Construction and Online Selection of Objectives / N. Kravtsov, M. Buzdalov, A. Buzdalova, A. Shalyto // Proceedings of 20th International Conference on Soft Computing MENDEL 2014. — Czech Republic, 2014. — P. 111–116.

117 *Zitzler E.*, *Laumanns M.*, *Thiele L.* SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization // Proceedings of the EUROGEN'2001 Conference. — 2001. — P. 95–100.

Ресурсы сети Интернет

118 Задача «Ships. Version 2» из архива задач Timus Online Judge [Электронный ресурс]. — URL: http://acm.timus.ru/problem.aspx?num=1394.

119 Google Code Jam: Problem Preparation Guide [Электронный ресурс]. — URL: https://code.google.com/codejam/problem-preparation.html.

120 *R Core Team.* R: A Language and Environment for Statistical Computing [Электронный ресурс] / R Foundation for Statistical Computing. — 2013. — URL: http://www.R-project.org/.

121 Timus Online Judge. Problem “Ships. Version 2” [Электронный ресурс]. — URL: http://acm.timus.ru/problem.aspx?num=1394.

ПРИЛОЖЕНИЕ А ПРОГРАММЫ, РЕШАЮЩИЕ ЗАДАЧУ «SHIPS. VERSION 2», СО ВСТАВЛЕННЫМИ СЧЕТЧИКАМИ

A.1 Программа 0937951

```
import java.io.*;
import java.util.*;

import ru.ifmo.ctd.ngp.demo.testgen.TimeoutChecker;
public class J0937951P {
    private static long counter$0 = 0;
    /* counter$1 ... counter$31 omitted for brevity */
    private static long counter$32 = 0;
    public static void profilerCleanup() {
        counter$0 = 0;
        /* counter$1 ... counter$31 omitted for brevity */
        counter$32 = 0;
    }
    public static java.util.Map<String, Long> profilerData() {
        java.util.Map<String, Long> rv = new java.util.HashMap<>();
        rv.put("counter$0", counter$0);
        /* counter$1 ... counter$31 omitted for brevity */
        rv.put("counter$32", counter$32);
        return rv;
    }
    static final int MAX_GREEDY = 1000;

    int n, m;
    int[] shipLen = new int[128];
    int[] shipCount = new int[128];
    int[] rowLen = new int[16];
    int[] rowLenCopy = new int[16];
    int[][] rowHasShip = new int[16][128];
    boolean[] isSum = new boolean[16384];
    int[] pred = new int[16384];
    int[] shipOnRow = new int[128];

    Random random = new Random(243632463);

    class ResultException extends RuntimeException {
        final int[][] answer;

        public ResultException(int[][] answer) {
            this.answer = answer;
        }
    }

    void findTwoRow() {
if ((++counter$0 & 262143) == 0) TimeoutChecker.check();
        int ship, sum, last;
```

```
        int searchLen = rowLen[m - 2];
        Arrays.fill(isSum, 1, searchLen + 1, false);
        isSum[0] = true;
        last = 0;
        for (ship = 0; ship < n; ++ship) {
if ((++counter$1 & 262143) == 0) TimeoutChecker.check();
            int curLen = shipLen[ship];
            for (int i = shipCount[ship]; i-- != 0; ) {
if ((++counter$2 & 262143) == 0) TimeoutChecker.check();
                sum = last;
                if (sum > searchLen - curLen) {
                    sum = searchLen - curLen;
                }
                for ( ; sum >= 0; --sum) {
if ((++counter$3 & 262143) == 0) TimeoutChecker.check();
                    if (isSum[sum] && !isSum[sum + curLen]) {
                        isSum[sum + curLen] = true;
                        pred[sum + curLen] = ship;
                    }
                }
                last += curLen;
            }
        }
        if (isSum[searchLen]) {
            for (ship = 0; ship < n; ++ship) {
if ((++counter$4 & 262143) == 0) TimeoutChecker.check();
                rowHasShip[m - 2][ship] = 0;
            }
            for (sum = searchLen; sum > 0; sum -= shipLen[pred[sum]]) {
if ((++counter$5 & 262143) == 0) TimeoutChecker.check();
                ship = pred[sum];
                rowHasShip[m - 2][ship]++;
                --shipCount[ship];
            }
            for (ship = 0; ship < n; ++ship) {
if ((++counter$6 & 262143) == 0) TimeoutChecker.check();
                rowHasShip[m - 1][ship] = shipCount[ship];
            }

            int[][] answer = new int[m][];

            for (int row = 0; row < m; ++row) {
if ((++counter$7 & 262143) == 0) TimeoutChecker.check();
                int realRow;
                for (realRow = 0; realRow < m; ++realRow) {
if ((++counter$8 & 262143) == 0) TimeoutChecker.check();
                    if (rowLenCopy[row] == rowLen[realRow]) {
                        break;
                    }
                }
                int count = 0;
```

```
                for (ship = 0; ship < n; ++ship) {
if ((++counter$9 & 262143) == 0) TimeoutChecker.check();
                    if (rowHasShip[realRow][ship] != 0) {
                        count += rowHasShip[realRow][ship];
                    }
                }
                int[] ans = new int[count];
                answer[row] = ans;
                int idx;
                for (ship = 0, idx = 0; ship < n; ++ship) {
if ((++counter$10 & 262143) == 0) TimeoutChecker.check();
                    for ( ; rowHasShip[realRow][ship] > 0; --rowHasShip[realRow][ship]) {
if ((++counter$11 & 262143) == 0) TimeoutChecker.check();
                        ans[idx++] = shipLen[ship];
                    }
                }
                rowLen[realRow] = 0;
            }
            throw new ResultException(answer);
        }
    }

    void find(int row, int ship, int freeLen) {
if ((++counter$12 & 262143) == 0) TimeoutChecker.check();
        if (row == m - 2) {
            findTwoRow();
        } else {
            for ( ; ship < n; ++ship) {
if ((++counter$13 & 262143) == 0) TimeoutChecker.check();
                if (shipCount[ship] == 0 || shipLen[ship] > freeLen) {
                    continue;
                }
                int j = freeLen / shipLen[ship];
                if (j > shipCount[ship]) {
                    j = shipCount[ship];
                }
                shipCount[ship] -= j;
                for ( ; j > 0; --j) {
if ((++counter$14 & 262143) == 0) TimeoutChecker.check();
                    rowHasShip[row][ship] = j;
                    int newFreeLen = freeLen - shipLen[ship] * j;
                    if (newFreeLen == 0) {
                        find(row + 1, 0, rowLen[row + 1]);
                    } else {
                        find(row, ship + 1, newFreeLen);
                    }
                    ++shipCount[ship];
                }
                rowHasShip[row][ship] = 0;
            }
        }
```

```
    }

    boolean findGreedy() {
if ((++counter$15 & 262143) == 0) TimeoutChecker.check();
        for (int ship = 0; ship < n; ++ship) {
if ((++counter$16 & 262143) == 0) TimeoutChecker.check();
            shipOnRow[ship] = -1;
        }
        for (int row = 0; row < m; ++row) {
if ((++counter$17 & 262143) == 0) TimeoutChecker.check();
            int curLen = rowLen[row];
            isSum[0] = true;
            for (int k = curLen; k > 0; --k) {
if ((++counter$18 & 262143) == 0) TimeoutChecker.check();
                isSum[k] = false;
            }
            for (int ship = 0; ship < n && !isSum[curLen]; ++ship) {
if ((++counter$19 & 262143) == 0) TimeoutChecker.check();
                if (shipOnRow[ship] < 0) {
                    for (int k = curLen - shipLen[ship]; k >= 0; --k) {
if ((++counter$20 & 262143) == 0) TimeoutChecker.check();
                        if (isSum[k]) {
                            int newK = k + shipLen[ship];
                            if (!isSum[newK]) {
                                isSum[newK] = true;
                                pred[newK] = ship;
                            }
                        }
                    }
                }
            }
            if (!isSum[curLen]) {
                return false;
            }
            for (int k = curLen; k > 0; k -= shipLen[pred[k]]) {
if ((++counter$21 & 262143) == 0) TimeoutChecker.check();
                shipOnRow[pred[k]] = row;
            }
        }
        return true;
    }

    void randGreedy() {
if ((++counter$22 & 262143) == 0) TimeoutChecker.check();
        for (int i = 0; i < n; ++i) {
if ((++counter$23 & 262143) == 0) TimeoutChecker.check();
            int j = random.nextInt(n);
            int t = shipLen[j];
            shipLen[j] = shipLen[i];
            shipLen[i] = t;
        }
```

```
    }

    public void solve(List<Integer> ships, List<Integer> havens) {
if ((++counter$24 & 262143) == 0) TimeoutChecker.check();
        try {
            n = ships.size();
            m = havens.size();
            for (int i = 0; i < n; ++i) {
if ((++counter$25 & 262143) == 0) TimeoutChecker.check();
                shipLen[i] = ships.get(i);
            }
            for (int i = 0; i < m; ++i) {
if ((++counter$26 & 262143) == 0) TimeoutChecker.check();
                rowLen[i] = havens.get(i);
            }
            for (int i = 0; i < MAX_GREEDY; ++i) {
if ((++counter$27 & 262143) == 0) TimeoutChecker.check();
                if (findGreedy()) {
                    int[][] answer = new int[m][];
                    for (int t = 0; t < m; ++t) {
if ((++counter$28 & 262143) == 0) TimeoutChecker.check();
                        int cnt = 0;
                        for (int j = 0; j < n; ++j) {
if ((++counter$29 & 262143) == 0) TimeoutChecker.check();
                            if (shipOnRow[j] == t) {
                                ++cnt;
                            }
                        }
                        int[] ans = new int[cnt];
                        answer[t] = ans;
                        for (int j = 0, idx = 0; j < n; ++j) {
if ((++counter$30 & 262143) == 0) TimeoutChecker.check();
                            if (shipOnRow[j] == t) {
                                ans[idx++] = shipLen[j];
                            }
                        }
                    }
                    throw new ResultException(answer);
                } else {
                    randGreedy();
                }
            }
            Arrays.sort(shipLen, 0, n);
            for (int i = 0, j = n - 1; i < j; ++i, --j) {
if ((++counter$31 & 262143) == 0) TimeoutChecker.check();
                int tmp = shipLen[i];
                shipLen[i] = shipLen[j];
                shipLen[j] = tmp;
            }
            System.arraycopy(rowLen, 0, rowLenCopy, 0, m);
            Arrays.sort(rowLen, 0, m);
```

```
            shipCount[0] = 1;
            int j = 0;
            for (int i = 1; i < n; ++i) {
if ((++counter$32 & 262143) == 0) TimeoutChecker.check();
                if (shipLen[i] == shipLen[j]) {
                    shipCount[j]++;
                } else {
                    shipLen[++j] = shipLen[i];
                    shipCount[j] = 1;
                }
            }
            n = j + 1;
            find(0, 0, rowLen[0]);
            throw new AssertionError();
        } catch (ResultException ex) {}
    }
}
```

A.2 Программа 1700736

```
import java.io.*;
import java.util.*;

import ru.ifmo.ctd.ngp.demo.testgen.TimeoutChecker;
public class J1700736P {
    private static long counter$0 = 0;
    /* counter$1 ... counter$31 omitted */
    private static long counter$32 = 0;
    public static void profilerCleanup() {
        counter$0 = 0;
        /* counter$1 ... counter$31 omitted */
        counter$32 = 0;
    }
    public static java.util.Map<String, Long> profilerData() {
        java.util.Map<String, Long> rv = new java.util.HashMap<>();
        rv.put("counter$0", counter$0);
        /* counter$1 ... counter$31 omitted */
        rv.put("counter$32", counter$32);
        return rv;
    }
    private final long mySeed;
    public J1700736P(long seed) {
        mySeed = seed;
    }
    public J1700736P() {
        this(1048);
    }

    private static final class Impl {
        public int[][] answer;
```

```
        private static final int maxn = 105;
        private static final int maxsize = maxn * maxn;

        private final int n, m;
        private int size;
        private final int[] A = new int[maxn];
        private final int[] K = new int[maxn];
        private final int[] L = new int[maxn];
        private final int[] R = new int[maxn];
        private final int[] P = new int[maxn];
        private final int[] ID = new int[maxn];
        private final int[] Q = new int[maxsize];
        private final int[] f = new int[maxsize];
        private final boolean[] used = new boolean[maxn];

        private final Random random;

        private void randomize(int[] a, int n) {
if ((++counter$10 & 262143) == 0) TimeoutChecker.check();
if (true) {
            for (int i = 1; i <= n; ++i) {
if ((++counter$11 & 262143) == 0) TimeoutChecker.check();
                int p = i + random.nextInt(n - i + 1);
                int tmp = a[i];
                a[i] = a[p];
                a[p] = tmp;
            }
}
        }

        public Impl(List<Integer> ships, List<Integer> havens, long seed) {
            random = new Random(seed);
            n = ships.size();
            m = havens.size();
            for (int i = 1; i <= n; ++i) {
if ((++counter$13 & 262143) == 0) TimeoutChecker.check();
                A[i] = ships.get(i - 1);
                P[i] = i;
            }
            for (int i = 1; i <= m; ++i) {
if ((++counter$14 & 262143) == 0) TimeoutChecker.check();
                L[i] = havens.get(i - 1);
                ID[i] = i;
            }
            for (int step = 1; ; ++step) {
if ((++counter$15 & 262143) == 0) TimeoutChecker.check();
                for (int i = 1; i <= m; ++i) {
if ((++counter$16 & 262143) == 0) TimeoutChecker.check();
                    for (int j = i + 1; j <= m; ++j) {
if ((++counter$17 & 262143) == 0) TimeoutChecker.check();
                        if (L[ID[j]] < L[ID[i]]) {
```

```
                            int tmp = ID[i];
                            ID[i] = ID[j];
                            ID[j] = tmp;
                        }
                    }
                }
                if (step == 1) {
                    for (int i = 1; i <= n; ++i) {
if ((++counter$18 & 262143) == 0) TimeoutChecker.check();
                        for (int j = i + 1; j <= n; ++j) {
if ((++counter$19 & 262143) == 0) TimeoutChecker.check();
                            if (A[P[j]] < A[P[i]]) {
                                int tmp = P[i];
                                P[i] = P[j];
                                P[j] = tmp;
                            }
                        }
                    }
                }
                Arrays.fill(used, false);
                int Tv = 0;
                for (Tv = 1; Tv <= m; ++Tv) {
if ((++counter$20 & 262143) == 0) TimeoutChecker.check();
                    int v = ID[Tv];
                    if (Tv == m) {
                        for (int i = 1; i <= n; i++) {
if ((++counter$21 & 262143) == 0) TimeoutChecker.check();
                            if (!used[i]) {
                                R[i] = v;
                            }
                        }
                        continue;
                    }
                    int L0 = L[v];
                    if (step > 1) {
                        randomize(P, n);
                    }
                    f[0] = 0;
                    for (int i = 1; i <= L0; ++i) {
if ((++counter$22 & 262143) == 0) TimeoutChecker.check();
                        f[i] = -1;
                    }
                    Q[size = 1] = 0;
                    for (int i = 1; i <= n; i++) {
if ((++counter$23 & 262143) == 0) TimeoutChecker.check();
                        if (!used[P[i]] && f[L0] == -1) {
                            if (size <= (L0 >> 2)) {
                                for (int _size = size, cl = 1; cl <= _size; cl++) {
if ((++counter$24 & 262143) == 0) TimeoutChecker.check();
                                    int t = Q[cl] + A[P[i]];
                                    if (t <= L0 && f[t] == -1) {
```

```
                                    f[t] = i;
                                    Q[++size] = t;
                                }
                            }
                        } else {
                            int now = A[P[i]];
                            for (int k = L0; k >= now; k--) {
if ((++counter$25 & 262143) == 0) TimeoutChecker.check();
                                if (f[k] == -1 && f[k - now] != -1) {
                                    f[k] = i;
                                }
                            }
                        }
                    }
                }
                if (f[L0] == -1) break;
                for (int k = L0; k > 0; k -= A[P[f[k]]]) {
if ((++counter$26 & 262143) == 0) TimeoutChecker.check();
                    R[P[f[k]]] = v;
                    used[P[f[k]]] = true;
                }
            }
            if (Tv<=m) continue;

            answer = new int[m][];
            for (int i = 1; i <=m; i++) {
if ((++counter$28 & 262143) == 0) TimeoutChecker.check();
                int C = 0;
                for (int k = 1; k <= n; k++) {
if ((++counter$29 & 262143) == 0) TimeoutChecker.check();
                    C += (R[k] == i) ? 1 : 0;
                }
                answer[i - 1] = new int[C];
                for (int k = 1, t = 0; k <= n; k++) {
if ((++counter$31 & 262143) == 0) TimeoutChecker.check();
                    if (R[k] == i) {
                        answer[i - 1][t++] = A[k];
                    }
                }
            }
            break;
        }
    }
}

    public void solve(List<Integer> ships, List<Integer> havens) {
if ((++counter$32 & 262143) == 0) TimeoutChecker.check();
if (true) {
    new Impl(ships, havens, mySeed);
}
    }
```

```
}
```

A.3 Программа 2208365

```
import java.util.*;

import ru.ifmo.ctd.ngp.demo.testgen.TimeoutChecker;
public class J2208365P {
    private static long counter$0 = 0;
    /* counter$1 ... counter$14 omitted for brevity */
    private static long counter$15 = 0;
    public static void profilerCleanup() {
        counter$0 = 0;
        /* counter$1 ... counter$14 omitted for brevity */
        counter$15 = 0;
    }
    public static java.util.Map<String, Long> profilerData() {
        java.util.Map<String, Long> rv = new java.util.HashMap<>();
        rv.put("counter$0", counter$0);
        /* counter$1 ... counter$14 omitted for brevity */
        rv.put("counter$15", counter$15);
        return rv;
    }
    public void solve(List<Integer> ships, List<Integer> havens) {
if ((++counter$0 & 262143) == 0) TimeoutChecker.check();
        new Implementation(ships, havens);
    }

    private static class Implementation {
        private int n;
        private int m;
        private static final int maxM = 12;
        private static final int maxN = 105;
        private static final int maxS = 10000;
        private final boolean[][] f = new boolean[maxN + 1][maxS + 1];
        private final int[] left = new int[maxN + 1];
        private final int[] row = new int[maxM + 1];
        private final int[] idrow = new int[maxM + 1];
        private final int[] ship = new int[maxN + 1];
        private final int[] id = new int[maxN + 1];
        private final int[] belong = new int[maxN + 1];

        public Implementation(List<Integer> items, List<Integer> sacks) {
            n = items.size();
            m = sacks.size();
            for (int i = 0; i < n; ++i) {
if ((++counter$1 & 262143) == 0) TimeoutChecker.check();
                ship[i + 1] = items.get(i);
            }
            for (int i = 1; i <= m; ++i) {
if ((++counter$2 & 262143) == 0) TimeoutChecker.check();
```

```
                row[i] = sacks.get(i - 1);
                idrow[i] = i;
            }
            for (int i = 1; i <= m; ++i) {
if ((++counter$3 & 262143) == 0) TimeoutChecker.check();
                for (int j = i + 1; j <= m; ++j) {
if ((++counter$4 & 262143) == 0) TimeoutChecker.check();
                    if (row[i] < row[j]) {
                        int tmp;
                        tmp = row[i]; row[i] = row[j]; row[j] = tmp;
                        tmp = idrow[i]; idrow[i] = idrow[j]; idrow[j] = tmp;
                    }
                }
            }
            for (int i = 1; i <= n; ++i) {
if ((++counter$5 & 262143) == 0) TimeoutChecker.check();
                id[i] = i;
            }

            Random random = new Random(239);
            f[0][0] = true;
            while (true) {
if ((++counter$6 & 262143) == 0) TimeoutChecker.check();
                for (int i = 1; i <= n; ++i) {
if ((++counter$7 & 262143) == 0) TimeoutChecker.check();
                    int j = random.nextInt(n) + 1;
                    int k = random.nextInt(n) + 1;
                    int tmp = id[j]; id[j] = id[k]; id[k] = tmp;
                }
                if (dfs(m)) {
                    return;
                }
            }
        }

        boolean dfs(int x) {
if ((++counter$8 & 262143) == 0) TimeoutChecker.check();
            if (x == 0) {
                return true;
            }
            int tot = 0;
            int k = 0;
            for (int i = 1; i <= n; ++i) {
if ((++counter$9 & 262143) == 0) TimeoutChecker.check();
                if (belong[id[i]] == 0) {
                    ++tot;
                    left[tot] = id[i];
                    k += ship[id[i]];
                }
            }
            f[0][0] = true;
```

```
            for (int i = 1; i <= tot; ++i) {
if ((++counter$10 & 262143) == 0) TimeoutChecker.check();
                Arrays.fill(f[i], 0, k + 1, false);
            }
            k = 0;
            for (int i = 1; i <= tot; ++i) {
if ((++counter$11 & 262143) == 0) TimeoutChecker.check();
                for (int j = 0; j <= k; ++j) {
if ((++counter$12 & 262143) == 0) TimeoutChecker.check();
                    if (f[i - 1][j]) {
                        f[i][j] = true;
                        f[i][j + ship[left[i]]] = true;
                    }
                }
                k += ship[left[i]];
            }
            k = 0;
            for (int i = 1; i <= x; ++i) {
if ((++counter$13 & 262143) == 0) TimeoutChecker.check();
                if (!f[tot][row[i]]) {
                    return false;
                }
                if (!f[tot - 1][row[i]]) {
                    ++k;
                }
            }
            if (k > 1) {
                return false;
            }
            k = row[x];
            for (int i = tot; i >= 1; --i) {
if ((++counter$14 & 262143) == 0) TimeoutChecker.check();
                if (!f[i - 1][k]) {
                    belong[left[i]] = x;
                    k -= ship[left[i]];
                }
            }

            if (dfs(x - 1)) {
                return true;
            }

            for (int i = 1; i <= n; ++i) {
if ((++counter$15 & 262143) == 0) TimeoutChecker.check();
                if (belong[i] == x) {
                    belong[i] = 0;
                }
            }
            return false;
        }
    }
```

```
}
```

A.4 Программа 2558302

```
import java.util.*;

import ru.ifmo.ctd.ngp.demo.testgen.TimeoutChecker;
public class J2558302P {
    private static long counter$0 = 0;
    /* counter$1 ... counter$15 omitted for brevity */
    private static long counter$16 = 0;
    public static void profilerCleanup() {
        counter$0 = 0;
        /* counter$1 ... counter$15 omitted for brevity */
        counter$16 = 0;
    }
    public static java.util.Map<String, Long> profilerData() {
        java.util.Map<String, Long> rv = new java.util.HashMap<>();
        rv.put("counter$0", counter$0);
        /* counter$1 ... counter$15 omitted for brevity */
        rv.put("counter$16", counter$16);
        return rv;
    }
    public void solve(List<Integer> ships, List<Integer> havens) {
if ((++counter$0 & 262143) == 0) TimeoutChecker.check();
        new Implementation(ships, havens);
    }

    private static class Implementation {
        int n;
        int m;
        int[] a = new int[102];
        int[] c = new int[12];
        int[] order = new int[102];
        int[] path = new int[10002];
        int[][] ans = new int[12][102];
        int[] tot = new int[12];
        int[] p = new int[12];
        boolean[] used = new boolean[102];
        boolean[] can = new boolean[10002];

        private void dp(int m) {
if ((++counter$1 & 262143) == 0) TimeoutChecker.check();
            Arrays.fill(can, false);
            can[0] = true;
            for (int i = 1; i <= n; i++) {
if ((++counter$2 & 262143) == 0) TimeoutChecker.check();
                if (!used[order[i]]) {
                    int j = order[i];
                    for (int k = m; k >= a[j]; k--) {
if ((++counter$3 & 262143) == 0) TimeoutChecker.check();
```

```
                        if (!can[k] && can[k - a[j]]) {
                            can[k] = true;
                            path[k] = j;
                        }
                    }
                }
            }
        }

        private boolean search(int k) {
if ((++counter$4 & 262143) == 0) TimeoutChecker.check();
            if (k > m) {
                return true;
            }
            int s = 0;
            for (int i = k; i <= m; i++) {
if ((++counter$5 & 262143) == 0) TimeoutChecker.check();
                s = Math.max(s, c[i]);
            }
            dp(s);
            for (int i = k; i <= m; i++) {
if ((++counter$6 & 262143) == 0) TimeoutChecker.check();
                if (!can[c[i]]) {
                    return false;
                }
            }
            int now = p[k];
            tot[now] = 0;
            for (int i = c[k]; i != 0; i -= a[path[i]]) {
if ((++counter$7 & 262143) == 0) TimeoutChecker.check();
                ans[now][tot[now]++] = path[i];
            }
            for (int i = 0; i < tot[now]; i++) {
if ((++counter$8 & 262143) == 0) TimeoutChecker.check();
                used[ans[now][i]] = true;
            }
            if (search(k + 1)) {
                return true;
            }
            for (int i = 0; i < tot[now]; i++) {
if ((++counter$9 & 262143) == 0) TimeoutChecker.check();
                used[ans[now][i]] = false;
            }
            return false;
        }

        protected Implementation(List<Integer> items, List<Integer> sacks) {
            Random r = new Random(872345623);
            n = items.size();
            m = sacks.size();
            for (int i = 0; i < n; ++i) {
```

```
if ((++counter$10 & 262143) == 0) TimeoutChecker.check();
                a[i + 1] = items.get(i);
            }
            for (int i = 0; i < m; ++i) {
if ((++counter$11 & 262143) == 0) TimeoutChecker.check();
                c[i + 1] = sacks.get(i);
            }
            for (int i = 1; i <= n; i++) {
if ((++counter$12 & 262143) == 0) TimeoutChecker.check();
                order[i] = i;
            }
            for (int i = 1; i <= m; i++) {
if ((++counter$13 & 262143) == 0) TimeoutChecker.check();
                p[i] = i;
            }
            do {
if ((++counter$14 & 262143) == 0) TimeoutChecker.check();
                for (int i = 1; i <= 10000; i++) {
if ((++counter$15 & 262143) == 0) TimeoutChecker.check();
                    int x = r.nextInt(n) + 1;
                    int y = r.nextInt(n) + 1;
                    int t = order[x];
                    order[x] = order[y];
                    order[y] = t;
                }
                for (int i=1;i<=m;i++) {
if ((++counter$16 & 262143) == 0) TimeoutChecker.check();
                    int x = r.nextInt(m) + 1;
                    int y = r.nextInt(m) + 1;
                    int t;
                    t = c[x]; c[x] = c[y]; c[y] = t;
                    t = p[x]; p[x] = p[y]; p[y] = t;
                }
            } while (!search(1));
        }
    }
}
```

A.5 Программа 3589819

```
import java.io.*;
import java.util.*;

import ru.ifmo.ctd.ngp.demo.testgen.TimeoutChecker;
public class J3589819P {
    private static long counter$0 = 0;
    /* counter$1 ... counter$27 omitted for brevity */
    private static long counter$28 = 0;
    public static void profilerCleanup() {
        counter$0 = 0;
        /* counter$1 ... counter$27 omitted for brevity */
```

```
        counter$28 = 0;
    }
    public static java.util.Map<String, Long> profilerData() {
        java.util.Map<String, Long> rv = new java.util.HashMap<>();
        rv.put("counter$0", counter$0);
        /* counter$1 ... counter$27 omitted for brevity */
        rv.put("counter$28", counter$28);
        return rv;
    }
    static void swap(int[] array, int i1, int i2) {
if ((++counter$0 & 262143) == 0) TimeoutChecker.check();
        int tmp = array[i1];
        array[i1] = array[i2];
        array[i2] = tmp;
    }
    static class RNG {
        int x = 1987657173, y = 712356789, z = 531288629, w = 138751267;

        int next() {
if ((++counter$1 & 262143) == 0) TimeoutChecker.check();
            int t = x ^ x << 11;
            x = y;
            y = z;
            z = w;
            w = ((t ^ t >>> 8) ^ w) ^ w >>> 19;
            return w;
        }

        void shuffle(int[] array, int from, int to) {
if ((++counter$2 & 262143) == 0) TimeoutChecker.check();
            for (int i = from; i < to; ++i) {
if ((++counter$3 & 262143) == 0) TimeoutChecker.check();
                int idx = from + (int) ((next() & ((1L << 32) - 1)) % (i - from + 1));
                swap(array, i, idx);
            }
        }
    }

    final static int N = 99;
    final static int M = 9;
    final static int S = N * 100;

    class Bitset {
        int[] d = new int[(S >> 5) + 1];
        int n = 0;

        void set(int i) {
if ((++counter$4 & 262143) == 0) TimeoutChecker.check();
            int m = i >> 5;
            for (; n <= m; d[n++] = 0) {
if ((++counter$5 & 262143) == 0) TimeoutChecker.check();
```

```
            }
            d[m] |= 1 << i;
        }
        void assignFrom(Bitset b) {
if ((++counter$6 & 262143) == 0) TimeoutChecker.check();
            this.n = b.n;
            System.arraycopy(b.d, 0, d, 0, d.length);
        }
        boolean get(int i) {
if ((++counter$7 & 262143) == 0) TimeoutChecker.check();
            return (i >> 5) < n && (d[i >> 5] & 1 << i) != 0;
        }
        void shiftOr(Bitset b, int sh, int m) {
if ((++counter$8 & 262143) == 0) TimeoutChecker.check();
            for (m >>= 5; n <= m; d[n++] = 0) {
if ((++counter$9 & 262143) == 0) TimeoutChecker.check();
            }
            int id_s = sh >> 5;
            int id_e = n;
            if (id_s < id_e) {
                int[] i = b.d;
                int ii_i = 0;
                if ((sh &= 31) != 0) {
                    int rh = 32 - sh;
                    id_e = Math.min(id_e, id_s + b.n);
                    for (; ((id_e - id_s) & 3) != 0; ++id_s, ++ii_i) {
if ((++counter$10 & 262143) == 0) TimeoutChecker.check();
                        d[id_s] |= i[ii_i] << sh;
                        d[id_s + 1] |= i[ii_i] >>> rh;
                    }
                    for (; id_e != id_s; ) {
if ((++counter$11 & 262143) == 0) TimeoutChecker.check();
                        d[id_s++] |= i[ii_i] << sh;
                        d[id_s] |= i[ii_i++] >>> rh;
                        d[id_s++] |= i[ii_i] << sh;
                        d[id_s] |= i[ii_i++] >>> rh;
                        d[id_s++] |= i[ii_i] << sh;
                        d[id_s] |= i[ii_i++] >>> rh;
                        d[id_s++] |= i[ii_i] << sh;
                        d[id_s] |= i[ii_i++] >>> rh;
                    }
                } else {
                    for (; ((id_e - id_s) & 3) != 0; ) {
if ((++counter$12 & 262143) == 0) TimeoutChecker.check();
                        d[id_s++] |= i[ii_i++];
                    }
                    for (; id_e != id_s; ) {
if ((++counter$13 & 262143) == 0) TimeoutChecker.check();
                        d[id_s++] |= i[ii_i++];
                        d[id_s++] |= i[ii_i++];
                        d[id_s++] |= i[ii_i++];
```

```
                        d[id_s++] |= i[ii_i++];
                    }
                }
            }
        }
    }

    Bitset[] f = new Bitset[N];
    int[] a = new int[N];
    int[] row = new int[M];
    int[] p = new int[N];
    int[] q = new int[M];
    int[] v = new int[N];
    int ve;
    int n, m;
    int[] first = new int[M];
    int[] next = new int[N];

    boolean fit(int k) {
if ((++counter$14 & 262143) == 0) TimeoutChecker.check();
        int h = row[k];
        for (int i = 0, s = 0; i != ve; ++i) {
if ((++counter$15 & 262143) == 0) TimeoutChecker.check();
            Bitset F = f[i + 1];
            F.assignFrom(f[i]);
            s = Math.min(s + a[v[i]], h);
            F.shiftOr(f[i], a[v[i]], s);
            if (F.get(h)) {
                for (int j = i; h != 0; --j) {
if ((++counter$16 & 262143) == 0) TimeoutChecker.check();
                    if (!f[j].get(h)) {
                        p[v[j]] = k;
                        h -= a[v[j]];
                    }
                }
                for (; i >= 0; --i) {
if ((++counter$17 & 262143) == 0) TimeoutChecker.check();
                    if ((~p[v[i]]) != 0) {
                        next[v[i]] = first[k];
                        first[k] = v[i];
                        v[i] = v[--ve];
                    }
                }
                return true;
            }
        }
        return false;
    }

    void cancel(int k) {
if ((++counter$18 & 262143) == 0) TimeoutChecker.check();
```

```
        for (int i = first[k]; (~i) != 0; i = next[i]) {
if ((++counter$19 & 262143) == 0) TimeoutChecker.check();
            p[v[ve++] = i] = -1;
        }
        first[k] = -1;
    }

    boolean solve() {
if ((++counter$20 & 262143) == 0) TimeoutChecker.check();
        if (m == 1) {
            return true;
        }
        --m;
        for (int i = 0; i < m; ++i) {
if ((++counter$21 & 262143) == 0) TimeoutChecker.check();
            if (fit(q[i])) {
                for (int j = i; j < m; ++j) {
if ((++counter$22 & 262143) == 0) TimeoutChecker.check();
                    swap(q, j, j + 1);
                }
                if (solve()) {
                    ++m;
                    return true;
                }
                for (int j = m; j > i; --j) {
if ((++counter$23 & 262143) == 0) TimeoutChecker.check();
                    swap(q, j - 1, j);
                }
                cancel(q[i]);
            }
        }
        ++m;
        return false;
    }

    public void solve(List<Integer> ships, List<Integer> havens) {
if ((++counter$24 & 262143) == 0) TimeoutChecker.check();
        RNG random = new RNG();
        for (int i = 0; i < N; ++i) {
if ((++counter$25 & 262143) == 0) TimeoutChecker.check();
            f[i] = new Bitset();
        }
        f[0].set(0);
        n = ships.size();
        m = havens.size();
        Arrays.fill(first, -1);
        for (int i = 0; i < n; ++i) {
if ((++counter$26 & 262143) == 0) TimeoutChecker.check();
            a[i] = ships.get(i);
            p[i] = -1;
            v[ve++] = i;
```

```
        }
        for (int i = 0; i < m; ++i) {
if ((++counter$27 & 262143) == 0) TimeoutChecker.check();
            row[i] = havens.get(i);
            q[i] = i;
        }
        while (true) {
if ((++counter$28 & 262143) == 0) TimeoutChecker.check();
            random.shuffle(v, 0, n);
            random.shuffle(q, 0, m);
            if (solve()) {
                break;
            }
        }
        fit(q[0]);
    }
}
```

A.6 Программа 3600147

```
import java.io.*;
import java.util.*;

import ru.ifmo.ctd.ngp.demo.testgen.TimeoutChecker;
public class J3600147P {
    private static long counter$0 = 0;
    /* counter$1 ... counter$30 omitted for brevity */
    private static long counter$31 = 0;
    public static void profilerCleanup() {
        counter$0 = 0;
        /* counter$1 ... counter$30 omitted for brevity */
        counter$31 = 0;
    }
    public static java.util.Map<String, Long> profilerData() {
        java.util.Map<String, Long> rv = new java.util.HashMap<>();
        rv.put("counter$0", counter$0);
        /* counter$1 ... counter$30 omitted for brevity */
        rv.put("counter$31", counter$31);
        return rv;
    }
    static void swap(int[] array, int i1, int i2) {
if ((++counter$0 & 262143) == 0) TimeoutChecker.check();
        int tmp = array[i1];
        array[i1] = array[i2];
        array[i2] = tmp;
    }
    static class RNG {
        int x = 1987657173, y = 712356789, z = 531288629, w = 138751267;

        int next() {
if ((++counter$1 & 262143) == 0) TimeoutChecker.check();
```

```
            int t = x ^ x << 11;
            x = y;
            y = z;
            z = w;
            w = ((t ^ t >>> 8) ^ w) ^ w >>> 19;
            return w;
        }

        void shuffle(int[] array, int from, int to) {
if ((++counter$2 & 262143) == 0) TimeoutChecker.check();
            for (int i = from; i < to; ++i) {
if ((++counter$3 & 262143) == 0) TimeoutChecker.check();
                int idx = from + (int) ((next() & ((1L << 32) - 1)) % (i - from + 1));
                swap(array, i, idx);
            }
        }
    }

    final static int N = 99;
    final static int M = 9;
    final static int S = N * 100;
    final static int L = 2000;

    static class Bitset {
        int[] d = new int[(S >> 5) + 1];
        int n = 0;

        void set(int i) {
if ((++counter$4 & 262143) == 0) TimeoutChecker.check();
            int m = i >> 5;
            for (; n <= m; d[n++] = 0) {
if ((++counter$5 & 262143) == 0) TimeoutChecker.check();
            }
            d[m] |= 1 << i;
        }
        void assignFrom(Bitset b) {
if ((++counter$6 & 262143) == 0) TimeoutChecker.check();
            this.n = b.n;
            System.arraycopy(b.d, 0, d, 0, d.length);
        }
        boolean get(int i) {
if ((++counter$7 & 262143) == 0) TimeoutChecker.check();
            return (i >> 5) < n && (d[i >> 5] & 1 << i) != 0;
        }
        void shiftOr(Bitset b, int sh, int m) {
if ((++counter$8 & 262143) == 0) TimeoutChecker.check();
            for (m >>= 5; n <= m; d[n++] = 0) {
if ((++counter$9 & 262143) == 0) TimeoutChecker.check();
            }
            int id_s = sh >> 5;
            int id_e = n;
```

```
            if (id_s < id_e) {
                int[] i = b.d;
                int ii_i = 0;
                if ((sh &= 31) != 0) {
                    int rh = 32 - sh;
                    id_e = Math.min(id_e, id_s + b.n);
                    for (; ((id_e - id_s) & 3) != 0; ++id_s, ++ii_i) {
if ((++counter$10 & 262143) == 0) TimeoutChecker.check();
                        d[id_s] |= i[ii_i] << sh;
                        d[id_s + 1] |= i[ii_i] >>> rh;
                    }
                    for (; id_e != id_s; ) {
if ((++counter$11 & 262143) == 0) TimeoutChecker.check();
                        d[id_s++] |= i[ii_i] << sh;
                        d[id_s] |= i[ii_i++] >>> rh;
                        d[id_s++] |= i[ii_i] << sh;
                        d[id_s] |= i[ii_i++] >>> rh;
                        d[id_s++] |= i[ii_i] << sh;
                        d[id_s] |= i[ii_i++] >>> rh;
                        d[id_s++] |= i[ii_i] << sh;
                        d[id_s] |= i[ii_i++] >>> rh;
                    }
                } else {
                    for (; ((id_e - id_s) & 3) != 0; ) {
if ((++counter$12 & 262143) == 0) TimeoutChecker.check();
                        d[id_s++] |= i[ii_i++];
                    }
                    for (; id_e != id_s; ) {
if ((++counter$13 & 262143) == 0) TimeoutChecker.check();
                        d[id_s++] |= i[ii_i++];
                        d[id_s++] |= i[ii_i++];
                        d[id_s++] |= i[ii_i++];
                        d[id_s++] |= i[ii_i++];
                    }
                }
            }
        }
    }

    Bitset[] f = new Bitset[N];
    int[] a = new int[N];
    int[] row = new int[M];
    int[] p = new int[N];
    int[] q = new int[M];
    int[] v = new int[101];
    int ve;
    int n, m;
    int[] first = new int[M];
    int[] next = new int[N];
    int[] u = new int[M];
    int same, same_cnt;
```

```
    int[] hs = new int[N];
    int hsh;
    TreeSet<Integer> U = new TreeSet<>();

    boolean fit(int k) {
if ((++counter$14 & 262143) == 0) TimeoutChecker.check();
        int h = row[k];
        if (same * same_cnt >= h && h % same == 0) {
            u[k] = h / same;
            same_cnt -= u[k];
            return true;
        }
        int mk = Math.min(h / same, same_cnt);
        for (int i = 0, s = 0; i != ve; ++i) {
if ((++counter$15 & 262143) == 0) TimeoutChecker.check();
            Bitset F = f[i + 1];
            F.assignFrom(f[i]);
            s = Math.min(s + a[v[i]], h);
            F.shiftOr(f[i], a[v[i]], s);
            for (int l = mk; l >= 0; --l) {
if ((++counter$16 & 262143) == 0) TimeoutChecker.check();
                if (h - l * same >= 0 && F.get(h - l * same)) {
                    h -= l * same;
                    for (int j = i; h != 0; --j) {
if ((++counter$17 & 262143) == 0) TimeoutChecker.check();
                        if (!f[j].get(h)) {
                            p[v[j]] = k;
                            h -= a[v[j]];
                        }
                    }
                    for (; i >= 0; --i) {
if ((++counter$18 & 262143) == 0) TimeoutChecker.check();
                        if ((~p[v[i]]) != 0) {
                            hsh ^= hs[v[i]];
                            next[v[i]] = first[k];
                            first[k] = v[i];
                            v[i] = v[--ve];
                        }
                    }
                    u[k] = l;
                    same_cnt -= l;
                    return true;
                }
            }
        }
        return false;
    }

    void reverse(int[] a, int from, int to) {
if ((++counter$19 & 262143) == 0) TimeoutChecker.check();
        --to;
```

```
        int tmp;
        while (from < to) {
if ((++counter$20 & 262143) == 0) TimeoutChecker.check();
            swap(a, from++, to--);
        }
    }

    void cancel(int k) {
if ((++counter$21 & 262143) == 0) TimeoutChecker.check();
        int s = ve;
        for (int i = first[k]; (~i) != 0; i = next[i]) {
if ((++counter$22 & 262143) == 0) TimeoutChecker.check();
            p[v[ve++] = i] = -1;
            hsh ^= hs[i];
        }
        first[k] = -1;
        same_cnt += u[k];
        u[k] = 0;
    }

    boolean solve(int k) {
if ((++counter$23 & 262143) == 0) TimeoutChecker.check();
        if (k == 1) {
            int i = row[q[1]] > row[q[0]] ? 1 : 0;
            if (fit(q[i])) {
                swap(q, i, k);
                return true;
            }
            return false;
        }
        if (hsh != 0 && k < 4) {
            if (U.contains(hsh)) {
                return false;
            }
            U.add(hsh);
        }
        for (int i = 0; i < k; ++i) {
if ((++counter$24 & 262143) == 0) TimeoutChecker.check();
            if (fit(q[i])) {
                if ((k & 1) != 0) {
                    reverse(q, i, k + 1);
                    if (solve(k - 1)) {
                        return true;
                    }
                    reverse(q, i, k + 1);
                } else {
                    swap(q, i, k);
                    if (solve(k - 1)) {
                        return true;
                    }
                    swap(q, i, k);
```

```
                }
                cancel(q[i]);
            } else {
                break;
            }
        }
        return false;
    }

    public void solve(List<Integer> ships, List<Integer> havens) {
if ((++counter$25 & 262143) == 0) TimeoutChecker.check();
        RNG random = new RNG();
        for (int i = 0; i < N; ++i) {
if ((++counter$26 & 262143) == 0) TimeoutChecker.check();
            f[i] = new Bitset();
        }
        f[0].set(0);
        n = ships.size();
        m = havens.size();
        Arrays.fill(first, -1);
        for (int i = 0; i < n; ++i) {
if ((++counter$27 & 262143) == 0) TimeoutChecker.check();
            hs[i] = random.next();
            a[i] = ships.get(i);
            p[i] = -1;
            v[a[i]]++;
        }
        for (int i = 0; i < m; ++i) {
if ((++counter$28 & 262143) == 0) TimeoutChecker.check();
            row[i] = havens.get(i);
            q[i] = i;
        }
        same = 0;
        for (int me = v[0], i = 0; i < v.length; ++i) {
if ((++counter$29 & 262143) == 0) TimeoutChecker.check();
            if (v[i] > me) {
                me = v[i];
                same = i;
            }
        }
        same_cnt = v[same];
        ve = 0;
        for (int i = 0; i < n; ++i) {
if ((++counter$30 & 262143) == 0) TimeoutChecker.check();
            if (a[i] != same) {
                v[ve++] = i;
            }
        }
        while (true) {
if ((++counter$31 & 262143) == 0) TimeoutChecker.check();
            random.shuffle(v, 0, ve);
```

```
            random.shuffle(q, 0, m);
            if (solve(m - 1)) {
                break;
            }
        }
        fit(q[0]);
    }
}
```

A.7 Программа 7294221

```
import java.util.*;

import ru.ifmo.ctd.ngp.demo.testgen.TimeoutChecker;
public class J7294221P {
    private static long counter$0 = 0;
    /* counter$1 ... counter$92 omitted for brevity */
    private static long counter$93 = 0;
    public static void profilerCleanup() {
        counter$0 = 0;
        /* counter$1 ... counter$92 omitted for brevity */
        counter$93 = 0;
    }
    public static java.util.Map<String, Long> profilerTimeData() {
        java.util.Map<String, Long> rv = new java.util.HashMap<>();
        rv.put("counter$0", counter$0);
        /* counter$1 ... counter$92 omitted for brevity */
        rv.put("counter$93", counter$93);
        return rv;
    }
    private static int n;
    private static int[] a;
    private static int[] pa;
    private static int m;
    private static int[] L;
    private static int[] pL;
    private static int[] r;
    private static int maxSize;
    private static final int MAX_VALUE = 100;
    private static int[] pos  = new int[MAX_VALUE + 1];
    private static int[] last = new int[MAX_VALUE + 1];
    private static int size;
    private static int[] q;
    private static int[] f;
    private static double[] f2;
    private static boolean[] used;
    private static Random random;
    private static long[] w;
    private static int[] c;
    private static HashSet<Long>[] visitedStates;
    private static int[] numMg;
```

```
    private static int[][] mg1;
    private static int[][] mg2;
    private static boolean[][][] dp;
    private static int[][] ret;
    private static int[] ss;
    private static boolean printSolution = false;

    private static void init(int nn, int mm) {
if ((++counter$2 & 262143) == 0) TimeoutChecker.check();
        n = nn;
        m = mm;
        size = 0;

        a = new int[n];
        pa = new int[n];
        r = new int[n];
        used = new boolean[n];

        L = new int[m];
        pL = new int[m];

        maxSize = n * (MAX_VALUE + 1);
        q = new int[maxSize];
        f = new int[maxSize];
        f2 = new double[maxSize];

        w = new long[n + 1];
        c = new int[n + 1];

        random = new Random(25735321L);

        visitedStates = new HashSet[m];
        for (int i = 0; i < m; ++i) {
if ((++counter$16 & 262143) == 0) TimeoutChecker.check();
            visitedStates[i] = new HashSet<>();
        }

        numMg = new int[32];
        mg1 = new int[32][32];
        mg2 = new int[32][32];

        dp = new boolean[32][32][maxSize];
        ret = new int[32][32];
        ss = new int[maxSize];
    }

    private static void shuffle(int[] p, int n) {
if ((++counter$24 & 262143) == 0) TimeoutChecker.check();
        for (int i = 0; i < n; ++i) {
if ((++counter$25 & 262143) == 0) TimeoutChecker.check();
            int j = random.nextInt(n - i) + i;
```

```
            int tmp = p[i];
            p[i] = p[j];
            p[j] = tmp;
        }
    }

    private static void reverse(int[] a, int n) {
if ((++counter$27 & 262143) == 0) TimeoutChecker.check();
        for (int i = 0, j = n - 1; i < j; ++i, --j) {
if ((++counter$28 & 262143) == 0) TimeoutChecker.check();
            int tmp = a[i];
            a[i] = a[j];
            a[j] = tmp;
        }
    }

    private static void computeF2() {
if ((++counter$29 & 262143) == 0) TimeoutChecker.check();
        int mL = 0;
        for (int i = 0; i < m; ++i) {
if ((++counter$30 & 262143) == 0) TimeoutChecker.check();
            if (mL < L[i]) {
                mL = L[i];
            }
        }
        Arrays.fill(f2, 0, mL + 1, 0.0);
        f2[0] = 1;

        // TODO: reimplement not boxed
        TreeMap<Integer, Integer> mc = new TreeMap<>();
        for (int i = 0; i < n; ++i) {
if ((++counter$32 & 262143) == 0) TimeoutChecker.check();
            Integer key = a[i];
            Integer v = mc.get(key);
            v = v == null ? 1 : v + 1;
            mc.put(key, v);
        }

        for (Map.Entry<Integer, Integer> p : mc.entrySet()) {
if ((++counter$35 & 262143) == 0) TimeoutChecker.check();
            int s = p.getKey();
            int c = p.getValue();
            for (int k = mL; k >= s; --k) {
if ((++counter$37 & 262143) == 0) TimeoutChecker.check();
                for (int d = k - s, i = 1; d >= 0 && i <= c; ++i, d -= s) {
if ((++counter$38 & 262143) == 0) TimeoutChecker.check();
                    f2[k] += f2[d];
                }
            }
        }
    }
```

```
    private static void refine(int current) {
if ((++counter$39 & 262143) == 0) TimeoutChecker.check();
        size = 0;
        for (int k = 0; k < n; k++) {
if ((++counter$40 & 262143) == 0) TimeoutChecker.check();
            if (used[k] && r[k] == current) {
                q[size++] = k;
            }
        }
        Arrays.fill(pos, -1);
        for (int k = 0; k < n; k++) {
if ((++counter$41 & 262143) == 0) TimeoutChecker.check();
            if (!used[k]) {
                pos[a[k]] = k;
            }
        }
        while (true) {
if ((++counter$42 & 262143) == 0) TimeoutChecker.check();
            boolean findmore = false;
            for (int i = 0; i < size; i++) {
if ((++counter$43 & 262143) == 0) TimeoutChecker.check();
                for (int j = i + 1; j < size; j++) {
if ((++counter$44 & 262143) == 0) TimeoutChecker.check();
                    if (a[q[i]] + a[q[j]] <= MAX_VALUE && pos[a[q[i]] + a[q[j]]] >= 0) {
                        findmore = true;
                        int idx = pos[a[q[i]] + a[q[j]]];
                        used[q[i]] = used[q[j]] = false;
                        used[idx] = true;
                        r[idx] = current;
                        q[i] = idx;
                        q[j] = q[--size];
                        Arrays.fill(pos, -1);
                        for (int k = 0; k < n; k++) {
if ((++counter$45 & 262143) == 0) TimeoutChecker.check();
                            if (!used[k]) {
                                pos[a[k]] = k;
                            }
                        }
                    }
                }
            }
            if (!findmore) {
                break;
            }
        }
    }

    private static void sortPLbyF2Dec() {
if ((++counter$46 & 262143) == 0) TimeoutChecker.check();
        int right = m - 1;
```

```
        for (int i = 0, j = i; i < right; j = ++i) {
if ((++counter$47 & 262143) == 0) TimeoutChecker.check();
            int aii = pL[i + 1];
            double ai = f2[L[aii]];
            while (ai > f2[L[pL[j]]]) {
if ((++counter$48 & 262143) == 0) TimeoutChecker.check();
                pL[j + 1] = pL[j];
                if (j-- == 0) {
                    break;
                }
            }
            pL[j + 1] = aii;
        }
    }

    private static void sortPLbyF2Inc() {
if ((++counter$49 & 262143) == 0) TimeoutChecker.check();
        int right = m - 1;
        for (int i = 0, j = i; i < right; j = ++i) {
if ((++counter$50 & 262143) == 0) TimeoutChecker.check();
            int aii = pL[i + 1];
            double ai = f2[L[aii]];
            while (ai < f2[L[pL[j]]]) {
if ((++counter$51 & 262143) == 0) TimeoutChecker.check();
                pL[j + 1] = pL[j];
                if (j-- == 0) {
                    break;
                }
            }
            pL[j + 1] = aii;
        }
    }

    private static void randomFits() {
if ((++counter$52 & 262143) == 0) TimeoutChecker.check();
        computeF2();
        for (int step = 0; ; step++) {
if ((++counter$53 & 262143) == 0) TimeoutChecker.check();
            if (step % 5 == 0) {
                Arrays.sort(a, 0, n);
                reverse(a, n);
            } else {
                shuffle(a, n);
            }
            if (step % 2 == 0) {
                sortPLbyF2Dec();
            } else {
                shuffle(pL, m);
            }
            if (step > 0 && step % 10 == 0) {
                continue;
```

```
            }

            boolean success = true;
            Arrays.fill(used, false);
            for (int cl = 0; cl < m; ++cl) {
if ((++counter$54 & 262143) == 0) TimeoutChecker.check();
                int ridx = pL[cl];
                if (cl == m - 1) {
                    for (int i = 0; i < n; ++i) {
if ((++counter$55 & 262143) == 0) TimeoutChecker.check();
                        if (!used[i]) {
                            r[i] = ridx;
                        }
                    }
                    break;
                }
                int length = L[ridx];
                f[0] = 0;
                Arrays.fill(f, 1, length + 1, -1);
                q[0] = 0;
                size = 1;
                Arrays.fill(last, 0);
                for (int i = 0; i < n; ++i) {
if ((++counter$56 & 262143) == 0) TimeoutChecker.check();
                    if (used[i]) continue;
                    int c = a[i];
                    int oldSize = size;
                    for (int stateCl = last[c]; stateCl < oldSize; ++stateCl) {
if ((++counter$57 & 262143) == 0) TimeoutChecker.check();
                        int t = q[stateCl] + c;
                        if (t <= length && f[t] < 0) {
                            f[t] = i;
                            q[size++] = t;
                        }
                    }
                    last[c] = oldSize;
                    if (f[length] >= 0) {
                        break;
                    }
                }
                if (f[length] < 0) {
                    success = false;
                    break;
                }
                for (int k = length; k > 0; k -= a[f[k]]) {
if ((++counter$58 & 262143) == 0) TimeoutChecker.check();
                    r[f[k]] = ridx;
                    used[f[k]] = true;
                }
                refine(ridx);
            }
```

```
            if (success) {
                if (printSolution) {
                    for (int i = 0; i < m; ++i) {
if ((++counter$59 & 262143) == 0) TimeoutChecker.check();
                        int cnt = 0;
                        for (int k = 0; k < n; ++k) {
if ((++counter$60 & 262143) == 0) TimeoutChecker.check();
                            if (r[k] == i) {
                                ++cnt;
                            }
                        }
                        System.out.println(cnt);
                        for (int k = 0; k < n; ++k) {
if ((++counter$61 & 262143) == 0) TimeoutChecker.check();
                            if (r[k] == i) {
                                System.out.print(a[k] + " ");
                            }
                        }
                        System.out.println();
                    }
                }
                return;
            }
        }
    }

    private static boolean dfs2(int depth, int leftLength, int idx, long state) {
if ((++counter$62 & 262143) == 0) TimeoutChecker.check();
        if (idx == 0) {
            return dfs(depth + 1, state);
        }
        if (idx == 1) {
            --idx;
            int cnt = leftLength / a[idx];
            c[idx] -= cnt;
            ret[depth][idx] = cnt;
            if (dfs(depth + 1, state + cnt * w[idx])) {
                return true;
            }
            c[idx] += cnt;
            return false;
        }

        --idx;
        int maxc = Math.min(c[idx], leftLength / a[idx]);
        int newLeftLength = leftLength - a[idx] * maxc;
        long newState = state - w[idx] * maxc;
        for (int i = maxc; i >= 0; --i, newState += w[idx], newLeftLength += a[idx]) {
if ((++counter$63 & 262143) == 0) TimeoutChecker.check();
            if (!dp[depth][idx][newLeftLength]) {
                continue;
```

```
            }
            c[idx] -= i;
            ret[depth][idx] = i;
            if (dfs2(depth, newLeftLength, idx, state + i * w[idx])) {
                return true;
            }
            c[idx] += i;
        }
        return false;
    }

    private static boolean dfs(int depth, long state) {
if ((++counter$64 & 262143) == 0) TimeoutChecker.check();
        if (depth == m) {
            if (printSolution) {
                for (int ci = 0; ci < m; ++ci) {
if ((++counter$65 & 262143) == 0) TimeoutChecker.check();
                    int i = 0;
                    while (pL[i] != ci) {
if ((++counter$66 & 262143) == 0) TimeoutChecker.check();
                        ++i;
                    }
                    int cnt = 0;
                    for (int j = 0; j < n; ++j) {
if ((++counter$67 & 262143) == 0) TimeoutChecker.check();
                        cnt += ret[i][j];
                    }
                    System.out.println(cnt);
                    for (int j = 0; j < n; ++j) {
if ((++counter$68 & 262143) == 0) TimeoutChecker.check();
                        for (int k = 0; k < ret[i][j]; ++k) {
if ((++counter$69 & 262143) == 0) TimeoutChecker.check();
                            System.out.print(a[j] + " ");
                        }
                    }
                    System.out.println();
                }
            }
            return true;
        }
        if (visitedStates[depth].contains(state)) {
            return false;
        }
        visitedStates[depth].add(state);
        int length = L[pL[depth]];
        Arrays.fill(dp[depth][0], 0, length + 1, false);
        dp[depth][0][0] = true;
        for (int i = 0; i < n; ++i) {
if ((++counter$72 & 262143) == 0) TimeoutChecker.check();
            boolean[] src = dp[depth][i];
            boolean[] target = dp[depth][i + 1];
```

```
            if (c[i] == 0) {
                System.arraycopy(src, 0, target, 0, length + 1);
                continue;
            }
            int a0 = a[i];
            if (c[i] == 1) {
                System.arraycopy(src, 0, target, 0, length + 1);
                for (int j = a0; j <= length; ++j) {
if ((++counter$73 & 262143) == 0) TimeoutChecker.check();
                    if (src[j - a0]) {
                        target[j] = true;
                    }
                }
            } else {
                int w = a0 * (c[i] + 1);
                for (int j = 0; j <= length; ++j) {
if ((++counter$74 & 262143) == 0) TimeoutChecker.check();
                    ss[j] = src[j] ? 1 : 0;
                    if (j >= a0) ss[j] += ss[j - a0];
                    if (j >= w) {
                        target[j] = ss[j] > ss[j - w];
                    } else {
                        target[j] = ss[j] != 0;
                    }
                }
            }
        }
        return dp[depth][n][length] && dfs2(depth, length, n, state);
    }

    private static void bruteForce() {
if ((++counter$75 & 262143) == 0) TimeoutChecker.check();
        computeF2();
        TreeMap<Integer, Integer> mc = new TreeMap<>();
        for (int i = 0; i < n; ++i) {
if ((++counter$77 & 262143) == 0) TimeoutChecker.check();
            Integer key = a[i];
            Integer v = mc.get(key);
            v = v == null ? 1 : v + 1;
            mc.put(key, v);
        }
        n = 0;
        for (Map.Entry<Integer, Integer> t : mc.entrySet()) {
if ((++counter$80 & 262143) == 0) TimeoutChecker.check();
            a[n] = t.getKey();
            c[n++] = t.getValue();
        }
        w[0] = 1;
        for (int i = 0; i < n; ++i) {
if ((++counter$82 & 262143) == 0) TimeoutChecker.check();
            w[i + 1] = w[i] * (c[i] + 1);
```

```
        }
        Arrays.fill(numMg, 0, numMg.length, 0);
        for(int i = 0; i < n; ++i) {
if ((++counter$83 & 262143) == 0) TimeoutChecker.check();
            for (int j = i + 1; j < n; ++j) {
if ((++counter$84 & 262143) == 0) TimeoutChecker.check();
                for (int k = j + 1; k < n; ++k) {
if ((++counter$85 & 262143) == 0) TimeoutChecker.check();
                    if (a[i] + a[j] == a[k]) {
                        mg1[i][numMg[i]] = j;
                        mg2[i][numMg[i]++] = k;
                    }
                }
            }
        }
        for (int i = 0; i < m; ++i) {
if ((++counter$86 & 262143) == 0) TimeoutChecker.check();
            pL[i] = i;
        }
        sortPLbyF2Inc();
        dfs(0, 0);
    }

    public void solve(List<Integer> ships, List<Integer> havens) {
if ((++counter$87 & 262143) == 0) TimeoutChecker.check();
        init(ships.size(), havens.size());
        for (int i = 0; i < n; ++i) {
if ((++counter$88 & 262143) == 0) TimeoutChecker.check();
            a[i] = ships.get(i);
            pa[i] = i;
        }
        for (int i = 0; i < m; ++i) {
if ((++counter$89 & 262143) == 0) TimeoutChecker.check();
            L[i] = havens.get(i);
            pL[i] = i;
        }
        Set<Integer> differentA = new HashSet<>();
        for (int i = 0; i < n; ++i) {
if ((++counter$91 & 262143) == 0) TimeoutChecker.check();
            differentA.add(a[i]);
        }
        if (differentA.size() <= 21) {
            bruteForce();
        } else {
            randomFits();
        }
    }
}
```