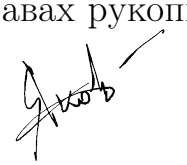
Федеральный исследовательский центр «Информатика и управление»
Российской академии наук

На правах рукописи



Яковлев Константин Сергеевич

Методы и алгоритмы эвристического поиска на графах регулярной декомпозиции в задачах планирования траекторий мобильных роботов

Специальность 1.2.1 —
Искусственный интеллект и машинное обучение

Диссертация на соискание учёной степени
доктора физико-математических наук

Москва — 2026

Оглавление

Стр.

Стр.

Стр.

Введение

Актуальность темы исследования. Задачи планирования, т.е. задачи выбора последовательности некоторых действий для достижения поставленных целей на протяжении долгого времени являются одними из центральных задач в искусственном интеллекте. Одним из распространенных и практически важных классов таких задач являются задачи планирования перемещений мобильных роботов, зачастую именуемых в искусственном интеллекте агентами. При планировании перемещения (траектории) возникают графы, вложенные в метрическое пространство. Вершинам таких графов соответствуют различные положения агента, в ребрам – так называемые элементарные траектории, т.е. такие траектории следование вдоль которых (возможно - с некоторой допустимой погрешностью) может быть обеспечено системой управления мобильного робота в автоматическом режиме. Это могут быть отрезки прямых, фрагменты окружности определенного радиуса, криволинейные фрагменты, удовлетворяющие заданным требованиям (например – описываемые с помощью уравнений n-го порядка) и др. Среди графовых моделей, наиболее часто применимых для решения задач планирования траектории, можно выделить графы видимости [1], диаграммы Вороного [2], вероятностые схемы местности [3] и др. При этом наиболее часто встречающейся в практических приложениях графовой моделью для широкого класса задач планирования траектории является граф, который может быть получен наложением регулярной сетки на рабочее пространство агента, для которого осуществляется планирование [4; 5]. Это граф представляется в виде совокупности клеток, каждая их которых либо проходима, либо нет для агента. Вершины графа располагаются в центрах проходимых клеток (иногда, вершины располагают не в центрах клеток, а в углах, т.е. на пересечении линий сетки, образующей граф). Ребра задаются имплицитно. В самом простом случае, каждая вершина связана с двумя горизонтально-смежными вершинами и двумя вертикально-смежными. Зачастую, допускаются также диагональные переходы, т.е. число потенциальных соседей у вершины увеличивается до 8. В самом общем случае, ребро может связывать две произвольные вершины

графа [6; 7]. Будем называть такие регулярные графы – графами регулярной декомпозиции[1]

Для указанных графов могут применяться известные алгоритмы поиска пути, например алгоритмы неинформированного поиска: Дейкстры [8], Беллмана-Форда [9; 10] и др., алгоритмы эвристического поиска: A* [11], FocalSearch [12] и их модификации. При этом, именно алгоритмы эвристического поиска наиболее востребованы на практике. Во-первых, для графов регулярной декомпозиции известно достаточно много простых (с точки зрения вычислений) и эффективных (с точки зрения фокусирования поиска) эвристических функций, обладающих свойством допустимости [13]. При использовании таких эвристик алгоритмы семейства A* гарантируют отыскание оптимального решения (т.е. кратчайшего пути). Во-вторых, для алгоритмов семейства A* известны техники, которые позволяют отказаться от гарантий отыскания оптимальных решений, но при этом существенно ускорить время работы алгоритма на практике [14]. В третьих, подобные алгоритмы достаточно легко модифицируются и к настоящему моменту известно множество модификаций A*, существенно повышающих его эффективность при поиске путей на графах регулярной декомпозиции. Например, модификации, использующие технику отсечения симметрий [15], или различные декомпозиционные техники декомпозиции [16—18]. Развивается в настоящее время и направление, подразумевающее интеграцию методов эвристического поиска и машинного обучения для решения задач планирования в целом и поиска путей на графах в частности [19; 20].

Итак, исследования, посвященные разнообразным подходам к элиминации перебора при поиске, ведутся достаточно активно и к настоящему моменту известно большое число эффективных, как в вычислительном плане, так и в плане предоставляемых теоретических гарантий, методов построения пути на графах регулярной декомпозиции, применимых для задач планирования. Однако, большинство практически-мотивированных задач, особенно в контексте планирования для мобильных робототехнических систем, требуют учета дополнительных факторов, среди которых следует отметить следующие. Во-первых, необходимо планировать траекторию с учетом (прогнозной) динамики

[1]Для обозначения схожих графов в литературе обычно используется термин графы-решётки или решётчатые графы. Однако последние являются регулярными сетками (мозаиками) в $\mathcal{R}^n$. В робототехнике и искусственном интеллекте из-за моделирования препятствий для движения агентов рассматриваются не сами решётки, а их частичные подграфы, поэтому вводится термин граф регулярной декомпозиции.

окружающей среды. Во-вторых, во многих приложениях встречается многоагентная постановка задачи, следовательно, необходимо учитывать цели других агентов и планировать координированные и безопасные траектории в общей рабочей среде. В-третьих, многие мобильные робототехнические системы обладают кинематическими ограничениями, то есть они не могут мгновенно менять направление своего движения. Наконец, несмотря на то, что распространенные эвристики используемые при поиске путей на графах регулярной декомпозиции, такие как расстояние Манхэттена или Чебышева, являются легко-вычислимыми и обладают свойством допустимости (что позволяет давать определенные теоретические гарантии), они не учитывают расположение препятствий в рабочем пространстве агента. Соответственно, при поиске пути в среде с препятствиями, что наиболее часто встречается на практике, не происходит желаемого сокращения пространства поиска. В некоторых случаях число рассмотренных состояний поиска близко или совпадает с числом состояний, рассматриваемых неиформированным поиском, т.е. поиском без эвристики (например, с помощью алгоритма Дейкстры). Таким образом, возникает потребность в методах автоматического конструирования более информативных (т.е. сокращающих пространство поиска) эвристик, которые учитывали бы расположение препятствий в рабочем пространстве агента.

Несомненно, к настоящему времени достигнут определенный прогресс в преодолении указанных выше проблем, однако имеющиеся методы не лишены и существенных недостатков. Так, для планирования в среде с динамическими препятствиями обычно применяются техники быстрого перепланирования, которые основаны на переиспользовании построенного на предыдущих итерациях дерева поиска [21; 22]. Подобные подходы во многих случаях приводят к зацикливающимся траекториям, даже когда траектории движения динамических препятствий известны планировщику, т.к. информация о последних фактически не используется. Также в последнее время набирает популярность подход, основанный на применении (глубоких) искусственных нейросетей для создания обучаемых методов безопасного движения в динамических средах [23—25]. Однако эти методы обычно сильно зависят от типа входных данных и обучающей выборки. В общем случае вопрос их генерализуемости, т.е. способности находить решение в случаях, которые не близки к примерам, использованным при обучении, остается открытым (что может быть критично для многих реальных робототехнических систем).

Для решения задачи (централизованного) планирования неконфликтных траекторий для группы мобильных роботов обычно вводится ряд ограничивающих допущений, касающихся продолжительности действий перемещения и ожидания, а также возможности смены направления движения. Так, обычно считается, что агенты могут перемещаться лишь по горизонтали и вертикали и каждое перемещение, так же как и ожидание, совершается за один такт времени [26]. Такой подход позволяет легко идентифицировать потенциальные конфликты и избегать их при планировании. Однако полученные в результате траектории содержат большое число поворотов и их суммарная продолжительность может быть достаточно велика. Известны также децентрализованное подходы к решению задачи много-агентной навигации. Обычно они опираются на реактивное избегание столкновений в некоторой локальной области и могут быть как обучаемыми [27—29] так и необучаемыми [30; 31]. При этом ни те ни другие методы не гарантируют в общем случае, что задача будет решена и все агенты достигнут целей.

Для учета кинематических ограничений существует множество подходов. Во-первых, можно отказаться от идеи формализации задачи планирования как задачи поиска пути на графе и перейти к другим постановкам, среди которых наиболее распространены задача оптимального управления [32—35] и планирование на основе случайного семплирования [36; 37]. При этом методы решения задач оптимального управления обычно могут работать только с аналитическим описанием запрещенных областей, т.е. препятствий, в то время как на практике такое описание недоступно. Эффективность же семплирующих планировщиков сильно зависит от случая, т.к. случайный выбор является неотъемлемой процедурой этих методов.

Для автоматического конструирования информативных эвристик, учитывающих расположение препятствий в рабочей области, в последнее время активно используются методы машинного обучения [38—40]. Однако, при использовании подобных эвристик встает вопрос, во-первых, о гарантиях качества отыскиваемых решений, во-вторых, о генерализуемости, т.е. о способности метода конструировать информативные эвристики для заданий, которые отличаются от тех, что были использованы на этапе обучения.

В целом, несмотря на активное развитие методов поиска путей на графах регулярной декомпозиции, применимых для решения задач планирования траектории мобильных роботов, в том числе за счет их интеграции с методами

машинного обучения, а также несмотря на наличие и развитие альтернативных подходов к решению задач поиска и планирования, существующие методы не лишены ряда недостатков, затрудняющих их практическое использование и применение. Таким образом, разработка и исследование методов и алгоритмов поиска и планирования, учитывающих важные особенности и ограничения, вытекающие из практических нужд, таких как, например, много-агентная постановка, учет кинематических ограничений, учет динамики окружающей среды, является важной и актуальной научно-технической задачей современности. Именно разработке подобных методов и алгоритмов и посвящена данная работа, что обуславливает её актуальность.

Степень разработанности темы. Методы и алгоритмы поиска пути (путей) на графах активно развиваются с середины XX века. Среди основополагающих работ в этой области можно отметить работы Э. Дейкстры, Л. Беллмана, Р. Форда и др. Каноническими публикациями принято считать работы: "Заметка о двух проблемах, связанных с графами"[2] [8], в которой Дейкстрой был предложен одноименный алгоритм поиска пути; "О Задаче Маршрутизации"[3] [9] за авторством Бэллмана и книгу Форда и Фалкерсона "О потоках в сетях"[4] [10]. В 70-х годах получили развитие методы эвристического поиска, среди которых основополагающим можно считать алгоритм A* [11], предложенный Хартом, Нильсоном и Рафаэлем в статье "Формальные основания построения путей наименьшей стоимости с помощью эвристик"[5] [11]. Важно отметить, что особое внимание в сообществе исследователей методов эвристического поиска уделялось гарантиям, которые предоставляют такие алгоритмы, в частности гарантиям оптимальности отыскиваемых решений, которые зависят от вида используемой эвристической функции и методологии поиска [13]. В 80-90х годах XX века в связи с развитием мобильной робототехники активно исследовались методы поиска путей на графах регулярной декомпозиции, которые применялись для автоматического планирования траектории мобильных роботов. В этом контексте стоит отметить таких авторов как Э. Стентс, М. Лихачев, С. Кениг, С. Тран и др., и предложенные ими ал-

[2]Оригинальное название на английском языке – A Note On Two Problems In Connexion With Graphs.

[3]Оригинальное название на английском языке – On A Routing Problem.

[4]Оригинальное название на английском языке – Flow In Networks.

[5]Оригинальное название на английском языке – A formal basis for the heuristic determination of minimum cost paths.

горитмы D* [41], LPA* [42], ARA* [43], D*Lite [21] и др. К значимым результатам в области поиска путей на графах регулярной декомпозиции, полученным в начале XXI века, можно отнести алгоритмы поиска с достраиванием ребер графа по ходу поиска (см. например работу Э. Неша, предлагающую алгоритм Theta* [7]), алгоритмы отсечения симметрий при поиске (см., например, алгоритм JPS [15] за авторством Д. Харабора). К другим темам, которые получили развитие можно отнести двунаправленный поиск (работы А. Фелнера, Р. Холте, см. например [44]), иерархический поиск (HPA* [16], HGA* [18], R* [17]), поиск ограниченно суб-оптимальных решений [45], поиск совокупности неконфликтных путей на графах. Последняя тема является одной из наиболее активно развивающихся в последнее десятилетие (2010 г. - н.в.). Это может быть объяснено двумя факторами. Во-первых, даже при наличии существенного числа ограничивающих допущений, получение оптимальных решений является NP-трудной задачей [46] (при этом для некоторых вариантов задачи известны полиномиальные алгоритмы получения субоптимальных решений). Во-вторых, методы построение совокупности неконфликтных путей высоко востребованы в практических приложениях, в частности в задачах робототехники, автоматизированной складской логистики и др. В этой области следует отметить таких исследователей как Т. Стенли, А. Фелнер, Р. Штерн, Н. Стюртевант, С. Кёниг и др., и такие методы как A*+ID+OD [47], CBS [48], M* [49], ICTS [50] и др.

Российские ученые также внесли существенных вклад в теорию графов и разработку методов и алгоритмов напрямую или косвенно связанных с вопросами построения пути/путей на графах. Так, например, известны труды Ерусалимского Я.М., Скороходова В.А., посвященные графам с нестандартной достижимостью, поиском потоков на таких графах, задачам о ресурсных сетях [51; 52]. Последним направлением также занимались и внесли существенный вклад в его развитие Жилякова Л.Ю, Кузнецов О.П (см., например, работы [53—55]). Известны работы Р.П. Агаева и П.Ю. Чеботарёва по алгебраической теории ориентированных графов [56; 57]. Значимый вклад в решение различных вариантов одной из классических графовых задач – задачи коммивояжера, – внесли Ченцов П.А., Ченцов А.Г., Курейчик В.М. [58—61] Проблематикой случайных графов, задачами раскраски (гипер)графов продуктивно занимались Пузынина С.А., Райгородский А.М., Шабанов Д.А. [62—64].

С точки зрения искусственного интеллекта, как междисциплинарной области науки, ставящей своей целью разработку и исследование искусственных

систем и устройств, способных к целенаправленному поведению и разумным рассуждениям [65][6], задача поиска (в т.ч. поиска в графе/дереве) тесно связана с задачей планирования, которая традиционно занимает одно из центральных мест в искусственном интеллекте. Такой интерес к планированию обусловлен рядом факторов. Во-первых, планирование подразумевает автоматизацию процесса рассуждения о действиях и последствиях действий, и такой тип высокоуровневых рассуждений, по-видимому, является неотъемлемым условиям разумного, целенаправленого поведения, которое и пытаются моделировать и/или имитировать исследователи в области искусственного интеллекта [66]. Во-вторых, методы решения задачи построения эффективных планов достижения целей находят множество применений на практике (планирование цепочек технологических операций, планирование маршрутов перемещения по городу (автомобильные навигаторы), планирование перемещений роботов на заводе и т.д.). Задачам планирования всегда уделялось большое внимание в искусственном интеллекте. Например, одним из первых широко известных проектов в области воплощенного искусственного интеллекта был проект по созданию автономного мобильного робота Shakey в исследовательском институте Стэнфорда (Stanford Research Institute). В ходе этого проекта была разработана система автоматического планирования STRIPS [67] за авторством Файкса и Нильсена, которая определила границы задачи и методов, её решающих, на годы вперёд. Среди классических работ в области автоматического планирования можно назвать работы [68—73]. В этих работах предлагались и исследовались такие идеи как: планирование как доказательство теорем, вычислительная сложность решения задач планирования, частично-упорядоченное планирование, иерархическое планирование и др. Весьма сильное влияние на область оказал формализм (язык описания задач планирования) PDDL [74]. При этом наиболее заметный прорыв в области произошел на стыке XX и XXI веков, когда было предложено рассматривать задачу планирования как задачу поиска в пространстве состояний и использовать эвристический поиск

[6]Безусловно всегда существовало и существует множество различных определений искусственного интеллекта как области науки. Указанное определение представляется, во-первых, справедливым, во-вторых, достаточно лаконичным и при этом ёмким. Его автор – Геннадий Семёнович Осипов, выдающийся советский и российский исследователь в области искусственного интеллекта и когнитивного моделирования. Почетный член Европейской ассоциации искусственного интеллекта (ECAI Fellow), президент Российской ассоциации искусственного интеллекта с 1996 по 2022 г.

для решения этой задачи – см. такие работы как [75—77] и такие системы как `GraphPlan`, `FF` (Fast Forward), `FD` (Fast Downward) и др. Начиная с 10-х годов XXI века наибольшее внимание исследователей в области искусственного интеллекта было посвящено методам обучения глубоких искусственных нейронных сетей. Одновременно с этим большой успех получили методы, реализующие идею интеграции глубокого обучения и классических алгоритмов поиска для решения задач планирования. В качестве наиболее яркого примера можно упомянуть систему `AlphaGo` [78], предназначенную для решения задачи выбора наиболее перспективного хода в настольной игре Го (пространство поиска которой насчитывает 10^{170} состояний) и впервые в истории человечества выигравшую серию партий у действующего чемпиона мира. `AlphaGo` опирается на особый вид поиска по графу (дереву) состояний для моделирования различных вариантов развития игры (т.е. фактические осуществляет планирование) и использует нейросетевые модели для фокусировки поиска. Схожие методы поиска используются и в настоящее время (20-е года XXI века) в контексте т.н. рассуждающих больших языковых моделей [79—81] для построения графа (дерева) рассуждений, что существенным образом повышает эффективность современных больших языковых моделей в решении нетривиальных задач, когда способность к планированию (т.е. рассуждению о последствиях тех или иных действий) является одним из существенных факторов, напрямую влияющих на эффективность модели и качество итогового решения (например текстового/символьного ответа на сложную математическую задачу). В целом, на современном этапе развития искусственного интеллекта методы и алгоритмы систематического (в т.ч. эвристического) поиска и автоматического планирования продолжают играть существенную роль для прогресса в области, поэтому их совершенствование представляет собой важную научную задачу.

Цели и задачи исследования. Основной *целью* работы является повышение эффективности современных систем управления мобильными роботами при решении навигационных задач за счет разработки и исследования новых методов поиска пути (совокупности путей) на графах, вложенных в метрическое пространство, и применимых для решения задач автоматического планирования траектории (совокупности неконфликтных траекторий).

В соответствии с указанной целью *задачами* исследования являются:

1. Разработка новых методов поиска пути графах регулярной декомпозиции, применимых для планирования траектории в среде со ста-

тическими и динамическими препятствиями (траектории движения которых известны); проведение теоретических исследований предложенных методов; разработка алгоритмов их реализующих; проведение экспериментальных исследований разработанных алгоритмов.

2. Разработка новых методов поиска совокупности неконфликтных путей на графах регулярной декомпозиции, применимых для решений задачи централизованного много-агентного планирования траекторий для группы агентов; проведение теоретических исследований предложенных методов; разработка алгоритмов их реализующих; проведение экспериментальных исследований разработанных алгоритмов, в том числе на реальных робототехнических платформах.
3. Разработка новых методов поиска пути на графах регулярной декомпозиции, косвенно учитывающих кинематические ограничения мобильного агента; проведение теоретических исследований предложенных методов; разработка алгоритмов их реализующих; проведение экспериментальных исследований разработанных алгоритмов.
4. Разработка новых методов автоматического конструирования эвристических функций для решения задач поиска пути на графе регулярной декомпозиции с помощью методов машинного обучения, в частности – глубокого обучения с применением современных нейросетевых архитектур, а также способов интеграции этих функций в алгоритмы эвристического поиска; программная реализация предлагаемых подходов и проведение экспериментальных исследований.

Научная новизна работы состоит в следующем:

– Разработан новый метод поиска пути на динамическом графе регулярной декомпозиции, допускающий использование переходов между произвольными вершинами графа, основанный на принципе безопасно-интервального планирования и оригинальном методе обращения направления поиска в сочетании с принципом ленивого поиска (т.е. откладывания наиболее вычислительно ресурсоемких операций до определенного момента). Указанный метод позволяет решить задачу планирования траектории мобильного агента в среде с динамическими препятствиями. В отличие от аналогов разработанный метод допускает перемещение агента в произвольном направлении, что положительно сказывается на качестве отыскиваемых решений (длина траектории и

время прибытия в заданное целевое положение сокращаются). Предложены новые алгоритмы поиска субоптимальных решений задачи поиска пути на динамическом графе регулярной декомпозиции.

– Разработаны новые методы и алгоритмы построения совокупности неконфликтных путей на графе регулярной декомпозиции, опирающиеся на принципы конфликтно-ориентированного поиска и приоритизированного планирования. Указанные методы и алгоритмы используют оригинальные техники элиминации перебора, что существенно повышает их эффективность при сохранении теоретических гарантий. Проведено экспериментальное исследование предложенных алгоритмов, в т.ч. на реальных робототехнических системах.
– Предложено семейство алгоритмов поиска на графе регулярной декомпозиции, косвенно учитывающих кинематические ограничения мобильного агента, а именно – ограничения на максимальный угол отклонения между прямолинейными сегментами, образующими траекторию. Исследованы теоретические свойства алгоритмов семейства. Сформулированы и доказаны гарантии отыскания решений в определенном классе. Проведено эмпирическое исследование предложенных алгоритмов.
– Предложены новые типы эвристических функций для задачи поиска пути на графе регулярной декомпозиции, которые для каждого экземпляра задачи поиска учитывают его специфику и успешно аппроксимируются современными нейросетевыми моделями. Разработаны алгоритмы поиска, основанные на комбинации предложенных обучаемых эвристик и классических техник систематизации поиска, обладающие строгими теоретическими гарантиями. Эффективность применения разработанных алгоритмов на практике подтверждается многочисленными численными экспериментами (в том числе экспериментами по решении задач визуальной навигации, т.е. построения опорного маршрута мобильного робота по изображению подстилающей поверхности).

Теоретическая и практическая значимость заключается в следующем. С теоретической точки зрения предложен ряд новых методов, предназначенных для решения различных вариаций задач поиска пути (или совокупности путей) на графах регулярной декомпозиции, исследованы свойства предложенных методов (доказан ряд теорем о предоставляемых алгоритмами гарантиях).

Их создание расширяет и обогащает основы теории графов и искусственного интеллекта и может служить фундаментом для дальнейших исследований в этой области. Отдельно стоит отметить, что предложенные в работе способы интеграции обучаемых (с помощью современных искусственных нейронных сетей) эвристик и алгоритмов классического поиска, гарантируют построение корректного решения, а также, в отдельных случаях, гарантируют отыскание решения, превышающего по стоимости оптимальное решение не более, чем в заданное пользователем число раз.

С практической точки зрения предложенные методы и алгоритмы могут применяться в робототехнических системах для повышения степени их автономности за счет повышения эффективности решения задач планирования траектории (с учетом различных ограничений). Так, ряд предложенных методов уже был апробирован в ходе работы на колесных мобильных роботах (в частности были апробированы методы построения неконфликтных траекторий для группы роботов). Результаты проведенных экспериментальных исследований подтвердили применимость разработанных подходов на практике.

Апробация работы. Результаты работы неоднократно докладывались на профильных Российских и международных научных мероприятиях (конференциях, семинарах и пр.), среди которых можно отметить следующие:

- Международная конференция “Системный анализ и информационные технологии” (САИТ) 2015 (Светлогорск, Россия)
- Всероссийский научно-практический семинар “Беспилотные транспортные средства с элементами искусственного интеллекта” (БТС-ИИ), 2016 (Иннополис, Россия), 2017 (Казань, Россия), 2019 (Санкт-Петербург, Россия), 2021 (Москва, Россия)
- Национальная конференция по искусственному интеллекту с международным участием (КИИ) 2016 (Смоленск, Россия), 2018 (Москва, Россия), 2021 (Таганрог, Россия)
- Международная научно-техническая конференция “Экстремальная робототехника” (ЭР) 2016 (Санкт-Петербург, Россия)
- Совместный семинар Российской ассоциации искусственного интеллекта и Федерального исследовательского центра “Информатика и управление” Российской академии наук (семинар РАИИ-ФИЦ ИУ РАН) 2023 (Москва, Россия), 2025 (Москва, Россия)

- German Conference on Artificial Intelligence (KI) 2015 (Дрезден, Германия), 2019 (Кассель, Германия)
- Workshop on Multi-agent Pathfinding (WoMAPF) 2016 (Нью-Йорк, США), 2019 (Макао, КНР), 2026 (Сингапур)
- International Conference on Automated Planning and Scheduling (ICAPS) 2017 (Питтсбург, США), 2020 (Франция, онлайн), 2021 (КНР, онлайн)
- International Conference on Autonomous Agents and Multi-agent Systems (AAMAS) 2018 (Стокгольм, Швеция)
- International Conference on Informatics in Control, Automation and Robotics (ICINCO) 2020 (Франция, онлайн)
- European Conference on Mobile Robotics (ECMR) 2019 (Прага, Чехия)
- Conference on Interactive Collaborative Robotics (ICR) 2019 (Стамбул, Турция), 2020 (Россия, онлайн)
- IFAC Symposium on Robot Control (SyRoCo) 2022 (Япония, онлайн)
- AAAI Conference on Artificial Intelligence (AAAI) 2023 (США, онлайн)
- International Symposium on Combinatorial Search (SoCS) 2024 (Канада, 2024)
- IEEE Conference on Intelligent Robots and Systems (IROS) 2024 (ОАЭ, 2024), 2025 (Ханчжоу, КНР)

Материалы настоящей работы используются при чтении курсов “Методы эвристического планирования” (магистерская программа “Методы и технологии искусственного интеллекта”, Московский физико-технический институт), “Методы и алгоритмы эвристического поиска” (бакалаврская программа “Науки о данных”, Санкт-Петербургский государственный университет).

Ряд результатов был получен в процессе выполнения работ по следующим грантам Российского фонда фундаментальных исследований (РФФИ), Российского научного фонда (РНФ), Министерства науки и высшего образования РФ (МНВО):

- 075-15-2024-544, “Математические модели и численные методы как основа для разработки робототехнических комплексов, новых материалов и интеллектуальных технологий конструирования” (МНФО РФ, 2024-2026 гг., исполнитель).
- 075-15-2020-799, “Методы построения и моделирования сложных систем на основе интеллектуальных и суперкомпьютерных технологий, направ-

ленные на преодоление больших вызовов" (МНФО РФ, 2020-2023 гг., исполнитель).

- 16-11-00048 "Создание теории, методов и моделей децентрализованного управления поведением коллективов когнитивных робототехнических систем в недетерминированной среде" (РНФ, 2016-2018 гг., исполнитель; 2019-2020 гг., руководитель);
- 20-57-00011 Бел_а "Вычислительно эффективные методы навигации группы автономных колесных роботов в динамической среде" (РФФИ, 2020-2021 гг., исполнитель);
- 18-37-20032 мол_а_вед "Методы управления автономными техническими объектами на основе планирования траектории в среде с динамическими препятствиями" (РФФИ, 2019-2020 гг., руководитель);
- 17-29-07053 офи_м "Гетерогенные иерархические системы динамического планирования и управления поведением интеллектуального агента" (РФФИ, 2017-2019 гг., исполнитель);
- 17-07-00281-а "Модели и методы решения задач интеллектуального управления коалицией сложных технических объектов" (РФФИ, 2017-2019 гг., исполнитель);
- 15-07-07483-а "Исследование методов и разработка алгоритмов картирования, локализации и автоматического планирования траектории сложных технических объектов, обладающих многими степенями свободы" (РФФИ, 2015-2017 гг., руководитель);
- 15-37-20893 мол_а_вед "Исследование и разработка методов и алгоритмов планирования траекторий в условиях коллективного решения задач интеллектуальными агентами" (РФФИ, 2015-2016 гг., руководитель);
- 14.514.11.4081 Исследование методов и разработка алгоритмов и программных средств управления современными беспилотными транспортными системами на базе свободно-распространяемого программного обеспечения (МНФО РФ, 2014 г., ответственный исполнитель).
- 14-11-00692 "Создание основ теории и технологии построения многоуровневых систем управления коалициями сложных технических объектов в динамической среде на основе взаимодействия методов искусственного интеллекта и теории динамических систем" (РНФ, 2014-2016 гг., исполнитель).

Достоверность результатов подтверждается строгой теоретической обоснованностью и корректным применением используемого математического аппарата (дискретная математика, теория графов, математическая логика), согласованием полученных результатов с известными теоретическими положениями в рассматриваемой предметной области, а также результатами экспериментальных исследований, выполненными как с использованием открытых данных, широко использующихся в научном сообществе, так и с использованием реальных робототехнических систем (колесных мобильных роботов).

Публикации. Основные результаты работы опубликованы в ведущих Российских и международных научных журналах и сборниках трудов по профилю исследования (эвристический поиск, планирование траектории, искусственный интеллект, мобильная робототехника). Общее число публикаций – 54, среди которых можно выделить следующие группы публикаций:

- Публикации в ведущих отечественных журналах (К1/К2 Списка ВАК): 12.
- Публикации в ведущих зарубежных журналах (Q1-Q3 по Scopus/WoS, приравненные к К1 Списка ВАК): 3.
- Публикации в сборниках трудов конференций, имеющих рейтинг A/A* по CORE: 6.
- Публикации в сборниках трудов ведущих конференций, индексируемые в международных базах данных (Scopus/WoS): 20.
- Русскоязычные публикации в сборниках трудов Российских профильных конференций, индексируемые в РИНЦ: 13.

Личный вклад автора. Содержание диссертационной работы в полной мере отражает личный вклад автора в проведенное исследование. Автор принимал непосредственное участие в получении представленных в диссертации результатов, включая разработку и теоретическое исследование предлагаемых методов и алгоритмов, а также их экспериментальную апробацию. Подготовка к публикации полученных результатов проводилась совместно с соавторами, причем вклад диссертанта был определяющим.

Соответствие паспорту специальности. Диссертация и полученные в ней результаты соответствуют паспорту специальности 1.2.1. – Искусственный интеллект и машинное обучение, а именно: пункту 16 “Исследования в области специальных методов оптимизации, проблем сложность и элиминации перебора, снижения размерности”, пункту 6 в части “Разработка систем управления с

использованием систем искусственного интеллекта и методов машинного обучения в том числе – управления роботами, автомобилями, БПЛА и т.п.", пункту 5 в части "Исследования в области совместного применения методов машинного обучения и классического математического моделирования" и пункту 2 в части "Исследования в области оценки качества и эффективности алгоритмических и программных решений для систем искусственного интеллекта и машинного обучения".

Основные положения, выносимые на защиту:

1. Метод поиска пути на динамических графах регулярной декомпозиции, и алгоритм его реализующий, допускающий возможность перемещения между произвольными вершинами графа и гарантирующий полноту и оптимальность отыскиваемых решений. Оценка сложности указанного алгоритма.
2. Метод поиска субоптимального решения задачи построения пути на динамическом графе регулярной декомпозиции и алгоритм его реализующий, использующий технику переназначения родителя в дереве поиска для повышения вычислительной эффективности. Теоретическое обоснование предоставляемых алгоритмом гарантий качества отыскиваемых решений.
3. Метод и алгоритм конфликтно-ориентированного планирования для построения совокупности неконфликтных путей на графе регулярной декомпозиции, допускающий возможность перемещения между произвольными вершинами графа, гарантирующий оптимальность отыскиваемых решений и использующий оригинальные техники сокращения перебора для повышения вычислительной эффективности.
4. Метод и алгоритм приоритизированного планирования для построения совокупности неконфликтных путей на графе регулярной декомпозиции, направленный на эффективный поиск субоптимальных решений и допускающий возможность перемещения между произвольными вершинами графа.
5. Семейство алгоритмов поиска пути на графе регулярной декомпозиции, учитывающих геометрические ограничения, и гарантирующее отыскание оптимальных решений в определенном классе.
6. Оригинальные эвристические функции для решения задачи поиска пути на графе регулярной декомпозиции и способы их интеграции

с алгоритмами систематического поиска. Способы автоматического построения (обучения) указанных эвристических функций на основе глубоких нейронных сетей.

7. Гибридные алгоритмы поиска пути на графе регулярной декомпозиции, использующие предлагаемые обучаемые эвристики и техники систематического поиска, гарантирующие корректность отыскиваемых решений вне зависимости от выходных значений искусственной нейронной сети.

Объем и структура работы. Диссертация состоит из введения, 5 глав, заключения. Полный объём диссертации составляет 350 страниц, включая 108 рисунков и 22 таблицы. Список литературы содержит 253 наименования.

Глава 1. Введение в предметную область

Работа посвящена методам поиска пути на графах, моделирующих окружающее пространство мобильного робота. Базовой задачей является поиск (кратчайшего) пути на графе особого типа – регулярном графе, вложенном в метрическое пространство и являющимся частичном подграфом графа-решётки. Базовой техникой решения задачи является систематический поиск, опирающийся на заданную (или автоматически конструируемую) эвристическую функцию – эвристический поиск. Соответственно, в данной главе приводятся основные используемые определения и термины, используемые в дальнейшем на протяжении всей работы; описывается базовая постановка задачи планирования траектории как задачи поиска пути на графе регулярной декомпозиции; описываются известные базовые техники и алгоритмы эвристического поиска, на основе которых впоследствии строятся новые методы и алгоритмы. Приводится анализ научно-технической литературы в области современных систем управления мобильными роботами, методов и алгоритмов планирования траектории, в т.ч. методов систематического, эвристического поиска.

1.1 Основные определения и базовые формулировки

Графы, вложенные в метрическое пространство. Рассмотрим пару:

$$(\mathcal{M}, \mathcal{G}), \tag{1.1}$$

где $\mathcal{M}$ – метрическое пространство, с заданной метрикой $\|\cdot\|$, а

$$\mathcal{G} = (V, E, w) \tag{1.2}$$

– неориентированный взвешенный граф, задаваемый конечным множеством вершин $V = \{v_1, ..., v_n\}$, конечным множеством ребер $E = \{e \mid e = (u, v) = (v, u),\ u \in V, v \in V,\ u \neq v\}$ и функцией весов (стоимостей) ребер $w : E \to \mathbb{R}^+$, $(\mathbb{R}^+ \equiv \{x \in \mathbb{R} \mid x > 0\})$.

Определение 1. Будем говорить, что граф $\mathcal{G}$ *вложен* в метрическое пространство $\mathcal{M}$, если задано отображение

$$coord : V \to \mathcal{M}, \tag{1.3}$$

т.е. каждой вершине графа соответствует элемент $\mathcal{M}$, при этом $w(e) = \|coord(u), coord(v)\|$, для $e = (u, v)$ т.е. вес ребра равен расстоянию между соответствующими точками в $\mathcal{M}$.

Здесь и далее в работе будем рассматривать только графы, вложенные в метрическое пространство. При этом по умолчанию будем предполагать, что $\mathcal{M} = \mathbb{R}^2$, т.е. граф вложен в двумерное пространство с Евклидовой метрикой и Декартовой системой координат (двумерное Евклидово пространство). Ребро графа $e = (u, v)$ соответствует отрезку AB, где $A = coord(v)$, $B = coord(u)$. Соответственно, вес этого ребра, $w(e)$, есть длина отрезка AB. Заметим, что эти допущения не являются существенными с точки зрения предлагаемых методов и алгоритмов[1], они вводятся для простоты изложения и наглядности иллюстраций.

Пример графа, вложенного в $\mathbb{R}^2$, показан на Рис. 1.1.

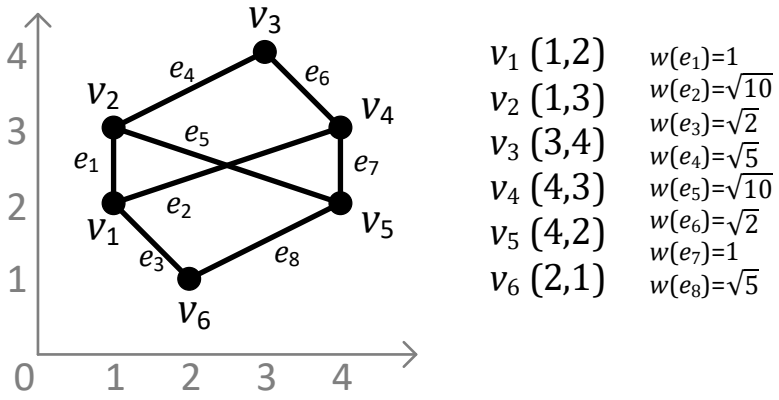


Рисунок 1.1 — Пример графа, вложенного в $\mathbb{R}^2$.

Изображенный граф содержит 6 вершин и 8 ребер. Координаты вершин и веса ребер (длины соответствующих отрезков) приведены справа от графа.

Графы регулярной декомпозиции. Рассмотрим теперь частный случай введенного выше в рассмотрение графа, который наиболее часто встречается

[1]Предлагаемые в работе методы и алгоритмы вполне можно адаптировать и для, например, трехмерного случая, и для случая, когда ребрам соответствуют не отрезки прямых, а кривые заданные тем или иным способом и так далее.

на практике в контексте задачи планирования траектории (траекторий) для мобильных колесных роботов – *граф регулярной декомпозиции* (ГРД).

Определение 2. ГРД – это граф, вложенный в $\mathbb{R}^2$, и содержащий вершины, которым соответствуют точки на плоскости, образующие регулярную сетку, т.е.:

$$\begin{aligned} &\forall v, u \in V : \\ &coord_x(u) = coord_x(v) + k \cdot \Delta \\ &coord_y(u) = coord_y(v) + k \cdot \Delta, \end{aligned} \tag{1.4}$$

где $k \in \mathbb{N} \cup 0$, $\Delta = const > 0$, а запись $coord_x(v)$ используется для обозначения X-координаты вершины v ($coord_y(v)$ – аналогично).

Для упрощения обозначения X-, Y-координат вершин здесь и далее будем использовать запись v_x, v_y, соответственно.

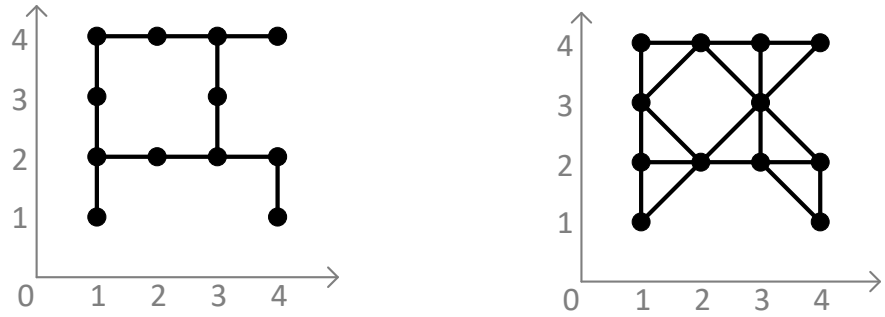
Параметр Δ, определяющий то, насколько вершины ГРД отстоят друг от друга в пространстве, будем называть шагом дискретизации, шагом сетки, или просто – шагом.

Ребра ГРД задаются следующим образом. Если $|v_x - u_x| = \Delta \wedge v_y = u_y$, то $(u,v) \in E$. Если $|v_y - u_y| = \Delta \wedge v_x = u_x$, то $(u,v) \in E$. То есть, ребром связываются две вершины, X (Y) координаты которых совпадают, а Y (X) координаты отличаются на Δ (т.е. между ними нет других вершин). Будем называть такие ребра ортогональными (т.е. это ребра, связывающие ортогонально смежные вершины). Поскольку максимальная степень вершины при таком способе задания ребер равна 4, такой ГРД называют 4-связным.

Аналогичным образом можно ввести в рассмотрение и 8-связный ГРД. В нём, помимо ортогональных ребер добавляются диагональные, т.е. такие которые связывают вершину с ее с ближайшими по диагонали соседями. Формально это условие связности можно сформулировать так. Если $|v_x - u_x| = \Delta \wedge |v_y - u_y| = \Delta$, то $(u,v) \in E$.

Пример 4-связного и 8-связного ГРД изображен на Рис. 1.2. В данном случае $\Delta = 1$.

Замечание. Введённое определение ГРД весьма схоже с более употребляемым в области теории графов определением т.н. решетчатого графа (графа-решетки), который имеет вершины в точках плоскости с неотрицательными целыми координатами. Ключевое отличие состоит в том, что ГРД является не графом-решёткой, а его частичным подграфом, в котором отсутствуют некоторые



а) 4-связный ГРД б) 8-связный ГРД

Рисунок 1.2 — Пример графов регулярной декомпозиции.

вершины. Это необходимо для моделирования препятствий в рабочей области некоторого мобильного робота, как будет показано далее. По этой причине в работе используется авторский термин – граф регулярной декомпозиции.

Для того, чтобы описать связь ГРД с задачей планирования пути (траектории) некоторого мобильного робота, формализуем сначала последнюю.

Задача планирования траектории. Рассмотрим некоторого колесного мобильного робота на плоскости. Будем ссылаться на такого робота в том числе как на мобильного агента. Пусть его рабочее пространство (т.е. то пространство, в котором он перемещается) является компактным подмножеством $\mathbb{R}^2$. Обозначим его за $\mathcal{W}$. Будем считать, что рабочее пространство состоит из проходимых и непроходимых областей: $\mathcal{W}_{free}$ и $\mathcal{W}_{obs}$, соответственно. При этом $\mathcal{W}_{free}$ замкнуто, а $\mathcal{W}_{obs} = \mathcal{W} \setminus \mathcal{W}_{free}$.

Конфигурацией агента или состоянием будем называть вектор $\mathbf{x} = (x, y, ...)$, где первые две компоненты задают положение (реперной точки) агента в рабочем пространстве, а другие компоненты используются для задания остальных характеристик агента, если требуется. В качестве примера таких дополнительных характеристик можно назвать ориентацию агента, его скорость и др.

Координатами состояния $\mathbf{x} = (x, y, ...)$ будем называть вектор (x,y) и обозначать его как $coord(\mathbf{x})$.

Множество всех состояний агента обозначим как C.

Далее введем в рассмотрение функцию:

$$shp : C \to 2^{\mathcal{W}}, \tag{1.5}$$

которая каждому состоянию ставит в соответствие область рабочего пространства, занимаемую агентом при нахождении в этом состоянии.

В англоязычной литературе для обозначения такого рода функции часто используется термин *footprint*, то есть "отпечаток" (проекция формы) агента на рабочее пространство. Например, для агента имеющего форму диска, *shp* возвращает все точки рабочего пространства, лежащие от центра диска на расстояние меньшем либо равном r, где r – радиус диска.

Определение 3. Множество допустимых состояний для агента – это множество вида $C_{free} = \{\mathbf{x} \in C \mid shp(\mathbf{x}) \cap \mathcal{W}_{obs} = \varnothing)\}$.

Неформально, это множество таких состояний, находясь в которых агент не сталкивается с препятствиями.

Рассмотрим два состояния, $\mathbf{x}_{start} \in C_{free}$ и $\mathbf{x}_{goal} \in C_{free}$ и введем понятие пути (или же – траектории[2]).

Определение 4. Путь (траектория) из $\mathbf{x}_{start} \in C_{free}$ в $\mathbf{x}_{goal} \in C_{free}$ – это отображение вида:

$$\begin{aligned} &\pi : [0,1] \to C, \\ &\text{т.ч.} \\ &\pi(0) = \mathbf{x}_{start}, \pi(1) = \mathbf{x}_{goal}. \end{aligned} \tag{1.6}$$

Особый интерес представляют такие траектории, при следовании вдоль которых агент не сталкивается с препятствиями, что может быть формализовано следующим образом.

Определение 5. Допустимый путь (траектория) из $\mathbf{x}_{start} \in C_{free}$ в $\mathbf{x}_{goal} \in C_{free}$ – это отображение $\pi : [0,1] \to C$, т.ч. $\pi(0) = \mathbf{x}_{start}$, $\pi(1) = \mathbf{x}_{goal}$ и выполняется следующее условие:

$$\forall \alpha \in [0,1] : \ \pi(\alpha) \in C_{free}. \tag{1.7}$$

Примеры различных путей показаны на Рис. 1.3.

[2]В литературе по планированию иногда разделяют термины *путь* (англ. *path*) и *траектория* (англ. *trajectory*). В данной работе мы будем использовать оба термина.

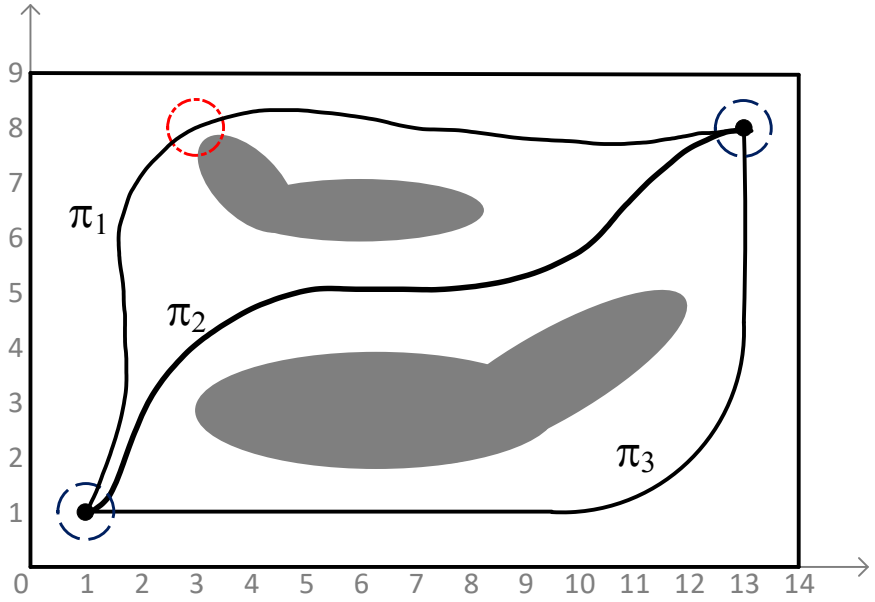


Рисунок 1.3 — Примеры путей (траекторий) в рабочем пространстве мобильного агента.

На рисунке черной рамкой отмечено рабочее пространство агента (прямоугольник на плоскости размером 14×9). Область, соответствующая множеству допустимых состояний, не имеет заливки, в то время как препятствия отмечены темно-серым. Предполагается, что агент моделируется диском диаметра 1. В этом случае траектория π_1 является недопустимой (т.к. при следовании по ней агент задевает верхнее препятствие), в то время как π_2 и π_3 допустимы.

Введем теперь формальное определение задачи планирования пути (траектории). Для обозначения этой задачи будем использовать запись PF, от англ. path finding (поиск пути).

Определение 6. Задача планирования пути – это пятерка:

$$\text{PF} = (\mathcal{W}, shp, C_{free}, \mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}). \tag{1.8}$$

По фиксированным начальному и целевому состояниям агента: $\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}$ (и описанию рабочего пространства, $\mathcal{W}$, множества допустимых состояний, C_{free}, заданной функции проекции формы агента, shp) необходимо найти (построить) допустимую траекторию $\pi(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}})$, связывающую эти состояния.

На практике, при решении задачи планирования, очевидно, возникает необходимость оценки качества траектории. Для этого вводится понятие *стоимости* траектории.

Определение 7. Пусть $\Pi = \{\pi_1, \pi_2, ...\}$ – множество траекторий, связывающих заданные начальное и целевое состояния. Тогда стоимость траектории – это скалярная функция:

$$cost : \Pi \to \mathbb{R}^+. \tag{1.9}$$

Заметим, что в литературе по планированию встречаются работы, в которых $cost(\pi)$ есть вектор-функция (см., например [82]), однако в данной работе рассматриваются лишь скалярные функции стоимости.

Обозначим за $\Pi^+ \subseteq \Pi$ множество допустимых траекторий (для некоторых фиксированных начального и целевого состояний). Тогда с использованием понятия стоимости можно дать определение оптимального решения задачи планирования.

Определение 8. Допустимая траектория $\pi^* \in \Pi^+$ является оптимальным решением задачи планирования, если:

$$\begin{aligned} &\forall \pi \in \Pi^+ : \\ &cost(\pi^*) \leqslant cost(\pi). \end{aligned} \tag{1.10}$$

То есть не существует другой допустимой траектории с более низким значением функции стоимости.

В качестве очевидного примера функции стоимости можно привести длину траектории, т.е. чем длина траектории меньше, тем лучше. Известны функции стоимости учитывающие кривизну траектории и другие характеристики [83].

Очевидно, что задача поиска оптимальной допустимой траектории может быть формально записана и как задача оптимизации:

$$\begin{aligned} &cost(\pi) \to \min \\ &\text{т.ч.:} \\ &\pi(0) = \mathbf{x}_{\text{start}}, \\ &\pi(1) = \mathbf{x}_{\text{goal}}, \\ &\forall \alpha \in [0,1] : \ \pi(\alpha) \in C_{free}. \end{aligned} \tag{1.11}$$

Формализовав задачу планирования траектории для некоторого агента, опишем теперь связь определенного типа этой задачи с задачей поиска пути на ГРД.

1.2 Геометрическое планирование и поиск пути на графах регулярной декомпозиции

Приведенные ранее в работе определения пути (траектории) не накладывают ограничений на вид этого пути. При этом одним из классов путей, отыскание которых может быть полезно для решения многих практических задач, являются пути, представляющие собой ломаные линии (последовательности отрезков), соединяющие начальное и целевое положения агента. Будем называть такие пути *геометрическими путями*, а задачу подразумевающую их построение – задачей *геометрического планирования*.

Формально геометрический путь определяется следующим образом.

Определение 9. Геометрический путь из $\mathbf{x}_{start} \in C_{free}$ в $\mathbf{x}_{goal} \in C_{free}$ – это отображение:

$$
\begin{aligned}
&\pi : [\alpha_0, \alpha_1, \alpha_2, ..., \alpha_{n-1}, \alpha_n] \to C, \\
&\text{т.ч.:} \\
&0 = \alpha_0 < \alpha_1 < ... < \alpha_{n-1} < \alpha_n = 1, \\
&\pi(0) = \pi(\alpha_0) = \mathbf{x}_{start},\ \pi(1) = \pi(\alpha_n) = \mathbf{x}_{goal}, \\
&\forall i \in \{0, 1, ..., n-1\}, \forall \alpha \in [\alpha_i, \alpha_{i+1}] : \\
&\qquad coord(\pi(\alpha)) = (1-\alpha) \cdot coord(\pi(\alpha_i)) + \alpha \cdot coord(\pi(\alpha_{i+1})).
\end{aligned}
\tag{1.12}
$$

Геометрическое планирования востребовано, например, в мобильной робототехнике в задачах автономной прокладки маршрутов для колесных роботов, которые могут двигаться в произвольном направлении без необходимости каких-либо поворотов (это достигается специальной конструкцией колес, которые обычно называются омни-колесами). Такой тип планирования весьма востребован и для колесных роботов с дифференциальным приводом, т.е. роботов, которые могут менять ориентацию без продольного движения. Такие роботы поворачиваются на месте путем вращения колес разных осей в разные стороны. Указанные робототехнические системы на практике могут достаточно точно следовать вдоль путей, представляющих собой ломаные линии, поэтому геометрическое планирование для них весьма актуально.

Будем считать, что решается задача геометрического планирования для дискообразного колесного робота с дифференциальным электроприводом или

на омни-колесах, состояние которого описывается его координатами на плоскости: $\mathbf{x} = (x, y)$. При этом $r = const > 0$ – это радиус робота (плюс, возможно, некоторый "запас безопасности", добавляемый из практических соображений, чтобы робот не маневрировал очень близко к препятствиями). Заметим здесь, что даже если рассматривается робот не круглой формы, то всегда можно перейти к постановке с роботом в форме диска, радиус которого определяется как минимально возможный радиус окружности, за пределы которой не выходит ни одна точка робота при повороте на месте.

Формально будем считать, что функция проекции формы робота на рабочее пространство задана следующий образом:

$$\forall \mathbf{x} = (x, y) \in C: \ shp(\mathbf{x}) = \{(x',y') \in \mathcal{W} | \ \sqrt{(x' - x)^2 + (y' - y)^2} < r\}. \quad (1.13)$$

Текущее (начальное) состояние робота обозначим как $\mathbf{x}_{\text{start}} = (x_{start}, y_{start})$, а целевое (в которое ему необходимо переместиться) как $\mathbf{x}_{\text{goal}} = (x_{goal}, y_{goal})$.

Зафиксируем некоторую константу $\Delta = const > 0$. Рассмотрим теперь 4(8)-связный ГРД $\mathcal{G}$ с шагом сетки Δ, содержащий вершину $v_{start} = (x_{start}, y_{start})$, и вершины, координаты которых определены в соответствии с формулой 1.4 и принадлежат рабочему пространству робота $\mathcal{W}$. Без ограничения общности будем считать, что ГРД содержит вершину v_{goal}, координаты которой равны (x_{goal}, y_{goal}), т.е. целевую вершину[3].

Будем ассоциировать с каждой вершиной $v \in \mathcal{G}$ область $\mathcal{W}$, имеющую форму квадрата с центром в (v_x, v_y) и длиной стороны Δ. Назовем эту область клеткой (англ. cell).

Определение 10. Клетка, ассоциированная с вершиной v ГРД, – это множество точек рабочего пространства, определяемое по формуле:

$$cell(v) = \{(x, y) \in \mathcal{W} \mid |v_x - x| \leqslant \frac{\Delta}{2} \wedge |v_y - y| \leqslant \frac{\Delta}{2}\} \quad (1.14)$$

Определение 11. Клетка $cell(v)$ является проходимой (свободной, незаблокированной), если выполняется соотношение:

[3]В общем случае, целевой можно считать ту вершину, которая ближе других расположена к целевому положению робота (в случае, если несколько вершин расположены на одинаковом удалении – целевая из них выбирается произвольным образом)

$$cell(v) \cap \mathcal{W}_{obs} = \varnothing. \tag{1.15}$$

В противном случае клетка является заблокированной.

Обычно ГРД изображается именно в виде совокупности клеток, при этом заблокированные клетки помечаются штрихом или заливкой – см. Рис. 1.4.

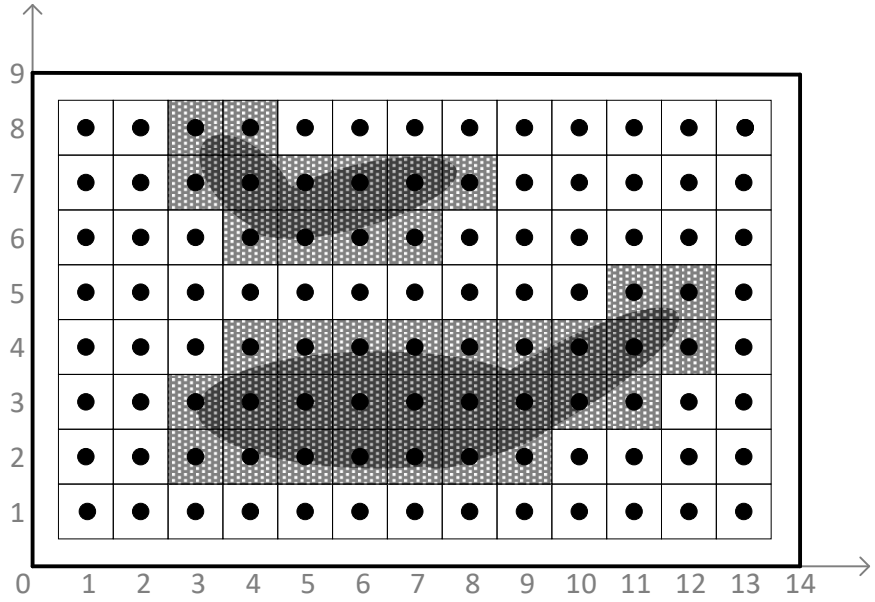


Рисунок 1.4 — Рабочее пространство мобильного агента и вершины соответствующего ГРД (ребра не изображены для облегчения восприятия).

Область рабочего пространства, покрываемую свободными клетками обозначим как $\mathcal{W}_{g\_free}$ (здесь следует обратить внимание на букву ‘g’ – первую букву нижнего индекса, которая отличает это обозначение от имеющегося ранее $\mathcal{W}_{free}$). Формально:

$$\mathcal{W}_{g\_free} = \{(x, y) \in \mathcal{M} \mid \exists v \in V_{free} : (x, y) \in cell(v)\}. \tag{1.16}$$

Здесь V_{free} – это множество всех проходимых клеток ГРД.

Область рабочего пространства, не покрываемую проходимыми клетками (т.е. покрываемую непроходимыми), обозначим как $\mathcal{W}_{g\_obs} = \mathcal{W} \setminus \mathcal{W}_{g\_free}$. Заметим, что в соответствии с введенными в работе определениями границы клеток входят в $\mathcal{W}_{g\_free}$, но не в $\mathcal{W}_{g\_obs}$.

Очевидно, что в общем случае множества $\mathcal{W}_{free}$ и $\mathcal{W}_{obs}$ не совпадают с $\mathcal{W}_{g\_free}$ и $\mathcal{W}_{g\_obs}$ соответственно – см. Рис. 1.5. При этом, очевидно, что чем меньше Δ, тем лучше $\mathcal{W}_{g\_free}$ и $\mathcal{W}_{g\_obs}$ аппроксимируют $\mathcal{W}_{free}$ и $\mathcal{W}_{obs}$.

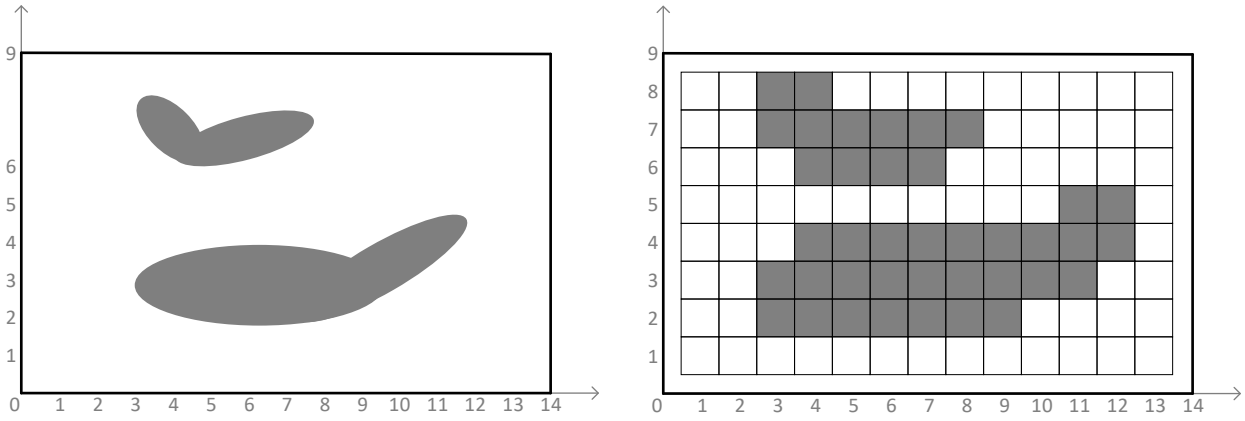


Рисунок 1.5 — Непроходимые области рабочего пространства (слева) и их аппроксимация в виде совокупности непроходимых клеток ГРД (справа).

Помимо понятия проходимости клетки, ассоциированной с некоторой вершиной ГРД v, введем понятие допустимости вершины v.

Определение 12. Вершина ГРД v является допустимой, если выполняется соотношение:

$$shp(coord(v)) \cap \mathcal{W}_{g\_obs} = \varnothing. \tag{1.17}$$

В отличие от проходимости клетки, допустимость вершины учитывает форму и размер агента (посредством функции проекции shp). Стоить заметить, что проходимость клетки определяется через $\mathcal{W}_{obs}$, а допустимость вершины – через $\mathcal{W}_{g\_obs}$, т.е. клеточную аппроксимацию $\mathcal{W}_{obs}$.

Неформально, проходимость клетки говорит лишь о том, что в некоторая окрестность вершины ГРД, ассоциированной с этой клеткой, не содержит препятствий; а допустимость вершины говорит о том, что агент может находиться в ней без столкновения с препятствиями, вернее – их клеточными аппроксимациями.

Очевидно, что в зависимости от радиуса безопасности агента r одна и та же проходимая клетка ГРД может быть или не быть допустимой – см. Рис. 1.6.

На рисунке изображен ГРД с единичным шагом сетки (т.е. $\Delta = 1$), аппроксимирующий рабочее пространство, содержащее одно прямоугольное препятствие: оно выделено темно-серой заливкой, в то время как непроходимые клетки ГРД – светло-серой. Клетка с координатами (2,4) проходима. При этом при $r = 0.5$ эта клетка является допустимой, а при $r = 1$ – нет. Заметим, что в последнем случае клетка (2,4) является недопустимой несмотря на то,

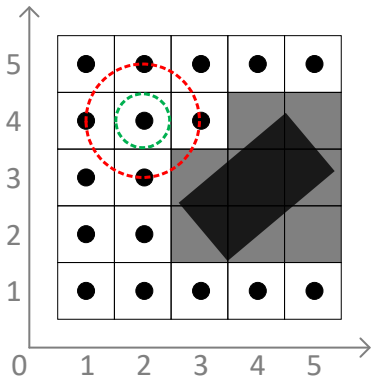


Рисунок 1.6 — Пример, иллюстрирующий понятие допустимости клетки ГРД.

что агент единичного радиуса, находясь в ней, не пересекается с препятствием (но пересекается с его клеточной аппроксимацией, относительно которой и определяется допустимость и недопустимость клетки).

Очевидно, что если шаг сетки ГРД больше либо равен диаметру агента, т.е. $\Delta \geqslant 2 \cdot r$, то любая проходимая клетка допустима.

Введем в рассмотрение понятия проходимости и допустимости ребра.

Определение 13. Ребро ГРД $e = (u, v)$ является проходимым, если вершины u и v проходимы.

Определение 14. Ребро ГРД $e = (u, v)$ является допустимым, если:

$$\forall (x,y) \in [u,v] : \ shp(x, y) \cap \mathcal{W}_{g\_obs} = \varnothing, \tag{1.18}$$

где $[u, v]$ – отрезок прямой, соединяющий точки (u_x,u_y) и (v_x,v_y).

Разница в понятиях допустимости и проходимости ребра ГРД аналогична разнице в тех же понятиях для вершин ГРД. Проходимость ребра никак не учитывает форму и размер агента и по сути является лишь декларацией того, что в некоторой окрестности вершин, образующих ребро, не содержится препятствий. Допустимость же ребра учитывает форму и размер агента и наличие этого свойства говорит о том, что агент может переместиться вдоль отрезка, задающего ребро, без столкновений с препятствиями (вернее – с их клеточными аппроксимациями). Допустимость и проходимость ребра являются эквивалентными понятиями лишь для случая, когда $r = 0$, т.е. при рассмотрении точечного агента.

Сформулируем теперь следующее очевидное утверждение, фиксирующее взаимосвязь задачи поиска пути на ГРД и задачи геометрического планирования.

Утверждение 1. *Путь на ГРД, первой вершиной которого является v_{start}, последней – v_{goal}, состоящий из допустимых ребер, является решением задачи геометрического планирования.*

Заметим, что обратное, в общем случае, неверно. Решение задачи планирования согласно Определению 5 может существовать, при этом соответствующего пути в графе может не быть. Это имеет место, когда шаг сетки Δ, определяющий то, насколько вершины ГРД, удалены друг от друга в пространстве, выбран достаточно большим. Ведь, чем выше значение Δ, тем хуже ГРД аппроксимирует связность проходимого пространства и, как следствие, некоторые проходимые области рабочего пространства могут разбиваться на несвязные подобласти.

В данной работе, вопрос выбора значения Δ при определении ГРД не рассматривается и считается, что ГРД построен с подходящим для задачи планирования значением Δ (на практике – это инженерная задача разработчика системы управления мобильным роботом).

Дополнительно предполагается, что шаг сетки Δ выбран так, что $\Delta \geqslant 2 \cdot r$, где $r = const > 0$ – это радиус диска, моделирующего робота. Такой выбор значения Δ облегчает техническую реализацию процедуры проверки вершин и ребер графа на допустимость.

Действительно в этом случае располагаясь в любой проходимой вершине ГРД агент не сталкивается с препятствиями. То есь любая проходимая вершина является допустимой. В этом же время допустимость любого ортогонального ребра (u, v) определяется лишь проходимостью клеток $cell(u)$, $cell(v)$, а именно – ортогональное ребро (u, v) допустимо, тогда и только тогда когда клетки $cell(u)$, $cell(v)$ проходимы.

Проверка на допустимость диагонального ребра сводится к проверке на проходимость нескольких вершин. Более точно – диагональное ребро $e = (u, v)$ допустимо, тогда и только тогда когда проходимы клетки, ассоциированные со следующими вершинами: $u = (u_x, u_y)$, $v = (v_x, v_y)$, $u' = (u_x + dx, u_y)$, $u'' = (u_x, u_y + dy)$, где $dx = v_x - u_x$, $dy = v_y - u_y$.

Примеры допустимых ребер 4- и 8-связных ГРД, аппроксимирующих рабочее пространство с шагом сетки $\Delta = 1$, для $r = 1/2$ показаны на Рис. 1.7.

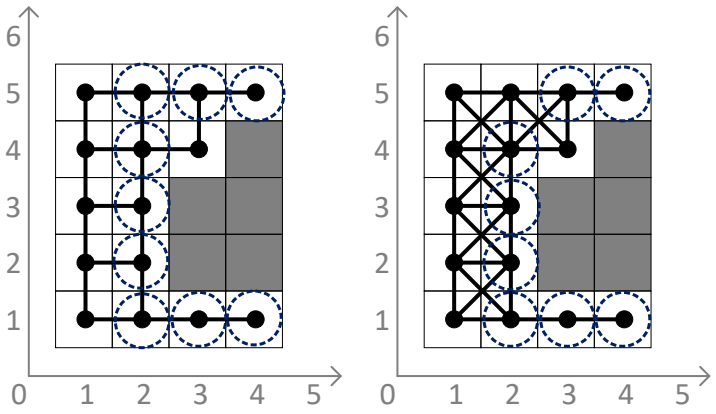


Рисунок 1.7 — Допустимые ребра для 4- и 8-связных ГРД при $r = 1$.

На рисунке изображены пути на ГРД из вершины с координатами (4,1) в вершину с координатами (4,5). Оба изображенных пути включат лишь допустимые ребра. Следует обратить внимание, что путь на 8-связном ГРД не включает, например, диагональное ребро (u,v), где $u = (u_x,u_y) = (3,1)$, $v = (v_x,v_y) = (2,2)$, т.к. это ребро (не изображенное на рисунке) не является допустимым. Действительно, в данном случае $dy = v_y - u_y = 2 - 1 = 1$ и вершина $u'' = (u_x, u_y + dy) = (3,1 + 1) = (3,2)$ является непроходимой.

Завершим изложение этого параграфа комментарием, касающимся случая когда шаг сетки ГРД не удовлетворяет соотношению $\Delta \geqslant 2 \cdot r$, то есть $\Delta < 2 \cdot r$. Основное отличие этого случая состоит в том, что теперь агент, находясь в вершине $v \in V$, накрывает и ряд смежных клеток (т.к. сторона клетки меньше диаметра агента). То есть понятия проходимости и допустимости клеток не эквивалентны. С практической точки зрения это означает, что теперь для определения того является ли клетка ГРД допустимой необходима отдельная процедура идентификации клеток, накрываемых агентом, находящимся в вершине v. Тоже самое касается и определения допустимости ребра – необходима отдельная процедура, идентифицирующая все накрываемые агентом клетки при движении вдоль ребра. Реализация обоих этих процедур не представляет труда и носит технический характер, см. например [84].

1.3 Алгоритмы эвристического поиска

В предыдущих параграфах были введены в рассмотрение графы особой структуры, вложенные в метрическое пространство, – ГРД, решение задачи поиска (кратчайшего) пути на которых может быть весьма полезным для автоматического планирования траектории некоторого агента (например, мобильного колесного робота), функционирующего в сложно-структурированном пространстве с препятствиями. Опишем теперь один из наиболее распространенных способов решения задачи поиска путей ГРД, а именно – алгоритм эвристического поиска $\mathtt{A}^*$ [11].

$\mathtt{A}^*$ – это итеративный алгоритм, осуществляющий поиск в пространстве состояний, индуцируемом вершинами исходного графа. В процессе этого поиска строится так называемое дерево поиска, корнем которого является состояние, соответствующее начальной вершине.

Каждой вершине графа соответствует единственное состояние поиска (при этом для некоторых вершин состояний поиска может и не быть). Запишем это как $n = v$, где n – это состояние поиска, v – вершина графа.

Для каждого состояния алгоритм поиска $\mathtt{A}^*$ хранит в памяти следующие величины:

- $g(n)$ (g-значение состояния) – наименьший вес пути из начального состояния в состояние n, известный к текущей итерации алгоритма поиска;
- $h(n)$ (h-значение состояния) – эвристическая оценка веса пути из состояния n в целевое состояние;
- $f(n)$ (f-значение состояния) – величина, рассчитываемая по формуле $f(n) = g(n) + h(n)$, т.е. оценка веса пути из начального состояние в целевое через n;
- $parent(n)$ – указатель на состояние, предшествующее состоянию n в дереве поиска.

Основной концептуальной операцией алгоритма $\mathtt{A}^*$ является раскрытие состояния (англ. expansion). В ходе раскрытия состояния n сначала в памяти создаются новые состояния, достижимые из n. Эти новые состояния соответствуют смежным вершинам графа. Затем для каждого из этих определяется g-значение по формуле:

$$g(n') = g(n) + cost(n,n'), \tag{1.19}$$

где n' – это состояние, непосредственно достижимое из n, а $cost(n, n')$ – стоимость перехода из n в n'. В рассматриваемом случае она равна $w(e), e = (v, v')$, где v, v' – это вершины графа, которым соответствуют состояния n, n'.

После расчета $g(n')$ производится проверка – не было ли такое состояние уже рассмотрено (создано в ходе раскрытия другого состояния) поиском ранее. Если да, и новое g-значение больше либо равно (не лучше) ранее рассчитанного, то вновь созданное состояние отбрасывается. Если да, и новое g-значение меньше (лучше) ранее рассчитанного, то старое состояние удаляется, а новое сохраняется. Если же состояние n' создано впервые, то оно сохраняется вне зависимости от присвоенного g-значения. В последних двух случаях выполняется присваивание $parent(n') = n$, т.е. состояние n становится родительским для состояния n'.

Основной смысл процедуры раскрытия состояния можно описать следующим образом. На основе знания о стоимости достижения некоторого состояния поиска n пересчитывается (или рассчитывается впервые) стоимость достижения непосредственно смежных состояний n' – состояний-последователей.

Нетрудно заметить, что при таком подходе у каждого созданного состояния в процессе поиска существует лишь один родитель. То есть алгоритм порождает дерево поиска. Для упорядочивания процесса раскрытий это дерево разделяется на две части, обычно именуемые списками: открытие состояния и закрытые состояния, OPEN и CLOSED соответственно. Список CLOSED содержит состояния, которые уже были раскрыты и не являются кандидатами на дальнейшее раскрытие, список OPEN содержит состояния, являющиеся кандидатами на дальнейшее раскрытие. В общем случае некоторое состояние может быть раскрыто несколько раз, т.е. перемещения состояния из OPEN в CLOSED, затем опять в OPEN, затем опять в CLOSED и так далее – возможны.

Поскольку в дальнейшей работе алгоритм A* будет тем или иным образом использоваться и/или модифицироваться для решения более сложных задач планирования траектории(й), приведём его псевдокод – см. Рис 1.8.

На этапе инициализации (строки 2–4) создается начальное состояние, которое помещается в список OPEN. Это – корень дерева поиска. Список CLOSED – пуст. Затем на каждой итерации основного цикла (строки 5–25) в списке OPEN определяется состояние с минимальным f-значением (строка 6), извлекается (удаляется) из OPEN и помещается в список CLOSED (строки 7–8). Если это состояние является целевым, т.е. оно соответствует целевой вершине графа,

```
1  Алгоритм A*(𝒢, v_start, v_goal, h):
       Входные данные: Граф 𝒢, начальная вершина v_start, целевая
                       вершина v_goal, эвристическая функция h
       Выходные данные: Путь π
2      n_start ← GenerateSearchNode(v_start)
3      g(n_start) ← 0; parent(n_start) ← null
4      OPEN ← {n_start}; CLOSED ← ∅
5      while OPEN ≠ ∅ do
6          n ← arg min_{n∈OPEN}(f(n) = g(n) + h(n))
7          OPEN ← OPEN \ {n}
8          CLOSED ← CLOSED ∪ {n}
9          if n.v = v_goal then
10             return ReconstructPath(n)
11         SUCC ← ∅
12         foreach v' ∈ GetAdjVertices(n.v, 𝒢) do
13             n' ← GenerateSearchNode(v')
14             g(n') ← g(n) + cost(n, n')
15             parent(n') ← n
16             SUCC ← SUCC ∪ {n'}
17         foreach n' ∈ SUCC do
18             n'' ← FindInOpenAndClosed(n')
19             if n'' = null then
20                 OPEN ← OPEN ∪ {n'}
21             else if g(n'') ⩽ g(n') then
22                 continue
23             else
24                 RemoveFromOpenClosed(n'')
25                 OPEN ← OPEN ∪ {n'}
26     return failure
```

Рисунок 1.8 — Псевдокод алгоритма эвристического поиска A*.

то искомый путь найден и может быть восстановлен с помощью родительских указателей (строки 9–10). В противном случае происходит раскрытие вершины (строки 11–25), в рамках которого сначала создаются состояния поиска, соответствующие смежным вершинам графа (строки 11–16). Для каждого вновь созданного состояния рассчитывается g-значение (строка 14) и присваивается родительских указатель (строка 15). Далее для каждого из этих состояний, n', проверятся (строки 17–25) есть ли такое же состояние в текущем дереве поиска (строка 18). Если нет, то вновь созданное состояние становится частью дерева поиска и добавляется в список OPEN. Если есть, и g-значение соответствующего состояния в дереве поиска, n'', меньше либо равно вновь рассчитанному g-значению, $g(n')$, то новое состояние отбрасывается (строки 21–22). Если есть, но старое g-значение хуже, то старое состояние поиска удаляется, а новое – добавляется в OPEN (строки 23–25). Если в процессе поиска список OPEN исчерпан (условие в строке 5 не выполняется) и при этом целевое состояние не было достигнуто (т.е. условие в строке 9 не было выполнено), то алгоритм возвращает *failure* (строка 26), что означает, что путь не удалось найти.

Свойства алгоритма $\mathtt{A}^*$ и его практическая вычислительная эффективность (число итераций при решении конкретной задачи поиска пути) во многом зависят от того, какая эвристическая функция используется при поиске. Обычно выделяют следующие типы эвристических функций.

Определение 15. *Допустимая* эвристическая функция h, это такая функция для которой выполняется неравенство:

$$\forall n : h(n) \leqslant \pi^*(n, n_{goal}), \tag{1.20}$$

где π^* – кратчайший путь на ГРД.

Как следует из определения, допустимая эвристическая функция не переоценивает вес пути до целевой вершины.

Определение 16. *Монотонная* эвристическая функция h, это такая функция для которой выполняется неравенство:

$$\forall n, n' : h(n') \leqslant h(n) + cost(\pi^*(n, n')). \tag{1.21}$$

Известно, что любая монотонная эвристика является допустимой (обратное в общем случае неверно) [13]. При этом доказано, что если используемая

алгоритмом поиска эвристическая функция является допустимой, то гарантируется, что найденное алгоритмом решение является оптимальным. Следовательно, использование на практике при поиске допустимой или монотонной эвристики гарантирует построение кратчайшего пути в графе. Более того, использование монотонной эвристики гарантирует, что ни одно состояние поиска не может быть раскрыто более, чем один раз [13]. Это обстоятельство позволяет несколько модифицировать (в сторону упрощения) логику работы алгоритма. Псевдокод алгоритма A* с монотонной эвристикой – A*-MH, – представлен на Рис. 1.9. Основное отличие этого алгоритма от стандартного A* (представленного ранее на Рис. 1.8) заключается в обработке состояний последователей при раскрытии вершины (строки, начиная с 17-й). При использовании монотонной эвристики осуществляется проверка на то, было ли созданное состояние-последователь раскрыто ранее. Для этого достаточно проверить, имеется ли оно в списке CLOSED (строка 18). Если это так, то состояние немедленно отбрасывается (строка 19), т.к. известно, что при использовании монотонной эвристики путь с более низкой стоимостью до любого раскрытого ранее состояния не может быть найден. Если же состояния-последователя нет в списке CLOSED, то теперь достаточно проверить его наличие лишь в списке OPEN и действовать сообразно: если состояния нет, то добавить его в OPEN (строки 21–22). Если же оно есть в OPEN, но с худшим g-значением, то старое состояние удаляется из OPEN, а новое (с меньшим g-значением) добавляется.

Обозначим процедуру создания состояний-последователей (строки 11–16) и процедуру модификации списков OPEN и CLOSED (строки 17–24) как `GenerateSuccesors` и `UpdateOpenClosed` соответственно. С использованием этих обозначений псевдокод алгоритма A*-MH может быть переписан в более компактном виде, представленном на Рис. 1.10. Будем ссылаться на такую версию алгоритма как на A*-C.

Все три приведенных псевдокода (Рис. 1.8, 1.9, 1.10) реализуют схожую логику работы. Отличие состоит лишь в том, какая эвристика передается на вход алгоритму (монотонная или нет) и в самой записи псевдокода (компактный вид или нет). Поэтому здесь и далее, будет ссылаться на любой из представленных вариантов как на алгоритм A*.

Важным вопросом при реализации алгоритма A* на практике является выбор структур данных (контейнеров) для хранения дерева поиска [85].

```
1  Алгоритм A*-MH(𝒢, v_start, v_goal, h):
       Входные данные: Граф 𝒢, начальная вершина v_start, целевая
                       вершина v_goal, монотонная эвристическая
                       функция h
       Выходные данные: Путь π
2      n_start ← GenerateSearchNode(v_start)
3      g(n_start) ← 0; parent(n_start) ← null
4      OPEN ← {n_start}; CLOSED ← ∅
5      while OPEN ≠ ∅ do
6          n ← arg min_{n∈OPEN}(g(n) + h(n))
7          OPEN ← OPEN \ {n}
8          CLOSED ← CLOSED ∪ {n}
9          if n.v = v_goal then
10             return ReconstructPath(n)
11         SUCC ← ∅
12         foreach v' ∈ GetAdjVertices(n.v, 𝒢) do
13             n' ← GenerateSearchNode(v')
14             g(n') ← g(n) + cost(n, n')
15             parent(n') ← n
16             SUCC ← SUCC ∪ {n'}
17         foreach n' ∈ SUCC do
18             if FindInClosed(n') then
19                 continue
20             n'' ← FindInOpen(n')
21             if n'' = null then
22                 OPEN ← OPEN ∪ {n'}
23             else if g(n'') > g(n') then
24                 OPEN ← (OPEN \ {n''}) ∪ {n'}
25     return failure
```

Рисунок 1.9 — Псевдокод алгоритма эвристического поиска A* с монотонной эвристикой.

```
1  Алгоритм A*-C(𝒢, v_start, v_goal,h):
       Входные данные: Граф 𝒢, начальная вершина v_start, целевая
                       вершина v_goal, монотонная эвристическая
                       функция h
       Выходные данные: Путь π
2      n_start ← GenerateSearchNode(v_start)
3      g(n_start) ← 0; parent(n_start) ← null
4      OPEN ← {n_start}; CLOSED ← ∅
5      while OPEN ≠ ∅ do
6          n ← arg min_{n∈OPEN}(g(n) + h(n))
7          OPEN ← OPEN \ {n}
8          CLOSED ← CLOSED ∪ {n}
9          if n.v = v_goal then
10             return ReconstructPath(n)
11         SUCC ← GenerateSuccesors(n.v, 𝒢)
12         UpdateOpenAndClosed(SUCC)
13     return failure
```

Рисунок 1.10 — Компактный псевдокод алгоритма эвристического поиска A* (с монотонной эвристикой).

Обычно для списка CLOSED используют неупорядоченные контейнеры с константным временем доступа, такие как, например, `unordered_set` или `unordered_map` в языке программирования C++. Уникальным идентификатором состояния обычно является пара целочисленных координат соответствующей вершины графа.

Для хранения списка OPEN логично использовать очередь с приоритетом (по f-значению), например, `priority_queue` в C++. Однако в таком случае вычислительно затратной становится проверка на наличие дубликатов в OPEN. Эту проверку на практике можно опустить, прибегнув к так называемой ленивой схеме обнаружения дубликатов. При ней новое состояние добавляется в список OPEN без проверки на наличие в нем такого же состояния. При извлечении же состояния из OPEN в начале очередной итерации поиска добавляется проверка – не раскрывалось ли такое состояние раньше. Если да, то это значит сейчас извлечен из списка дубликат с более высоким g-значением, и такой дуб-

ликат можно отбросить. Применение такого приема возможно только в случае, если используемая эвристическая функция является монотонной.

Заметим, что в общем случае в списке OPEN перед началом очередной итерации поиска может одновременно находиться множество состояний, f-значение которых минимально. Формально алгоритм $\mathtt{A}^*$ не регламентирует, какое именно из этих состояний должно быть извлечено из списка для раскрытия. Иными словами, допускается извлечение произвольного состояния с минимальными f-значением для сохранения всех теоретических гарантий алгоритма. На практике же число итераций алгоритма, а, следовательно, и время работы, может существенно отличаться в зависимости от реализованной стратегии выбора одного состояния из множества одинаковых с точки зрения f-значения. Подробнее этот вопрос исследуется в [86]. Обычно при поиске пути на графах, вложенных в метрическое пространство, когда достаточно легко определить монотонную эвристическую функцию, оценивающую длину пути (например, Евклидово расстояние, которое не учитывает наличие непроходимых областей), используется следующее правило: если f-значения двух или более состояний поиска совпадают, то предпочитать следуют то состояние, h-значение которого меньше. Если же и h-значения совпадают, то выбор среди таких состояний обычно осуществляется произвольно.

1.4 Обзор работ и результатов в предметной области

Выше была описана задача планирования траектории в общем виде и задача геометрического планирования, подразумевающая построение траектории в виде последовательности отрезков (секций), соединяющих заданные начальное и целевое положение. Была установлена связь этой задачи с задачей поиска пути на графе особой структуры, вложенном в метрическое пространство – графе регулярной декомпозиции. Придем далее обзор научного ландшафта, который обуславливает постановку таких задач, необходимость их решения и ограничения, которые целесообразно при этом учитывать.

Интеллектуальные системы управления Одной из наиболее активно-развивающихся областей науки и техники в настоящее время является робото-

техника. При этом фокус интереса исследователей и разработчиков в этой области смещается от теле-управляемых к автономным роботам, способным к выполнению широкого круга задач в сложноструктурированной динамической среде в полностью или частично автоматическом режиме. Создание таких роботов невозможно без разработки программных систем управления, функции которых не исчерпываются лишь стабилизацией объекта управления или обеспечения (точного) следования вдоль определенной траектории, но и позволяющих осуществлять комплексный анализ поступающей с бортовых сенсоров информации, прогнозировать изменения окружающей среды, строить адаптивные планы достижения целей миссии и обеспечивать выполнение этих планов (или их корректировку по мере необходимости). Такие системы управления принято называть интеллектуальными.

Традиционно системы управления в робототехнике строятся по модульному принципу, т.е. выделяются отдельные модули, реализующие различный функционал (планирование пути, распознавание объектов интереса, построение карты местности и др.) и взаимосвязи между ними [87; 88]. Например, очень часто выделяют такие модули как: восприятие (perception), прогнозирование (prediction), планирование (planning) и управление (control) [89]. Нередко системы управления являются многоуровневыми. Например, известно разделение на делиберативный и реактивный уровни управления [90]. На реактивном уровне решаются базовые задачи, связанные с выдерживанием заданных параметров объекта управления (скорость, угловое положение и т.д.). На делиберативном уровне осуществляется приоритизация целей, прогнозирование, планирование. В [91; 92] обоснована целесообразность разработки иерархических трехуровневых систем управления, в которых выделены следующие уровни:

- стратегический (верхний) уровень управления, ответственный за постановку и выбор целей, прогнозирование, высокоуровневую обработку информации;
- тактический (промежуточный) уровень управления, ответственный за распознавание образов, построение карты местности, планирование траектории;
- уровень управления (нижний уровень), ответственный за выдерживание пара-метров управления исполнительными механизмами

Пример предлагаемой в [91] трехуровневой системы управления STRL приведен на Рис. 1.11.

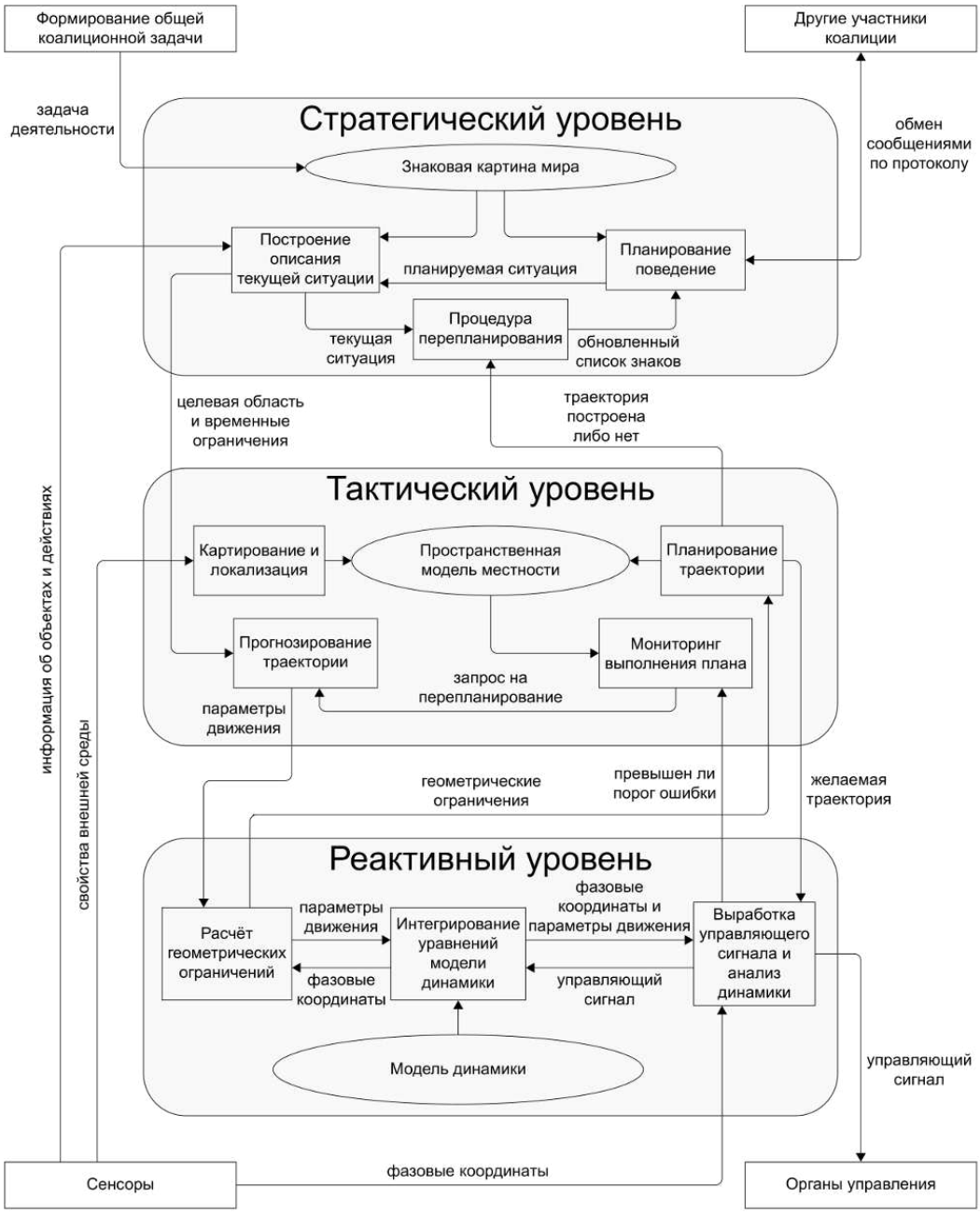


Рисунок 1.11 — Иерархическая система управления сложными техническими объектами STRL.

Почти всегда в подобных системах управления выделяется модуль планирования траектории (пути), на вход которого передается актуальная модель местности, в которой функционирует робот (и дополнительно – прогноз её изменений). Таким образом, задача планирования траектории обособляется от других задач управления робототехнической системой и становится возможным разработка изолированных методов её решения. С другой стороны, очевидно, что такие методы должны учитывать специфику общей задачи и возникающие из-за этого требования и ограничения: высокая скорость работы, возможность

прямого или косвенного учета ограничений объекта управления (например, ограничения на максимальный радиус поворота), возможность учета (прогнозной) динамики окружающей среды и т.д. Разработка и исследование подобных методов планирования траектории является актуальной научно-технической задачей.

Следует заметить, что в последние годы с связи с активным развитием методов машинного обучения, многослойных искусственных нейронных сетей, появлением больших массивов различных данных (в том числе данных робототехнических экспериментов) развивается альтернативный подход к созданию систем управления роботами, когда система управления представляет собой монолитную, нейросетевую модель, обучаемую в сквозном (end-to-end) режиме. Первые работы, описывающие подобные идеи появились в 2016-2017 гг., см., например, [93—95], и с тех пор этот подход активно развивается. В настоящее время, подобные нейросетевые архитектуры имеют в качестве основных составных блоков визуально-языковую модель (vision-language model) [96], и так называемую модель действий (action model), что в совокупности приводит к визуально-языковой модели действий – Vision-Language-Action Model (VLA) [97; 98]. Среди подобных моделей можно назвать, например, модели RT-1 и RT-2 от компании Deepmind, MolmoAct от института искусственного интеллекта Аллена, семейство моделей π от компании Physical Intelligence и др.[4]. Однако подобный подход имеет ряд существенных ограничений. Перечислим основные из них. Во-первых он не является транспарентным, т.е. затруднительно или даже невозможно установить логическую взаимосвязь между входными и выходными сигналами системы управления (почему робот в определенной ситуации совершает те или иные действия). Соответственно, отладка такой системы практически невозможна. Во-вторых, современные визуально-языковые модели действий являются чрезмерно ресурсоёмкими, т.к. подразумевают использование искусственных нейронных сетей, содержащих миллиарды параметров (весов). Как следствие, использование таких систем непосредственно на борту робота невозможно, т.к. бортовые вычислители подавляющего большинства роботов не обладают необходимой вычислительной мощностью[5]. В

[4]Здесь уместно отметить, что эта область стремительно развивается и приведенные данные, актуальные на конец 2025 года, могут стремительно устареть.

[5]Стоит отметить, что разработка специализированных робототехнических вычислителей активно ускоряется и, вероятно, с течением времени эта проблема отойдёт на второй план.

третьих, подобный подход к построению системы управления весьма чувствителен к данным, которые использовались для обучения нейросетевой модели, и на практике часто не является робастным, т.е. применение подобных систем управления на практике затруднено в условиях, когда решаемые задачи отличаются от тех, что использовались на этапе обучения.

Для компенсации указанных выше проблем может быть использован гибридный подход к построению систем управления [99]. В рамках этого подхода сохраняется модульная структура системы, при этом некоторые модули могут являться обучаемыми, а некоторые – классическими, т.е. основанными на необучаемых подходах к решению поставленных задач. Возможна интеграция обучаемых и необучаемых компонент в рамках одного модуля, в частности – в рамках модуля планирования траектории. Такой подход представляется весьма перспективным, т.к. интеграция нейросетевых и классических подходов может, с одной стороны, привести к повышению эффективности решения задачи планирования (например, за счет использования обучающих выборок), с другой – сохранить теоретические гарантии, например – гарантию того, что на любых входных данных будет получено корректное решение.

Обобщая вышесказанное, можно сделать следующий вывод. На текущем этапе развития науки и техники целесообразным представляется применение модульных систем управления, в рамках которых выделяется отдельный модуль планирования траектории, ответственный за построение пути, вдоль которого впоследствии будет осуществляться перемещение объекта управления (с помощью других модулей системы управления). При этом методы планирования траектории должны учитывать ряд особенностей, возникающих ввиду интеграции решения этой задачи в более общую задачу управления сложным робототехническим объектом: необходимость учета динамики окружающей среды, ограничений на возможные движения робота и др. Для обеспечения прозрачности системы управления и возможности её отладки, целесообразно применения классических методов планирования или гибридных методов, т.е. сочетающих в себе аналитические техники оптимизации и машинное обучение, и обеспечивающих при этом ряд гарантий, основной из которых является гарантия корректности решения на произвольных входных данных.

Методы планирования и следования по траектории В настоящее время известно множество подходов к решению задачи построения пути (траектории)

и обеспечения следования вдоль него для достижения целевого состояния. При этом в контексте современных модульных систем управления (см. предыдущий параграф) эта комплексная задача обычно разбивается на две подзадачи: построения геометрического пути (геометрическое планирование) и обеспечение следования вдоль этого пути [6]. Под геометрическим планированием понимается построение траектории, обладающей лишь геометрическими характеристиками. Подобные траектории представляют собой последовательность точек (на плоскости или в пространстве) и соединяющих их отрезков или других геометрических примитивов (например, фрагментов окружности определенного радиуса, или фрагментов кривых описываемых аналитическим уравнением определенного типа). Для обеспечения следования вдоль построенного пути используются различные методы выработки управляющих воздействий, т.е. методы управления, учитывающие особенности конкретной робототехнической системы.

Для решения задачи геометрического планирования обычно применяется методы одного из двух классов: методы, основанные на графовом представлении окружающей среды и методы, основанные на случайном сэмплировании (и не подразумевающие построение/наличие графой модели местности). В первом случае, сначала строится граф, вложенный в метрическое пространство, описывающий допустимые переходы от одной точки этого пространства к другой, и затем осуществляется поиск пути на подобном графе. Во втором случае, построение пути происходит за счет итерационного сэмплирования (случайного выбора) точек метрического пространства, с помощью которых и строится итоговый путь.

В качестве графовых моделей обычно используются такие графы как графы видимости [1], диаграммы Вороного [2] и графы регулярной декомпозиции [4; 5]. На практике наиболее часто применяются графы последнего типа, т.к. их построение обычно не связано с дополнительными накладными расходами – многие методы и алгоритмы автоматического построения карт местности по наблюдениям датчиков робототехнической системы представляют карты именно в таком виде таких. При этом размер подобных графов (т.е. число вершин и рёбер) может быть весьма велик (десятки или даже сотни тысяч вершин/рёбер), поэтому стандартные методы поиска, такие как алгоритм Дейкстры [8] или

[6] В зарубежной научной литературе на эти две связанные между собой задачи часто ссылаются как на *глобальное* и *локальное* планирование.

Беллмана-Форда [9], оказываются малоэффективными для поиска путей. Для повышения эффективности поиска используются алгоритмы семейства A* [11], такие как JPS [15], HGA* [18], ARA* [43] и др. Эти алгоритмы подразумевают использование эвристических функций для фокусировки поиска. При этом для графов регулярной декомпозиции известен ряд подобных функций, которые удовлетворяют определенным свойствам (а именно свойствам допустимости и монотонности – см. Раздел 1.3), позволяющим сохранить важные теоретические гарантии поиска, в частности гарантию полноты (т.е. гарантию того, что любая решаемая задача будет решена алгоритмом, а на нерешаемой задаче алгоритм корректно завершится) и гарантию отыскания оптимального решения. Благодаря подобным свойствам эти алгоритмы находят широкое применение в робототехнике.

Среди сэмплирующих планировщиков, наиболее распространены алгоритмы семейства RRT [100]. Эти алгоритмы строят дерево поиска, корень которого соответствует начальному положению агента в пространстве. На каждой итерации случайным образом выбирается точка в направлении, которой осуществляется рост дерева. При определенной (достаточно простой) стратегии этого роста можно гарантировать, что алгоритм её реализующий (RRT) является частично полным в вероятностном смысле, т.е. для любой задачи, имеющей решение, вероятность того, что оно будет найдено, стремится к 1 при стремлении числа сэмплов к бесконечности [101]. При этом, к сожалению, гарантии того, что алгоритм корректно завершится за конечное число шагов на задаче, не имеющей решение, нет. К настоящему моменту известно множество модификаций RRT. Некоторые из них, такие как, например, RRT-Connect [102], SBL-RRT [103], DD-RRT [104], направлены на ускорение поиска решения. Другие, такие как, например, RRT-Rope [103], направлены на повышение качества отыскиваемых путей (т.е. снижения их длины). Существенным недостатком RRT является тот факт, что различные запуски алгоритма на одних и тех же входных данных (задачах планирования) приводят к построению различных решений (из-за использования случайного выбора в ключевой процедуре алгоритма – выбора направления для роста дерева). При этом решения нередко получаются низкого качества. Эти два обстоятельства существенным образом снижают практическую применимость алгоритма. Для устранения указанных проблем могут применяться модификации метода, использующие механизм переназначения родителей в дереве поиска (re-wiring) и не останавливающие поиск

после построения решения, а продолжающие сэмплирование и рост дерева для поиска более короткого пути в пределе – кратчайшего. Первым алгоритмом такого сорта являлся RRT*, представленный в [36]. В этой работе, как и в работе [105] продемонстрировано и формально доказано, что при стремлении числа сэмплов к бесконечности, вероятность отыскания оптимального решения (при условии его существования) алгоритмом RRT* стремится к 1. В настоящее время известно множество модификаций алгоритма, направленных на повышение его эффективности: Informed RRT* [106], Informed RRT#[107], BIT* [108] и др. В целом, можно констатировать что подход к построению пути, основанный на случайном сэмплировании, является достаточно развитым и богатым на различные методы и алгоритмы. К их достоинствам можно отнести относительную простоту реализации и наличие теоретических гарантий. С другой стороны предоставляемые гарантии имеют лишь вероятностный характер, т.е. какие-либо утверждения о свойствах алгоритмах верны лишь при стремлении числа сэмплов к бесконечности, что невозможно на практике. Таким образом, во-первых встает вопрос о выборе подходящего ограничения на максимальное число сэмплов: при недостаточно высоком значении вероятности построить путь снижается, при недостаточно низком значении повышается время работы алгоритма. Во-вторых при любом заданном ограничении отыскиваемые решения теряют свойство повторяемости. Т.е. планируемые траектории для одной и той же задачи отличаются от запуска к запуску.

Перейдём теперь к задаче следования вдоль построенной (любым из вышеперечисленных методов) геометрической траектории. Как было сказано выше, эту задачу обычно отделяют от задачи планирования траектории и в англоязычной литературе подобные задачи носят название path following или path tracking, т.е. буквально слежение или следование вдоль пути. К настоящему моменту известно множество методов, предназначенных для выработки управляющих воздействий для обеспечения следования по заданной траектории, например: линеаризация обратной связи [109], метод обратного хода (backstepping) [110], управление с плавающим окном [111], управление с прогнозирующими моделями [112; 113] и др. Если управляемая система является плоской [114], то перспективным представляется создание законов управления, опирающихся на это свойство [115; 116]. Для подобных систем, к числу которых относятся, например, весьма распространенные на практике колесные роботы с дифференциальным приводом или беспилотные летательные аппараты

мульти-роторного типа (коптеры, дроны), известен ряд подходов к построению законов управления. Например, распространен подход, при котором осуществляется линеаризация статической обратной связью и выбор траектории в виде полиномиальной зависимости от времени [117]. Иногда подобный подход комбинируется с методом накрытий [118; 119]. В целом, можно утверждать, что область, предметном изучения которой является задача следования по траектории, изобилует различными методами и подходами, опирающимися на основы теории автоматического управления, и обладающими, во многих случаях, строгими теоретическими гарантиями.

Учет кинематических ограничений при планировании Как было сказано выше, зачастую в системах управления сложными мобильными робототехническими объектами разделяют задачу построения геометрического пути (в виде опорных точек, соединенных отрезками прямых) и задачу следования вдоль этого пути, в рамках которой происходит учет различных ограничений, накладываемых на возможности робота по перемещению в пространстве. Например, робот автомобильного типа не может изменить направление своего движения без продольного перемещения. Такие ограничения принято называть кинематическими. Эти ограничения могут учитываться (в той или иной степени) на этапе построения пути.

Например, известен подход, когда сначала строятся так называемые примитивы движения [120—122], т.е. короткие фрагменты траектории, которые учитывают кинематические ограничения, а затем траектории конструируется из этих примитивов с помощью алгоритмов эвристического поиска. В зарубежной литературе такой подход часто именуют поиском на решетке примитивов (lattice-based planning). Среди широко известных алгоритмов, его реализующих, можно отметить `Hybrid A*` [123], `AD*` [124] и др. Основной недостаток подобных методов состоит в том, что пространство поиска при использовании примитивов движения существенно расширяется по сравнению с геометрическими планированием, что существенно снижает вычислительную эффективность на практике.

Сэмплирующие планировщики, например алгоритмы семейства `RRT`, тоже могут быть адаптированы для учета кинематических ограничений при планировании. Один из таких способов адаптации заключается в замене сэмплирования

в пространстве состояний на сэмплирование в пространстве управлений, как было предложено в [125]. Другой способ состоит в использовании так называемых функций руления (steering functions) при поиске. Эти функции предназначены для генерации примитива движения, соединяющего два произвольных состояния [126; 127]. Конструирование таких функций в общем случае – нетривиальная задача, требующая ресурсоемких вычислений (хотя в отдельных случаях, например для роботов автомобильного типа с фиксированным радиусом поворота, она может быть решена достаточно легко [128—130]). Более того зачастую после генерации примитива, учитывающего кинематические ограничения, оказывается, что такой примитив проходит через препятствия (т.к. точки начала и конца были выбраны случайным образом), и он отбрасывается. В результате число итераций алгоритма существенно возрастает. Для устранения этого недостатка в последнее время применяются различные обучаемые функции руления, см. например [23; 131], позволяющие конструировать примитив движения, следование вдоль которого не приведет к столкновению с препятствиями. Этот подход является перспективным, однако, как и любой подход, опирающийся на методы машинного обучения, он не является робастным, т.е. его эффективность сильно зависит от того на каких данных и сценариях производилось обучение.

Учет кинематических ограничений возможен помимо прочего за счет пост-обработки (сглаживания) построенного геометрического пути [132—135]. Подобный подход концептуально прост и не является ресурсоемким. Тем не менее он требует изначально построенного геометрического пути. Более того, он является чувствительным к входным данным. Т.е. пост-обработка одних путей может приводить к нужному эффекту (они будут успешно преобразованы в траектории, следование вдоль которых не представляет труда), а других – нет. В этой связи разумной представляется идея косвенного учета кинематический ограничений уже на этапе построения первоначального (опорного) пути. То есть перевод кинематических ограничений в геометрические и планирование с учетом этих геометрических ограничений (например, ограничение на максимальную длину секции пути, или на максимальный угол между секциями и т.д.).

Итак, учет кинематических ограничений на этапе планирования траектории является трудоемкой и ресурсоемкой задачей, требующей либо конструирования широкого набора примитивов движения, что снижает эффективность

планирования, основанного на систематическом поиске, либо разработки нетривиальных функций руления для сэмплирующих планировщиков. Наименее трудоемким представляется подход, основанный на сглаживании геометрического пути. Для повышения эффективности данного подхода целесообразным представляется наложение определенных геометрических ограничений на вид пути, конструируемого при планировании, и учет этих ограничений при поиске.

Учет динамики окружающей среды при планировании В реальных приложениях мобильный агент (робот) нередко функционирует в условиях динамической окружающей среды. Основным источником такой динамики служат другие мобильные объекты, движущиеся в общем с агентом пространстве, например, это могут быть пешеходы или другие роботы. Одним из наиболее простых способов учета динамики окружающей среды является использование специфических алгоритмов следования по траектории. В рамках этого подхода, единожды построенный (геометрический) путь не изменяется, но агент, при следовании по этому пути осуществляет локальное (пере)планирование с учетом изменяющейся обстановки. Методы такого локального планирования представлены, например, в [136—139]. Основной недостаток этого подхода состоит в том, что он не гарантирует в общем случае достижения агентом цели даже в случаях когда, например, единственным источником динамики среды являются движущиеся препятствия и траектории их движения известны (например, это другие роботы в мульти-роботной системе). Более того, итоговая траектория агента при таком подходе обычно оказывается заметно длиннее по сравнению с той, которая бы могла быть построена в случае учета динамики среды на этапе глобального (геометрического) планирования.

Для решения последней задачи, – построения глобального пути до целевого положения с учетом динамики окружающей среды, обычно применяются два подхода, на которые принято ссылаться как на реактивное перепланирование и проактивное планирование. В первом случае акцент делается на адаптацию ранее построенного пути к наблюдаемым в текущий момент изменениям. Иными словами речь идёт о быстром (глобальном) перепланировании в статической среде. Подобные методы известны как для планировщиков, использующих графовое представление окружающей среды, так и для сэмплирующих. Среди методов первого рода стоит отметить алгоритм D* [41] и наиболее известную в

области робототехники модификацию этого алгоритма D*lite [21]. Эти алгоритмы опираются на принцип переиспользования вычислений, произведенных на прошлых итерациях построения пути, за счет которого во многих случаях удается существенно ускорить поиск на текущей итерации. Подобные алгоритмы поиска пути на графах часто носят название инкрементальных [140]. Среди подобных алгоритмов можно отметить LPA* [42], FSA* [141], DifferentialA* [142], GAA* [143], MPGAA* [144] и другие. Одним из первых известных алгоритмов планирования на основе случайного сэмплирования, использующих идею быстрой адаптации ранее построенного плана (пути) к текущим условиям можно назвать алгоритм DynamicRRT [145], который сами авторы характеризуют как "вероятностный аналог широко-используемого алгоритма D*lite". К алгоритмам подобного класса относятся и такие алгоритмы как MP-RRT [146], RRT^X [22] (этот алгоритм является одним из наиболее популярных среди исследователей и разработчиков), RT-RRT [147] и др.

Алгоритмы перечисленные выше относятся подходам, реализующим реактивное (пере)планирование, и никак не учитывают прогноз информации о динамике окружающей среды. При этом такая прогнозная информация обычно доступна. Она поступает либо от модуля прогнозирования, который на основе наблюдений строит прогноз перемещения окружающих агента объектов, либо от модуля коммуникации, в процессе обмена информацией с другими агентами (последнее особенно распространено в контексте мульти-агентных сценариев). Подход, при котором подобная информация о пространственно-временных траекториях динамических препятствий учитывается на этапе построения траектории, принято называть проактивным планированием. Основная сложность в реализации подобного подхода заключается в необходимости учета не только пространственного, но и временного измерения задачи, т.к. доступность произвольных локаций (и, соответственно, возможность их использования) и переходов между ними меняется со временем.

В планировщиках проактивного типа, основанных на графовом представлении рабочего пространства и эвристическом поиске, для учета времени вводится дискретная шкала $[0, 1, ..., T-1, T]$. Теперь каждая вершина графа может индуцировать T состояний поиска, где T – ограничение на максимальное время достижения цели. Далее в подобном пространстве состояний осуществляется стандартный эвристический поиск, как это описано, например в [148]. Основной недостаток подобного подхода состоит в существенном расширение

(за счет учета времени) пространства состояний и, как следствие, снижении скорости работы алгоритмы (которое может быть весьма существенным в практических задачах). Одним из возможных решений этой проблемы является использование техники частичного раскрытия состояний когда при некоторые из потомков текущего состояния не создаются сразу (и не добавляются в дерево поиска) [149; 150]. Другой, более распространенной и эффективной техникой является *безопасно-интервальное планирование*, при котором последовательные моменты времени группируются в интервалы и поиск осуществляется в пространстве пар "вершина графа – (безопасный) интервал времени". Впервые подобный алгоритм, SIPP (от англ. safe interval path planning), был предложен в [151]. К настоящему моменту известно множество модификаций этого метода [152—155]. Подход безопасно-интервального планирования позволяет с одной стороны существенно сократить пространство поиска (т.е. повысить вычислительную эффективность алгоритма), с другой – сохранить все теоретические гарантии, присущие алгоритмам поиска на графе, в том числе гарантию отыскания оптимального решения задачи планирования. Комбинация этих свойств выгодно отличает алгоритмы семейства SIPP от аналогов и обуславливает их активное применение на практике.

Что касается сэмплирующих планировщиков, алгоритмов семейства RRT, то для них тоже имеются модификации, обеспечивающие проактивное планирование с учетом траекторий будущего движения окружающих объектов (препятствий, других роботов и т.д.). Одним из первых планировщиков такого рода был алгоритм TB-RRT [156], опирающийся на сэмплирование в пространстве-времени и стандартный (однонаправленный) рост случайного дерева поиска. В [157] был предложен алгоритм ST-RRT*, использующий двунаправленное дерево и стратегию изменения временной границы, что в совокупности обеспечивает построение ассимптотически оптимальных решений (т.е. вероятность того, что будет построен путь, минимизирующий время достижения целевого состояния, стремится к единице при стремлении числа сэмплов к бесконечности). Аналогичный алгоритм, но предназначенный для более общей формулировки задачи (когда речь идёт не о динамической среде, а о динамической функции стоимости) представлен в [158]. Идея использования безопасных интервалов, т.е. осуществления безопасно-интервального планирования, используется и при разработке сэмплирующих планировщиков для проактивного планирования траектории в динамических средах [159; 160]. Как показано

в [160] такой подход характеризуются повышенной вычислительной эффективностью, особенно, когда число динамических препятствий велико.

В целом, можно утверждать, что задача учета динамики окружающей среды на этапе планирования траектории является достаточно проработанной и к настоящему моменту существует большое число способов её решения. В случае, когда отсутствует прогнозная информация о траекториях движения окружающих агента объектов целесообразно использование реактивных планировщиков, опирающихся на принцип переиспользования вычислений, произведенных на предыдущих итерациях планирования. В случаях, когда такая информация имеется (например, в мульти-агентных системах, когда в качестве динамических препятствий выступают другие агенты/роботы), то предпочтительно использование проактивных планировщиков, которые осуществляют поиск в пространстве-времени. При этом наиболее эффективными из них являются те, что опираются на принцип безопасно-интервального планирования.

Повышение вычислительной эффективности алгоритмов поиска пути на графах регулярной декомпозиции Как было сказано выше, одним из наиболее распространенных подходов к планированию траектории, обладающих рядом несомненных достоинств (таких как, например, строгие теоретические гарантии), является эвристический поиск пути на графах регулярной декомпозиции. Базовым алгоритмом такого поиска является алгоритм A* [11]. Соответственно, важным направлением исследований и разработок является направление, связанное с повышением вычислительной эффективности алгоритма. Такого эффекта можно достичь, очевидно, разными способами. Например, за счет комбинирования нескольких эвристик при поиске [161], использования техники взвешивания эвристики [14], дополнительной фокусировки поиска с помощью вторичной эвристики [12], техники отсечения симметрий [15], иерархического поиска [18] и др. Однако подавляющее большинство подобных модификаций базового алгоритма опирается на использование стандартных, сконструированных вручную эвристических функций.

Для графов регулярной декомпозиции к подобным функциям относятся Евклидово расстояние, расстояние Чебышева, Манхеттенское расстояние и др. Эти функции легки в вычислении и обладают важными свойствами, позволяющими гарантировать, например, отыскание оптимальных (или – ограниченно

суб-оптимальных) решений. Однако эти эвристики в силу своей универсальности часто не являются информативными на практике, т.к. не учитывают расположение препятствий в рабочей области агента (наличие непроходимых вершин на ГРД). Из-за этого происходит наивная фокусировка поиска и рассматривается чрезмерно большое число вершин, расположенных рядом с препятствиями. Разрешить эту фундаментальную проблему, присущую всем модификациям алгоритма A*, можно лишь за счет использования более информированных эвристик, которые более точно оценивают расстояния до цели с учетом взаимного расположения начальной и целевой вершин и препятствий.

Много-агентная постановка Многие практические задачи (например, сортировка товаров на автоматизированных складах или мониторинг больших территорий) подразумевают функционирование не одного а сразу нескольких мобильных роботов в общем рабочем пространстве. Более того, число этих роботов может в отдельных случаях достигать сотен и даже тысяч [162]. Возникает задача мульти-агентной (или много-агентной) навигации, которая в общем виде формулируется следующим образом. Группа мобильных агентов функционирует в общем пространстве, при этом каждому агенту необходимо переместится в известное ему целевое положение, избегая столкновений как с другими агентами, так и со статическими и динамическими препятствиями. В последнее время интерес к методам решения этой задачи существенно возрос, в основном в связи с их востребованностью в области складской и сервисной робототехники [163; 164] и в интеллектуальных транспортных системах [165; 166].

Различные допущения, принимаемые на этапе формализации указанной задачи, оказывают существенное влияние на выбор методов её решения. Одной из наиболее распространенных и активно изучаемых формализаций, наиболее релевантной данной работе, является так называемая классическая задача много-агентного планирования путей (в англоязычной терминологии – Classical Multi-Agent Pathfinding) [26]. Здесь и далее будем ссылаться на неё как на ЗМПП. В этой задаче время дискретизуется, т.е. разбивается на такты, и считается что за один за один такт каждый агент может совершить действие перемещения либо ожидания. Рабочее пространство, в котором перемещаются агенты, моделируется в виде графа, т.е. считается, что агенты могут перемещаться лишь вдоль ребер априори заданного графа и совершать действия ожидания в его вершинах. На практике в классической задаче много-агентного

планирования обычно используются 4-связные графы регулярной декомпозиции [4; 5]. Это связано с тем, что подобные графы, во-первых, позволяют упростить формализацию конфликта (конфликт привязывается к конкретному ребру или вершине графа и конкретному такту времени), во-вторых, они достаточно хорошо соотносятся с реальными применениями, особенно в области складской робототехнике, которая в настоящее время является одним из основных драйверов развития методов решения ЗМПП. Сама задача состоит в том, чтобы построить n неконфликтных путей (где n – число агентов на графе), следуя по которым агенты достигают целевых положений без столкновений. В качестве критерия качества решения задачи обычно выступает либо суммарное время достижения целей (в англоязычной терминологии – flowtime или sum of costs) или максимальное время достижения цели индивидуальным агентом (в англоязычной терминологии – makespan). Эти критерии не являются равнозначными и в общем случае их одновременная минимизация невозможна [167].

Известно, что в случае неориентированного графа получение допустимого решения ЗМПП возможно за полиномиальное время [168]. С другой стороны, получение оптимальных решений – это NP-трудная задача [46; 167][7]. Таким образом, большинство исследований в этой области сфокусировано на разработке алгоритмов двух типов: алгоритмов, гарантирующих построение оптимального решения, более эффективных в вычислительном плане, чем методы полного перебора, и алгоритмов, направленных на поиск суб-оптимальных решений и обеспечивавших высокую производительность в случаях, когда построение оптимальных решений (с помощью алгоритмов первого класса) затруднительно из-за ограничений по времени и памяти.

Методы, предназначенные для оптимального решения ЗМПП, принято разделять на два класса. К первому относятся решатели, основанные на сведении ЗМПП к одной из классических задач компьютерных наук, таких как, например, задача о потоках. Ко второму классу относятся алгоритмы, опирающиеся непосредственно на поиск совокупности неконфликтных путей на заданном графе [170]. Среди известных и популярных сведений ЗМПП можно упомянуть сведения к задаче о потоках [171; 172], задаче о выполнимости булевых формул (задаче SAT) [173; 174], задаче целочисленного линейного программирования [175; 176] и др. Среди алгоритмов, основанных на поиске,

[7] В случае же ориентированных графов даже получение неоптимальных решений ЗМПП является NP-трудной задачей [169]

известны такие методы как A*+ID+OD [47], ICTS [50; 177], M* [49; 178], CBS [48; 179]. Пожалуй, наиболее популярным при этом является последний алгоритм – CBS, реализующий принцип так называемого конфликтно-ориентированного планирования. Этот метод, так же как и классический алгоритм эвристического поиска (единственного пути) A*, представляет собой некоторую генерализованную схему (или же – протокол) поиска, которая может быть дополнена и модифицированы многими разными способами, что и обеспечивает его популярность. Существует множество модификаций алгоритма CBS, сохраняющих основные свойства алгоритма, в частности – гарантию построения оптимального решения, но при этом позволяющих существенно повысить быстродействие алгоритма на практике, см. [180—184]. Известны вариации алгоритма CBS, которые нацелены на получение ограниченно суб-оптимальных решений, т.е. таких решений, стоимость которых не превосходит стоимость оптимальных решений, более чем в заданное пользователем число раз. Среди них стоит отметить такие алгоритмы как ECBS [185] и EECCBS [186]. В целом, можно утверждать, что к настоящему моменту известно множество решателей ЗМПП, гарантирующих построение оптимальных (и ограниченно суб-оптимальных) решений. Однако, подобные методы, несмотря на многочисленные улучшения, не позволяют решать практические задачи за разумное время, когда число агентов, для которых требуется построить некофликтные пути, достаточно велико (превышает десятки, сотни агентов). Поэтому активно развиваются алгоритмы построения субоптимальных решений ЗМПП, направленные на максимальное повышение практической вычислительной эффективности.

Среди вычислительно-эффективных субоптимальных решателей ЗМПП можно выделить алгоритмы, основанные на правилах (rule-based planning), и алгоритмы, основанные на идее приоритизированного планирования (priority-based planning). Среди решателей первого класса широко-известными являются такие методы, как Bibox [187], Push And Swap [188], Push And Rotate [189], PIBT [190]. При этом последний алгоритм является наиболее популярным из-за простоты реализации и того факта, что по качеству решений он зачастую превосходит другие планировщики, основанные на правилах. Решатели второго класса основаны на сведении ЗМПП к серии задач планирования индивидуальных траекторий в динамической среде [191]. Их общая схема выглядит так: сначала каким-либо образом агентам назначаются приоритеты; после этого осуществляется последовательное планирование индивидуальных траек-

торий, в соответствии с установленными приоритетами. При этом на каждой последующей итерации планирования все пути, построенные ранее, трактуются как динамические препятствия. Благодаря этому ЗМПП декомпозируется на ряд более простых задач индивидуального планирования, которые решаются последовательно. При соблюдении некоторых условий на расположение начальных и конечных вершин в графе приоритизированные планировщики являются полными, т.е. гарантируют отыскание решение ЗМПП (в общем случае это неверно) [192; 193]. Помимо этого, существует ряд техник, позволяющих повысить эффективность приоритизированного планирования, например, за счет ре-приоиритизации [194], введения частичного порядка на множестве приоритетов [195] или динамической локальной приоритизации в ходе планирования [196].

На основе перечисленных выше субоптимальных алгоритмов решения ЗМПП разрабатываются и многосоставные решатели, направленные на получение решения более высокого качества и использующие указанные алгоритмы в качестве вспомогательных процедур. Например, алгоритмы семейства `LaCAM` [197—199] опираются на выделение подцелей в рамках ЗМПП (промежуточных положений агентов на графе) и используют алгоритм `PIBT` для последовательного достижения этих подцелей. Иными словами исходная ЗМПП разбивается на последовательность других (более простых) ЗМПП, которые решаются с помощью субоптимального алгоритма. Подобная схема позволяет в том числе получить т.н. решатель с отсечением по времени (в англоязычной терминологии – anytime algorithm). Подобные решатели реализуют следующий (практико-ориентированный) принцип – получить первоначальное решение ЗМПП как можно быстрее (при этом качество такого решения может быть низким), а затем, при наличии времени, итеративно модифицировать решение, улучшая его (вплоть до получения оптимального решения). Такие подходы часто используют в качестве алгоритма для генерации первоначального решения и приоритизированные планировщики, см., например, [200—203].

Помимо классических техник, используемых для решения ЗМПП (эвристический поиск, комбинаторная оптимизация, сведение к задаче о выполнимости будевых формул и пр.), в последнее время набирает популярность применения методов и моделей машинного обучения для много-агентного планирования путей. Например, методы машинного обучения могут использоваться для выбора алгоритма много-агентного планирования, наиболее

подходящего под конкретную задачу (карту, расположение агентов) [204; 205]. Помимо этого методы машинного обучения могут использоваться для выучивания различных эвристических правил выбора, присутствующих в классических алгоритмах решения ЗМПП [206; 207]. Широкое распространение получают и специализированные решатели ЗМПП, опирающиеся на обучение с подкреплением, имитационное обучение, интеграцию обучаемых компонент с классическими техниками поиска. Обычно такие подходы предполагают формирование на этапе обучения индивидуальной стратегии, предназначенной для выбора действия на основе локального наблюдения. Затем, выученная стратегия используется для решения ЗМПП. Одним из первых таких алгоритмов, получивших широкую известность и распространение, был `PRIMAL` [208]. Позже он был доработан и представлен в [27] под именем `PRIMAL2`. Оба этих алгоритма используют обучение с подкреплением для формирования стратегии агентов. В методах `DHC` [28], `DCC` [209], `PICO` [210], `SCRIMP` [211] помимо блоков выбора действия используются (обучаемые) блоки обмена информацией между агентами. Методы `FOLLOWER` [212], `MATS-LP` [213], `Switcher` [214] основаны на комбинации эвристического поиска и обучения с подкреплением для решения ЗМПП. Наконец, в последнее время наблюдается большой интерес к разработке решателей ЗМПП, основанных на идее имитационного обучения, см. например [29]. Можно констатировать, что идет активное развитие обучаемых методов много-агентного планирования путей. Подобные методы обладают определенными достоинствам. Например, большинство таких методов являются распределенными (distibuted) в том смысле, что каждый агент принимает решение о том, какое действие выбрать в той или иной ситуации, независимо от других агентов. Это приводит к неограниченным возможностям масштабирования, т.к. количество агентов в системе больше не является ограничивающим фактором. С другой стороны, обучаемые решатели ЗМПП не обладают важными теоретическими гарантиями, в частности они не гарантируют, что задача будет решена (т.е. что все агенты достигнут своих целевых положений). Многие из таких методов не гарантируют построение безопасных стратегий, т.е. стратегий, обеспечивающих выбор таких действий, исполнение которых не приведёт к конфликту (столкновению). Это обстоятельство существенно ограничивает практическую ценность подобных решателей.

Одним из наиболее перспективных направлений в области много-агентного планирования путей является разработка методов, позволяющих отказаться

от основных ограничений классической ЗМПП, которые, состоят в следующем, любое действие агентов (перемещение, ожидание) занимает одинаковое количество времени (один такт); агенты могут перемещаться в пространстве лишь, используя заранее фиксированную топологию ГРД (в основном – 4-связный ГРД). В частности интерес представляет ЗМПП, когда допускается перемещение между произвольными вершинами ГРД, т.к. такая задача (как было показано выше) наиболее близка к реальным робототехническим применениям, когда многоуровневая система управления роботом подразумевает построение (на тактическом уровне) геометрического пути, т.е. последовательности прямолинейных сегментов, и затем следование вдоль пути (на реактивном уровне) с помощью методов локального планирования и управления. В настоящее время эффективные оптимальные и суб-оптимальные методы решения такой вариации ЗМПП отсутствуют.

Выводы по итогам анализа литературы Одной из наиболее актуальных проблем, в контексте которой возникают различные задачи поиска пути (путей) на графе регулярной декомпозиции, является проблема разработки систем управления сложными робототехническими устройствами, предназначенными для их автономного функционирования в сложно-структурированной среде. На текущем этапе развития науки и техники целесообразным представляется применение модульных систем управления, в рамках которых выделяется отдельный модуль навигации, обеспечивающий достижение объектом управления заданного целевого положения (состояния). Сама задача навигации может рассматриваться либо как задача последовательного принятия решения, либо как комбинированная задача, включающая этап планирования траектории (пути) до целевой позиции, а затем следования вдоль этого пути. В контексте последнего подхода возникают различные формализации задачи планирования, в том числе представление этой задачи в виде поиска пути на графе регулярной декомпозиции. При этом методы поиска пути должны учитывать ряд практико-ориентированных особенностей и ограничений, возникающих ввиду интеграции решения задачи поиска в более общую задачу управления сложным робототехническим объектом – см. Рис. 1.12.

К основным ограничениям относится, во-первых, необходимость учета динамики окружающей среды на этапе планирования, в частности, в случае когда

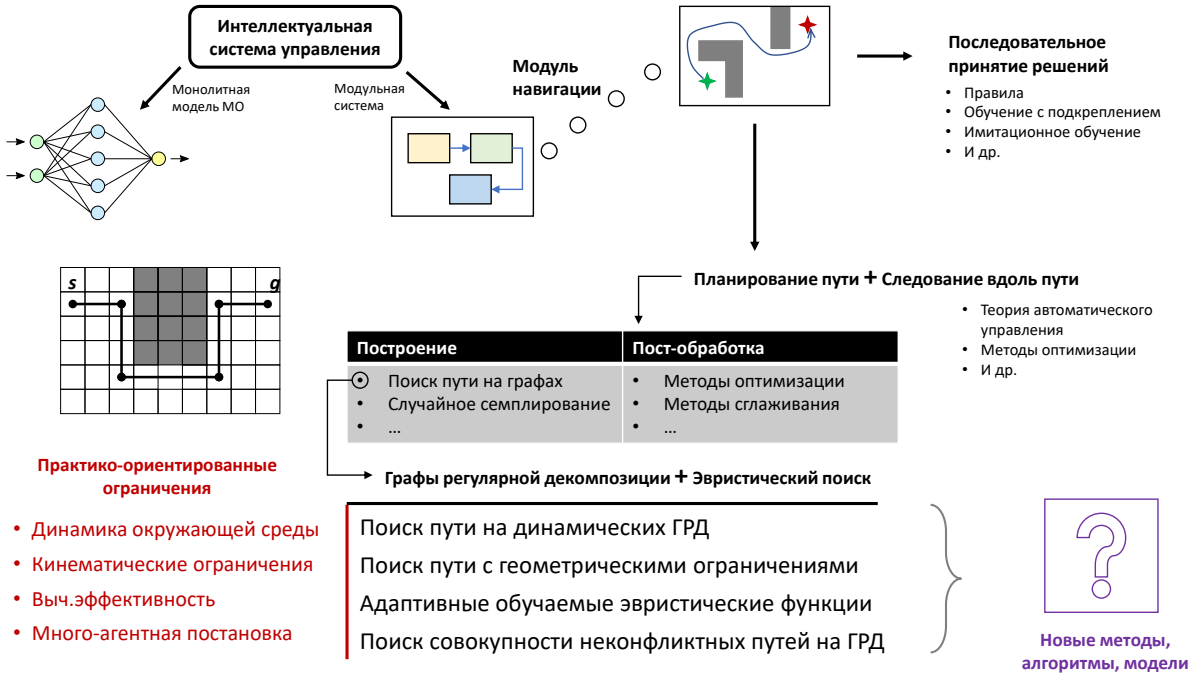


Рисунок 1.12 — Перспективные задачи поиска пути на графе регулярной декомпозиции, возникающие в контексте современных систем управления робототехническими объектами.

траектории движения динамических препятствий известны. Отдельный интерес в этой связи представляет много-агентная постановка задачи планирования, когда речь идёт о поиске совокупности неконфликтных (в пространстве-времени) путей, следование вдоль которых обеспечивает достижение целей группой мобильных роботов. В этом случае, если задача решается централизовано, спланированные траектории для отдельных роботов могут рассматриваться как траектории динамических препятствий и должны соответствующим образом учитываться при поиске путей для других роботов группы. В настоящее время известен ряд эффективных подходов к решению подобного рода задач, таких как безопасно-интервальное планирование, конфликтно-ориентированное планирование и приоритизированное планирование. Однако указанные подходы подразумевают возможность перемещения лишь между ортогонально-смежными вершинами графа регулярной декомпозиции. Это приводит к построению путей, содержащих большое число поворотов и избыточных по длине (и, соответственно, по времени достижения целевых состояний). Для устранения этого недостатка необходима разработка новых методов. В частности, перспективным представляется разработка и исследование алгоритмов, допускающих возможность перемещения между произвольными вершинами графа: алгоритмов,

гарантирующих отыскание оптимальных решений, практическая эффективность которых превосходит эффективность алгоритмов полного перебора и традиционных методов эвристического поиска, и эффективных алгоритмов, направленных на быстрое получение субоптимальных решений, качество которых близко к качеству оптимальных решений на практике (но, возможно, без соответствующих теоретических гарантий).

Актуальной задачей является повышение вычислительной эффективности алгоритмов эвристического поиска за счет построения информативных эвристик, учитывающих особенности конкретного экземпляра задачи планирования (т.е. расположения препятствий, начальной и целевой позиции). Перспективным подходом к решению этой задачи является применения методов машинного обучения для автоматической аппроксимации эвристических функций и новых способов интеграции этих функций с алгоритмами поиска так, чтобы обеспечивались гарантии получения корректного ответа за конечное число шагов на произвольных входных данных. Отдельный интерес представляет создание таких гибридных планировщиков, т.е. планировщиков, использующих обучаемые эвристики, которые бы гарантировали отыскание решений, стоимость которых не превышает стоимость оптимальных решение более, чем в заданное пользователем число раз (т.н. ограниченно суб-оптимальных решений).

Наконец, ещё одной важной задачей является учет кинематических ограничений объекта управления при планировании. Традиционные способы такого учета (планирование в пространстве поиска, индуцируемого примитивами движения, или использование функций руления для решения краевых задач в сэмплирующих планировщиках) являются достаточно ресурсоемкими. Поэтому целесообразным представляется декомпозиционный подход, основанный на построении геометрического пути, косвенно учитывающего кинематические ограничения, для последующего применения алгоритма сглаживания/оптимизации пути. В этой связи перспективным представляется разработка и исследование методов поиска пути на графе регулярной декомпозиции, учитывающих геометрические ограничения, в частности – методов, предназначенных для поиска путей, состоящих из прямолинейных секций, т.ч. угол отклонения одной секции от другой не превышает заданного порога.

Именно на решении указанных выше проблем, ассоциированных с ограничениями, возникающими при разработке современных систем управления

подвижными (мобильными) робототехническими объектами, и решении соответствующих задач поиска пути на графе регулярной декомпозиции предлагается сфокусироваться в данной работе.

1.5 Выводы по главе

В данной главе были приведены предварительные сведения о предметной области исследования: описаны графы специального вида, вложенные в метрическое пространство – графы регулярной декомпозиции, показана связь задач поиска путей на таких графах с задачей планирования траектории для мобильных агентов – колесных роботов, оперирующих в пространстве с препятствиями. Описаны базовые техники и алгоритмы систематического эвристического поиска для построения путей. Приведен обзор литературы, на основании которого можно сделать вывод о перспективности следующих направлений исследований, на которых предлагается сконцентрироваться в данной работе:

1. Разработка и исследование новых, вычислительно эффективных алгоритмов поиска пути на графах регулярной декомпозиции, проходимость вершин/ребер которых зависит от времени, допускающих возможность перемещения между произвольными вершинами графами.
2. Разработка и исследование новых методов много-агентного планирования, т.е. поиска совокупности неконфликтных путей на графах регулярной декомпозиции. В частности, методов опирающихся на принципы конфликтно-оринетированного поиска для обеспечения гарантий оптимальности отыскиваемых решений и методов, опирающихся на принципы приоритизированного планирования для обеспечения высокорой скорости отысакния (субоптимальных) решений.
3. Разработка методов поиска пути на графе регулярной декомпозиции с учетом геометрических ограничений, в частности – ограничений на угол поворота между секциями, образующими путь, т.к. это позволяет косвенно учитывать кинематические ограничения, возникающие в реальных робототехнических применениях.

4. Интеграция методов машинного обучения и методов эвристического поиска для построения алгоритмов планирования, обладающих теоретическими гарантиями на корректность отыскиваемых решений (что не всегда возможно при использовании исключительно обучаемых моделей) и позволяющих сократить число необходимых итераций поиска до получения решения за счёт автоматического конструирования информативных эвристик (с помощью современных нейросетевых моделей), учитывающих особенности конкретной задачи планирования.

Глава 2. Поиск пути на динамических графах регулярной декомпозиции

В предыдущей главе показана связь между задачей поиска пути на графе регулярной декомпозиции, вложенном в метрическое пространство, с задачей планирования траектории мобильного агента, например – робота, перемещающегося в среде с препятствиями. Обычно система управления таким роботом строится по модульному принципу (см. например [91; 92; 215; 216]), в рамках которого выделяется модуль построения опорной траектории в виде промежуточных точек, соединенных прямолинейными сегментами. Следование по этой траектории обеспечивается отдельным модулем управления. В практических задачах необходимо обеспечить безопасное и надежное функционирование робота в условиях динамической среды, в частности, когда в ней присутствуют другие движущиеся объекты, будущие траектории движения которых известны (например, имеется прогноз их движения, построенный с помощью современных алгоритмов анализа сенсорной информации). Целесообразно в таком случае учитывать динамику окружающей среды не только на этапе следования вдоль траектории, но и на этапе построения самой траектории. С формальной точки зрения задача планирования такой траектории может быть сформулирована как задача поиска пути на графе, возможность прохождения по вершинам и ребрам которого зависит от времени. Будем называть такие графы для краткости динамическими. Это существенно более сложная задача, нежели поиск пути на статическом графе, т.к. пространство поиска теперь включает в себе дополнительное измерение (временное). Именно эта задача будет рассмотрена в данной главе работы. В ней будет предложен и исследован новый метод поиска, сочетающий в себе принципы безопасно-интервального планирования, ленивого поиска и техники достраивания ребер графа между произвольными вершинами по ходу поиска. Будет доказан ряд утверждений, характеризующих свойства предложенного алгоритма, в частности будут доказаны теорема об оптимальности отыскиваемых решений. Будут предложены субоптимальные модификация алгоритма, существенно повышающие его вычислительную эффективность за счет процедуры жадного переопределения предшественника (родителя) в дереве поиска и за счет фокусировки поиска. Будут установлены свойства предложенных модификаций, в том числе касающиеся качества

отыскиваемых решений. Помимо теоретических исследований предложенных алгоритмов, будут проведены их экспериментальные исследования (численные эксперименты) и представлен анализ полученных результатов.

2.1 Постановка задачи

2.1.1 Базовая постановка (задача PFD)

Рассмотрим конечный 8-связный взвешенный ГРД $\mathcal{G} = (V, E, w)$, вложенный в метрическое пространство $\mathcal{M} = \mathbb{R}^2$ так, как это описано в Главе 1. То есть каждой вершине ГРД соответствует точка на плоскости. С каждым ребром графа $e = (u, v)$ будем ассоциировать отрезок AB, т.ч. $coord(u) = A$, $coord(v) = B$, где $coord : V \to \mathbb{R}^2$ – функция, сопоставляющая вершины графа точкам в $\mathbb{R}^2$ (функция вложения графа в пространство). Определим вес ребра как $w(e) = \|AB\|$.

Пусть расстояние измеряется в условных единицах. Без ограничения общности можно считать, что горизонтально-смежные и вертикально-смежные вершины ГРД отстоят друг от друга на расстоянии 1. Следовательно, вес ортогональных ребер ГРД равен 1, а диагональных ребер – $\sqrt{2}$.

Введем в рассмотрение множество временных моментов $T = [0, +\infty)$ и с каждой вершиной ГРД будем ассоциировать множество моментов времени, $T_{safe}(v) \subseteq T$, именуемых *безопасными*. При этом эти будем полагать, что эти моменты сгруппированы в конечную последовательность *безопасных интервалов*:

$$\begin{aligned} &T_{safe}(v) = (SI_1(v), SI_2(v), ..., SI_n(v)), \\ &SI_i(v) = [t_l^i, t_u^i], \\ &t_l^i, t_u^i \in T_{safe}(v), \\ &t_l^i \leqslant t_u^i. \end{aligned} \tag{2.1}$$

Указанные безопасные интервалы являются максимальными в том смысле, что расширение каждого из них невозможно:

$$\begin{aligned} &\forall SI = [t_l, t_u] \in T_{safe}(v), \forall \varepsilon > 0 : \\ &t_l - \varepsilon \notin T_{safe}(v), \\ &t_u + \varepsilon \notin T_{safe}(v). \end{aligned} \tag{2.2}$$

Будем допускать, что последний безопасный интервал в последовательности может иметь вид $[t_l, +\infty)$, где $0 \leqslant t_l < \infty$. Условие 2.2 для него записывается в усеченном виде: $\forall \varepsilon > 0 : t_l - \varepsilon \notin T_{safe}(v)$. Очевидно, что введенные определения допускают наличие для некоторой вершины v множества безопасных моментов времени $T_{safe}(v)$, представимого в виде единственного безопасного интервала $[0, +\infty)$.

Интуитивно, безопасные интервалы вершины определяют, в какой момент времени эта вершина может использоваться, а в какой нахождение в ней невозможно. Это востребовано при решении практических задач планирования траектории некоторого мобильного агента (например, мобильного робота) в среде с движущимися препятствиями. Более подробно об этом будет сказано ниже.

Аналогично множествам безопасных интервалов для вершин графа, для каждого ребра e задано множество безопасных моментов времени $T_{safe}(e)$, сгруппированных в последовательность безопасных интервалов (для которых выполняются условие 2.2).

Опираясь на понятие безопасных интервалов вершин и ребер введем следующее определение.

Определение 17. *Динамический* ГРД – это тройка:

$$(\mathcal{G} = (V,E),\ T_{safe}(V),\ T_{safe}(E)), \tag{2.3}$$

где $\mathcal{G} = (V,E)$ – это ГРД, а $T_{safe}(V)$, $T_{safe}(E)$ – множества безопасных интервалов его вершин и ребер соответственно.

Введем понятие действия перемещения.

Определение 18. *Действие перемещения* – это пара (e, t), где $e = (u,v) \in E, t \in T$.

Интуитивно действию перемещения соответствует движение от одной вершины к другой вдоль ребра ГРД, начало которого приходится на момент времени t. Здесь уместно заметить, что действия (e, t_1) и (e,t_2) при $t_1 \neq t_2$ формально являются разными действиями.

С каждым действием перемещения будем ассоциировать его *продолжительность*, равную весу соответствующего ребра. Поскольку вес ребра в рассматриваемом случае равен длине отрезка, соединяющего вершины графа, то можно считать, что движение от одной вершины в другую происходит с постоянной единичной скоростью.

Определение 19. Действие перемещения (e, t) является *допустимым*, если:

$$\begin{aligned} & t \in T_{safe}(source(e)), \\ & t \in T_{safe}(e), \\ & t + w(e) \in T_{safe}(target(e)). \end{aligned} \tag{2.4}$$

Здесь (и далее в работе) запись $source(e)$, $target(e)$ означает начальную, конечную вершину ребра e.

Согласно определению, действие перемещения допустимо, если момент начала действия принадлежит безопасному интервалу исходной вершины и ребра, определяющего это действие, при этом момент окончания перемещения принадлежит безопасному интервалу целевой вершины.

Рассмотрим теперь две вершины ГРД v_{start} и v_{goal}, такие что среди безопасных интервалов v_{start} присутствует интервал $SI_1 = [0, t_u], t_u > 0$, а среди безопасных интервалов v_{goal} присутствует интервал $SI_{last} = [t_l, +\infty), t_l = const < +\infty$. Будем называть такие вершины v_{start} и v_{goal} допустимыми начальной и целевой вершиной.

Определение 20. Допустимый путь из v_{start} в v_{goal} на динамическом ГРД – это последовательность пар $(e, t), e \in E, t \in T$:

$$\pi(v_{start}, v_{goal}) = ((e_1, t_1), (e_2, t_2), ..., (e_n, t_n)),$$

т.ч.:

$$source(e_1) = v_{start}, \tag{2.5а}$$

$$target(e_n) = v_{goal}, \tag{2.5б}$$

$$\forall i = \overline{1,n-1} : \ target(e_i) = source(e_{i+1}), \tag{2.5в}$$

$$\forall i = \overline{1,n} : \ t_i \in T_{safe}(e_i), \tag{2.5г}$$

$$\forall i = \overline{1,n-1} : \ t_{i+1} \geqslant t_i + w(e_i) \tag{2.5д}$$

$$\forall i = \overline{1,n-1} : \ [t_i + w(e_i), t_{i+1}] \subseteq T_{safe}(v_{i+1}), \tag{2.5е}$$

$$t_1 \in SI_1(v_{start}), \tag{2.5ж}$$

$$t_n + w(e_n) \in SI_{last}(v_{goal}). \tag{2.5и}$$

Условия 2.5а, 2.5б означают, что допустимый путь начинается в v_{start} и заканчивается в v_{goal}; 2.5в говорит о том, что каждые два подряд идущих ребра должны иметь одну общую вершину; 2.5г – момент начала следования по каждому ребру принадлежит безопасному интервалу; 2.5д – движение по ребру начинается либо сразу же после окончания движения вдоль предшествующего ребра, либо – после некоторой паузы, при этом, если такая возникает, то соответствующий временной интервал ожидания является безопасным, что диктуется условиями 2.5е, 2.5ж; наконец, условие 2.5и говорит о том, что допустимый путь завершается в конечной вершине в последнем безопасном интервале этой вершины.

Заметим, что согласно данному выше определению, допустимый путь содержит лишь действия перемещения. Однако условие 2.5е позволяет начинать очередное действие перемещения не обязательно в момент окончания предыдущего. Таким образом можно говорить о неявно заданных *действиях ожидания* в вершинах графа. При этом такие действия, очевидно, обязаны быть допустимыми, т.е. интервал ожидания должен полностью принадлежать одному из безопасных интервалов вершины, в которой это ожидание совершается (см. 2.5ж).

Определение 21. Вес (стоимость) допустимого пути из $\pi(v_{start}, v_{goal})$ на динамическом ГРД это величина, рассчитываемая по формуле:

$$cost(\pi) = t_n + w(e_n). \tag{2.6}$$

То есть вес пути равен (первому) моменту времени достижения целевой вершины.

Определим теперь формально интересующую нас задачу поиска.

Определение 22. Задача поиска (допустимого) пути на динамическом графе – это набор:

$$\text{PFD} = (\mathcal{G}, T, \{T_{safe}(v)\}, \{T_{safe}(e)\}, v_{start}, v_{goal}) \tag{2.7}$$

Аббревиатура PFD используется как сокращение следующего словосочетания на английском языке: **P**ath **F**inding on **D**ynamic Grid.

Определение 23. Решение задачи PFD – это допустимый (в соответствии с Определением 20) путь из первого безопасного интервала v_{start} в последний безопасный интервал v_{goal}.

Определение 24. Оптимальное решение задачи PDF – это решение, минимизирующее стоимость пути (в соответствии с Определением 21).

Пример задачи PDF и двух её решений представлен на Рис. 2.1.

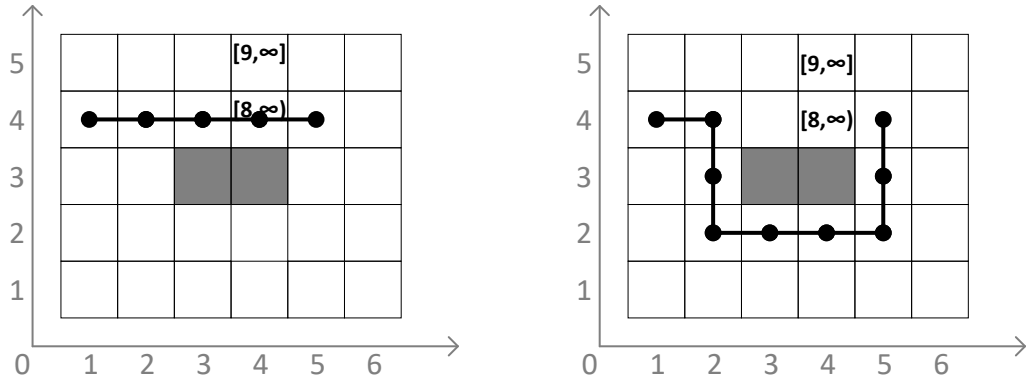


Рисунок 2.1 — Пример задачи поиска пути на динамическом ГРД.

В этой задаче начальная вершина имеет координаты $v_{start} = (1,4)$, конечная: $v_{goal} = (5,4)$. Множество безопасных интервалов для любого ребра в графе равно $\{(0, \infty)\}$. Множество безопасных интервалов для любой вершины ГРД, кроме $u = (4,4)$ и $v = (4,5)$, равно $\{(0, \infty)\}$. Множества безопасных интервалов для вершин u, v определены как $T_{safe}(u) = \{(8,\infty)\}$, $T_{safe}(v) = \{(9,\infty)\}$, соответственно. Слева и справа на рисунке показаны два различных пути: π_1, π_2. Путь π_1 является кратчайшим в геометрическом смысле путем из начальной вершины в конечную, однако этот путь подразумевает что действие перемещения в вершину $u = (4,4)$ из предшествующей вершины $(3,4)$ может быть совершенно не раньше момента времени 7, т.к. иначе вершина u будет достигнута в небезопасный интервал (что противоречит определению допустимости пути). Соответственно, вершина u будет достигнута в $t = 8$ (начало безопасного интервала этой вершины), а целевая вершина – в момент времени 9. То есть $cost(\pi_1) = 9$. При этом второй изображенный путь, π_2, который обходит препятствие снизу и является более длинным в геометрическом смысле, имеет меньшую стоимость: $cost(\pi_2) = 8$ (т.к. все вершины и ребра на этом пути содержат лишь один бесконечный безопасный интервал). Именно путь π_2 является оптимальным решением задачи PDF в данном случае.

Связь с задачей планирования траектории мобильного агента В контексте планирования траектории для мобильного агента, поставленная задача может трактоваться следующим образом. Необходимо найти траекторию из

начального положения агента в целевое, избегающую столкновений как со статическими так и с динамическими препятствиями. При этом траектория может состоять из прямолинейных сегментов (ребер графа), вдоль которых агент движется с постоянной скоростью, без учета инерциальных эффектов и действий ожидания произвольной продолжительности. Само ожидание может совершаться только в вершинах графа. Продолжительность того или иного ожидания заранее не фиксирована и должна рассчитываться в ходе решения задачи. Минимизируется время достижения целевого положения. Чем раньше (быстрее) траектория привела агента в целевой положение, тем лучше. Не обязательно, что кратчайший в геометрическом смысле путь порождает наилучшую траекторию, т.к. при следовании вдоль этого пути может потребоваться совершение длительных действий ожидания для того, чтобы избежать столкновений в динамическими препятствиями. При этом может быть выгодней достичь цели более длинным путем, но при следовании по которому ожиданий совершать не нужно. Таким образом перед планировщиком стоит нетривиальная задача оптимизации, где пространство поиска решений является комбинированным – необходим учет различных вариантов как с геометрической точки зрения (более короткие пути предпочтительны), так и с временной (траектории содержащие суммарно менее продолжительные ожидания предпочтительны).

Отдельно стоит сказать, что на практике при решении задач автоматизации перемещения (навигации) в среде с движущимися препятствиями планирование обычно производится на некоторый временной горизонт, т.е. траектории движения динамических препятствий учитываются лишь в определенном интервале. При этом само предсказание траектории движения препятствий – это отдельная (сложная) задача, для которой известны и применяются на практике множество разных методов. В данной работе эта задача не рассматривается. После того как траектории предсказаны, граф аннотируется безопасными интервалами вершин и ребер, которые и передаются на вход планировщику. Сама процедура аннотирования является внешней по отношению к процессу планирования и в данной работе предполагается, что такое аннотирование произведено.

2.1.2 Расширенная постановка (задача AA-PFD)

Будем считать, что на парах (различных) вершин ГРД определена функция:

$$los: \ V \times V \rightarrow \{true, false\}, \tag{2.8}$$

которая определяет возможность перехода не только между смежными, но и между *произвольными* вершинами графа.

Будем считать, что функция w, определяющая веса ребёр, определена и на произвольных парах вершин: $w : V \times V \rightarrow \mathbb{R}^+$.

С точки зрения задачи планирования траектории, функция $los(u, v)$ определяет возможность перехода между удаленными точками рабочего пространства с учетом особенностей агента (например, его размера) и расположения статических препятствий. На практике функция $los(u, v)$ (от. англ. "line-of-sight" – линия видимости) возвращает $true$ тогда и только тогда, когда агент может совершить перемещение по прямой от вершины u до вершины v без столкновения со статическими препятствиями, расположенными в рабочем пространстве.

Будем считать, что для каждой пары вершин u и v, для которых функция los возвращает $true$, задано множество безопасных моментов времени $T_{safe}(u, v)$ (по аналогии с $T_{safe}(e)$, $e \in E$). Как и ранее, будет считать, что переход из u в v возможен, только если он начинается в один из моментов времени, принадлежащих $T_{safe}(u, v)$.

Теперь мы можем модифицировать определение допустимого пути (Определение 20) так, что элементом пути может быть не только ребро графа, но и произвольная пара вершин, при условии, что функция los на этих вершинах возвращает $true$ (и все остальные ограничения, диктуемые Определением 20 сохраняются). Путь теперь может содержать переходы между удаленно расположенными вершинами, если такие переходы безопасны с точки зрения учета статических (функция los) и динамических (множества $T_{safe}(u,v)$) препятствий. В англоязычной литературе по планированию подобного рода пути на ГРД носят название any-angle путей, т.е. путей, допускающих переходы под произвольным углом, а не только по горизонтали, вертикали и диагонали, как на стандартных 4(8)-связных ГРД. В данной работе такие пути будут называться *ПН-путями*, где приставка "ПН" означает "произвольное направление".

Для полноты дадим формальное определение ПН-пути (на основе ранее введенного Определения 20).

Определение 25. ПН-путем из v_{start} в v_{goal} на динамическом ГРД назовем последовательность троек $(u, v, t), u \in V, v \in V, t \in T$:

$$\pi(v_{start}, v_{goal}) = ((u_1, v_1, t_1), (u_2, v_2, t_2), ..., (u_n, v_n, t_n)),$$

т.ч.:

$$u_1 = v_{start}, \tag{2.9а}$$

$$v_n = v_{goal}, \tag{2.9б}$$

$$\forall i = \overline{1,n-1} : \ v_i = u_{i+1}, \tag{2.9в}$$

$$\forall i = \overline{1,n} : \ t_i \in T_{safe}(u_i, v_i), \tag{2.9г}$$

$$\forall i = \overline{1,n-1} : \ t_{i+1} \geqslant t_i + w(u_i, v_i) \tag{2.9д}$$

$$\forall i = \overline{1,n-1} : \ [t_i + w(u_i, v_i), t_{i+1}] \subseteq T_{safe}(u_{i+1}), \tag{2.9е}$$

$$t_1 \in SI_1(v_{start}), \tag{2.9ж}$$

$$t_n + w(u_n, v_n) \in SI_{last}(v_{goal}) \tag{2.9и}$$

$$\forall i = \overline{1,n} : \ los(u_i, v_i) = true \tag{2.9к}$$

Определим теперь задачу поиска ПН-пути на динамическом ГРД.

Определение 26. Задача поиска (допустимого) ПН-пути на динамическом ГРД – это набор:

$$\text{AA-PFD} = (\mathcal{G}, T, los, \{T_{safe}(v)\}, \{T_{safe}(u, v)\}, v_{start}, v_{goal}). \tag{2.10}$$

Определение 27. Решение задачи AA-PFD (от англ. Any-Angle Path Finding on Dynamic grids) – допустимый (в соответствии с определением 26) ПН-путь из первого безопасного интервала v_{start} в последний безопасный интервал v_{goal}.

Определение 28. Оптимальным решением задачи AA-PFD будем называть решение, минимизирующее стоимость ПН-пути.

Итак, оптимальное решение задачи AA-PFD – это такой ПН-путь, при следовании по которому целевая вершина достигается в как можно более ранний момент времени.

Здесь и далее будем ссылаться на ПН-путь просто как на путь (там где это понятно из контекста).

Пример задачи AA-PFD и несколько вариантов её решения представлен на Рис. 2.2

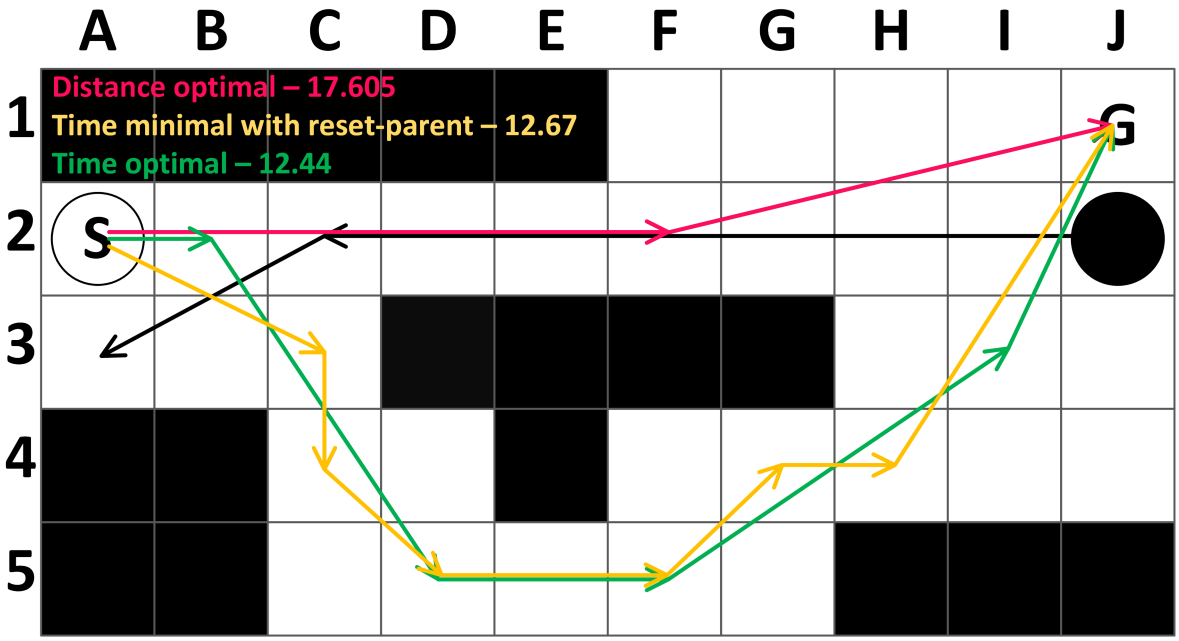


Рисунок 2.2 — Пример задачи AA-PFD.

На рисунке выше не изображены безопасные интервалы, но показана траектория движения динамического препятствия, которое перемещается из клетки $J2$ в клетку $A3$. Из-за этого часть ребер и вершин, x-координата которых равна 2, блокируется в определенные моменты времени и безопасные интервалы этих вершин и ребер сокращаются так, что стоимость кратчайшего геометрического пути, $\pi_{geom} = A2 \rightarrow J2$, становится равной 17.605. При этом стоимость путей, обходящих центральное статическое препятствие через клетки $D5$,$F5$, ниже. Так, зеленым цветом на рисунке показан путь $\pi_{opt} = A2 \rightarrow B2 \rightarrow D5 \rightarrow F5 \rightarrow I3 \rightarrow J1$, который является оптимальным решение представленной задачи AA-PFD.

Заметим, что задача AA-PFD в общем случае допускает больше вариантов решения, чем чем задача PFD, т.к. при решении первой задачи разрешаются переходы между произвольными вершинами графа. Более того, теоретически возможны случаи, когда на одном и том же динамическом ГРД решение AA-PFD существует, а PFD – нет, т.к. в исходной топологии графа невозможно построить путь, удовлетворяющий всем ограничениям, при этом если же разре-

шить переходы вне исходной топологии (т.е. переходы между произвольными вершинами как в AA-PFD), то путь может быть найден.

2.2 Методы и алгоритмы

Опишем в начале наивный подход к решению задачи PFD, основанный на использовании известного метода эвристического поиска A* [11], затем опишем метод поиска, основанный на безопасно-интервальном планировании, SIPP [151]. Далее предложим новые методы решения более сложной модификации задачи PFD, а именно – задачи AA-PFD, опирающиеся на эвристический поиск и безопасно-интервальное планирование, и установим их теоретические свойства.

2.2.1 Эвристический поиск с настраиваемой минимальной задержкой

Введем в рассмотрение минимально возможную длительность ожидания – $\delta = const > 0$. Будем искать теперь решение задачи PFD с учетом этого обстоятельства с помощью алгоритма эвристического поиска A* (описание которого для случая статического графа дано ранее в Разделе 1.3).

В рассматриваемом случае определим состояние поиска как пару $n = (v, t)$, где v – вершина графа, t – момент времени. Концептуально состояние означает, что "агент находится в вершине v в момент времени t". Заметим, что в пространстве состояний могут порождаться состояния с совпадающими вершинами, но разными моментами времени. Такие состояния не являются дубликатами и считаются различными. Как и в стандартном A* для каждого состояния n будем хранить следующую информацию:

- $g(n)$ (g-значение состояния) – минимальный вес пути из начального состояния в состояние n, известный к текущей итерации алгоритма поиска;

– $h(n)$ (h-значение состояния) – эвристическая оценка веса пути из состояния n в целевое состояние;
– $f(n)$ (f-значение состояния) – величина, рассчитываемая по формуле $f(n) = g(n) + h(n)$, т.е. оценка веса пути из начального состояние в целевое через n;
– $parent(n)$ – указатель на состояние, предшествующее состоянию n в дереве поиска.

В рассматриваемом случае g-значение состояния $n = (v,t)$ определяется как временной момент достижения вершины v из начальной вершины, т.е. $g(n) = t$. h-значение вершины – это оценочное время, требуемое для достижения v_{goal} из v. Будем использовать в качестве функции h (эвристики) Евклидово расстояние. Поскольку скорость перемещения вдоль ребер графа равна единице, то данная эвристическая функция является допустимой и монотонной. Следовательно, как было показано в разеде 1.3, использование этой эвристики гарантирует отыскание оптимального решения.

Для адаптации алгоритма A* необходимо модифицировать процедуру создания состояний-последователей. В стандартном алгоритме A* эти состояния соответствуют смежным вершинам графа. В рассматриваемом случае необходимо, во-первых, определить принадлежит ли время перехода и время достижения этих состояний заданным безопасным временными интервалам, во-вторых дополнить множество состояний-последователей состоянием, которое получается в результате осуществления действия ожидания в текущей вершине (опять же, проверив принадлежность момента окончания ожидания безопасному интервалу. Псевдокод процедуры получения состояний-последователей представлен на Рис. 2.3. В представленной процедуре цикл в строках 2-9 отвечает за генерацию состояний потомков, соответствующих перемещению в смежные вершины, а строки 10-14 – за генерацию потомка, соответствующего ожиданию в текущей вершине.

Стоит обратить внимание на важный момент. В строке 10 производится проверка условия $t + \delta \in T_{safe}(v)$, и если эта проверка выполняется то считается, что в текущий момент времени, t, можно (безопасно) совершить действие ожидание продолжительностью δ в вершине v (и соответствующее состояние добавляется в множество потомков). Заметим однако, что величина задержки δ может быть в общей случае достаточно большой, что может существовать такой момент времени $\hat{t} \in [t, t + \delta]$, что $\hat{t} \notin T_{safe}(v)$. То есть в общем случае

Процедура `GenerateTSuccesors`(n, $\mathcal{G}$, $\{T_{safe}(v)\}$, $\{T_{safe}(e)\}$, δ)**:**

Входные данные: Состояние поиска $n = (v,t)$, граф $\mathcal{G}$, множества безопасных временных моментов (интервалов) для вершин и ребер графа $\{T_{safe}(v)\}$, $\{T_{safe}(e)\}$, минимальная задержка для действия ожидания δ

Выходные данные: Множество состояний-последователей $SUCC$ для состояния n

```
1  SUCC ← ∅
2  foreach e = (v, v') ∈ G do
3      t' ← t + w(e)
4      if t' ∉ T_safe(v') or t ∉ T_safe(e) then
5          continue
6      n' ← GenerateSearchNode(v', t')
7      g(n') ← t'
8      parent(n') ← n
9      SUCC ← SUCC ∪ {n'}
10 if t + δ ∈ T_safe(v) then
11     n_wait ← GenerateSearchNode(v, t + δ)
12     g(n_wait) ← t + δ
13     parent(n_wait) ← n
14     SUCC ← SUCC ∪ {n_wait}
15 return SUCC
```

Рисунок 2.3 — Процедура генерации состояний-потомков при эвристическом поиске на динамическом ГРД.

тот факт, что начальный и конечный момент времени совершения действия ожидания принадлежат безопасному интервалу вершины, не гарантирует, что само действие ожидание полностью безопасно. Здесь и далее будем считать, что параметр δ выбран так, что:

$$\forall v \in \mathcal{G},\, T_{safe}(v) = \{SI_i(v) = [t_l^i, t_u^i]\} :\ \delta < t_l^{i+1} - t_u^i \tag{2.11}$$

Тогда очевидно, что процедура, приведенная на Рис. 2.3, гарантирует генерацию только корректных состояний-последователей.

Теперь модификацию алгоритма A* для поиска пути на динамическом ГРД с учетом настраиваемой задержки можно представить виде, показанном на Рис. 2.4.

Алгоритм A*-WithTime($\mathcal{G}$, v_{start}, v_{goal}, $\{T_{safe}(v)\}$, $\{T_{safe}(e)\}$, δ, h):

Входные данные: Граф $\mathcal{G}$, начальная и целевая вершины v_{start}, v_{goal}, множества безопасных временных моментов (интервалов) для вершин и ребер графа $\{T_{safe}(v)\}$, $\{T_{safe}(e)\}$, минимальная продолжительность действия ожидания δ, монотонная эвристическая функция h

Выходные данные: Путь π

1 $n_{start} \leftarrow$ GenerateSearchNode$(v_{start}, 0)$
2 $g(n_{start}) \leftarrow 0$; $parent(n_{start}) \leftarrow$ null
3 $OPEN \leftarrow \{n_{start}\}$; $CLOSED \leftarrow \varnothing$
4 **while** $OPEN \neq \varnothing$ **do**
5 $n \leftarrow arg\min_{n \in OPEN}(f(n) = g(n) + h(n))$
6 $OPEN = OPEN \setminus \{n\}$
7 $CLOSED = CLOSED \cup \{n\}$
8 **if** $n.v = v_{goal}$ **and** $n.t \in SI_{last}(v_{goal})$ **then**
9 **return** ReconstructPath(n)
10 $SUCC \leftarrow$ GenerateTSuccesors$(n, \mathcal{G}, \{T_{safe}(v)\}, \{T_{safe}(e)\}, \delta)$
11 UpdateOpenClosed$(SUCC)$
12 **return** *failure*

Рисунок 2.4 — Модификация алгоритма эвристического поиска A* для динамического ГРД.

Как и в стандартном алгоритме A*, на этапе инициализации (строки 1–3) создается начальное состояние, которое помещается в список OPEN. Естественно, в рассматриваемом случае начальное состояние характеризуется не только заданной вершиной графа v_{start}, но и моментом времени 0. Это состояние является корнем дерева поиска. Список CLOSED изначально пуст. Затем на каждой

итерации основного цикла (строки 4–11) в списке OPEN определяется состояние с минимальным f-значением (строка 5), извлекается (удаляется) из OPEN и помещается в список CLOSED (строки 6–7). Если это состояние является целевым, т.е. оно соответствует целевой вершине графа, v_{goal}, и при этом момент достижения v_{goal} принадлежит последнему безопасному интервалу этой вершины, $SI_{last}(v_{goal})$, то искомый путь найден и может быть восстановлен с помощью родительских указателей (строки 8–9). В противном случае происходит раскрытие состояния поиска (строки 10–11). Сначала (строка 10) создаются состояния-последователи с помощью представленной на Рис. 2.3 функции `GenerateTSuccessors`, которая учитывает время. Затем эти состояния используются для обновления списков OPEN и CLOSED (строка 11) так же, как и в классическом A*. На этом шаге используется поиск (и сравнение) состояний в OPEN и CLOSED. В рассматриваемом случае состояния сравниваются не только по вершинам графа, но и по моментам времени. То есть два состояния $n = (v, t)$ и $n' = (v', t')$ считаются совпадающими тогда и только тогда, когда $v = v'$ и $t = t'$. Будем ссылаться на подобную модификацию алгоритма A* как на алгоритм A*-WithTime.

2.2.2 Безопасно-интервальное планирование

Основным недостатком алгоритма A*-WithTime, описанного выше, при решении задач поиска пути на динамических ГРД, является чрезмерно большое пространство поиска. Это происходит из-за того, что алгоритм оперирует отдельными моментами времени при поиске, т.е. определяет состояние поиска как $n = (v, t)$, где v – вершина графа, а t – момент времени. Теоретически каждая вершина графа может порождать бесконечное число состояний поиска. На практике этого, т.е. порождения бесконечного числа состояний, обычно не происходит, т.к. целевая вершина достигается раньше. Тем не менее число создаваемых состояний (и соответственно итераций алгоритма поиска) может быть весьма велико. Для сокращения этого числа в работе [151] был предложен принцип безопасно-интервального планирования (англ. Safe Interval Path Planning) и одноименный алгоритм его реализующий – SIPP. Его идея состоит в определении состояний поиска не через отдельные моменты времени, а че-

рез интервалы времени, содержащие множества моментов времени. Далее при поиске состояния, характеризующиеся одним и тем же интервалом, считаются идентичными, что позволяет существенно сократить варианты перебора.

Формально состояние поиска в `SIPP` определяется как $n = (v, SI)$, где $SI = [t_l, t_u]$ – это один из безопасных интервалов вершины v. Идентичными (совпадающими) считаются состояния, у которых совпадают и вершины и интервалы.

Основное отличие алгоритма `SIPP` от алгоритма `A*-WithTime` состоит (помимо способа определения состояний поиска) в том, как создаются состояния-последователи для раскрываемого состояния. Псевдокод этой процедуры представлен на Рис. 2.5.

Процедура `GenerateSIPPSuccesors`(n, $\mathcal{G}$, $\{T_{safe}(v)\}$, $\{T_{safe}(e)\}$):

Входные данные: Состояние поиска $n = (v,[t_l, t_u])$, граф $\mathcal{G}$, множества безопасных интервалов для вершин графа $\{T_{safe}(v)\}$, множества безопасных интервалов для ребер графа $\{T_{safe}(e)\}$

Выходные данные: Множество состояний-последователей $SUCC$ для состояния n

1 $SUCC \leftarrow \varnothing$
2 **foreach** $e = (v, v') \in \mathcal{G}$ **do**
3 **foreach** $SI' = [t'_l, t'_u] \in T_{safe}(v')$ **do**
4 $\delta \leftarrow$ `GetMinDelay`$(e, g(n), [t_l, t_u], [t'_l, t'_u], T_{safe}(e))$
5 **if** $\delta \neq \varnothing$ **then**
6 $n' \leftarrow$ `GenerateSearchNode`(v', SI')
7 $g(n') \leftarrow g(n) + \delta + w(e)$
8 $parent(n') \leftarrow n$
9 $SUCC \leftarrow SUCC \cup \{n'\}$
10 **return** $SUCC$

Рисунок 2.5 — Процедура генерации состояний-потомков при эвристическом поиске на динамическом ГРД.

Для каждой вершины v', смежной с вершиной v (строка 2), происходит итерация по всем безопасным интервалам $\{SI'\}$ (строка 3). Для каждого такого

безопасного интервала SI' далее определяется минимально возможное время достижения v' из v (принадлежащее SI'). Очевидно, что самый ранний момент времени, в который вершина v' может быть достигнута, достигается тогда, когда переход по ребру $e = (v, v')$ совершается сразу после достижения v. Время достижения вершины v, определяющей состояние n, есть g-значение этого состояния, т.е. $g(n)$, а продолжительность перехода есть вес ребра, т.е. $w(e)$. Очевидно, минимально возможное время достижения v' может быть вычислено как $g(n)+w(e)$. Однако существует ряд случаев, когда переход по ребру e сразу же невозможен. Например, если $g(n) \notin T_{safe}(e)$, то есть момент времени $g(n)$ не принадлежит безопасному интервалу ребра, или, если $g(n) + w(e) < t'_l$, то есть переход по ребру сразу же приводит к достижению v' до начала безопасного интервала SI'. В общем случае, удобно ввести в рассмотрение понятия минимальной задержки и ее продолжительности, δ, т.е. такой задержки, которую необходимо совершить в v, чтобы последующий переход в v' был корректным и при этом вершина v' достигалась бы в минимально возможный момент времени, принадлежащий интервалу SI'. Тогда, общая формула расчета g-значения состояния потомка выглядит следующим образом:

$$g(n') = g(n) + \delta + w(e) \tag{2.12}$$

Вычисление δ выполняется согласно процедуре `GetMinDelay`, псевдокод которой будет представлен чуть позже. Это вычисление происходит в строке 5 алгоритма генерации состояний-последователей `GenerateSIPPSuccesors`. Далее, если такая задержка успешно определена (строка 5), то создается соответствующее состояние (строки 6–8) и оно добавляется в множество потомков для состояния n (строка 9).

Важно отметить, что алгоритм `SIPP` не подразумевает использования явно-заданного действия ожидания в процессе генерации состояний-потомков, в отличии от алгоритма `A*-WithTime`. В `SIPP` считается, что продолжительность минимально-необходимого действия ожидания вычисляется автоматически с помощью процедуры `GetMinDelay`. В определенном смысле эта процедура считается внешней по отношению к алгоритму непосредственно поиска, т.е. допускается, что она может быть реализована по-разному. Более того в оригинальной статье, посвященной алгоритму `SIPP`, [151], реализация этой процедуры не приводится. Авторы лишь постулируют, что при генерации со-

стояний-потомков алгоритм `SIPP` некоторым образом определяет минимально возможный момент времени их достижения.

В рассматриваемом случае, когда для вершин и ребер графа заданы множества безопасных интервалов процедура `GetMinDelay` может быть реализована, например, так, как показано на Рис. 2.6. Опишем эту процедуру более подробно.

В начале (строка 1) проверяются два необходимых условия корректности перехода из вершины v в вершину v' по ребру e. Во-первых, переход с нулевой задержкой не должен достигать v' в более поздний момент времени, чем заканчивается безопасный интервал этой вершины, равный t'_u (первое условие оператора `if`). Во-вторых, переход с максимальной задержкой, т.е. переход в момент окончания безопасного интервала исходной вершины, равный t_u, не должен достигать v' в более ранний момент времени, чем начинается безопасный интервал этой вершины, равный t'_l (второе условие оператора `If`). Если хотя бы одно из этих условий нарушено, то корректного перехода не существует и возвращается $\varnothing$ (строка 2) – см. Рис. 2.7.

Далее в строке 3 устанавливается минимально-возможная задержка $\delta = 0$. Переменная s_time хранит время начала перехода (равное t_s изначально). Переменная $found$ используется для определения, найден ли корректный момент времени начала перехода (и корректное значение δ). Изначально $found = false$.

В строке 4 происходит проверка на то, каким будет момент времени достижения вершины v', если переход из v будет совершен в момент s_time. Может быть, что этот момент будет лежать слева от начала безопасного интервала вершины v'. Если это так, то момент начала перехода сдвигается вправо (строка 5) на минимально-возможный промежуток, т.е. на такой промежуток, чтобы после ожидания в v достичь v' в момент начала ее безопасного интервала, который равен t'_l. Значение δ обновляется соответствующим образом (строка 6) – см. Рис. 2.8 слева.

Итак, к этому моменту известна продолжительность минимальной задержки δ, такая что вершина v' достигается из вершины v в минимальной возможный момент времени, принадлежащий безопасному интервалу $[t'_l, t'_u]$. При этом момент начала перехода принадлежит безопасному интервалу исходной вершины v', т.е. интервалу $[t_l, t_u]$. Следовательно, переход согласован с

Процедура `GetMinDelay`(e, t_g, $[t_l, t_u]$, $[t'_l, t'_u]$, $T_{safe}(e)$)**:**

Входные данные: Ребро $e = (v, v')$, момент времени, начиная с которого должен быть совершен переход, t_g, безопасные интервалы вершин, образующих ребро, $[t_l, t_u]$, $[t'_l, t'_u]$, безопасные интервалы ребра $T_{safe}(e) = \{SI(e)\}$

Выходные данные: Продолжительность минимальной задержки δ

1 **if** $(t_g + w(e) > t'_u)$ **or** $(t_u + w(e) < t'_l)$ **then**
2 **return** $\varnothing$
3 $found \leftarrow false$; $\delta \leftarrow 0$; $s\_time \leftarrow t_g + \delta$
4 **if** $s\_time + w(e) < t'_l$ **then**
5 $s\_time \leftarrow t'_l - w(e)$
6 $\delta \leftarrow s\_time - t_g$
7 **while** $s\_time \leqslant t_u$ **or** $s\_time + w(e) \leqslant t'_u$ **do**
8 **if** $s\_time \in T_{safe}(e)$ **then**
9 $found \leftarrow true$
10 **break**
11 **else**
12 $SI(e) \leftarrow$ first SI of e after time moment s_time
13 **if** $SI(e) = \varnothing$ **then**
14 **break**
15 $s\_time \leftarrow beg(SI(e))$
16 $\delta \leftarrow s\_time - t_g$
17 **if** $found = false$ **then**
18 **return** $\varnothing$
19 **return** δ

Рисунок 2.6 — Процедура расчета продолжительности минимальной задержки при переходе из одного состояния в другое в алгоритме `SIPP`.

заданными безопасными интервалами обоих вершин. Однако, нужно убедиться, что он согласован и с безопасными интервалами самого ребра e, т.е. с $T_{safe}(e)$.

Для того, чтобы исключить возможность совершения перехода в момент времени, не принадлежащий $T_{safe}(e)$, выполняются цикл, описанный в строках

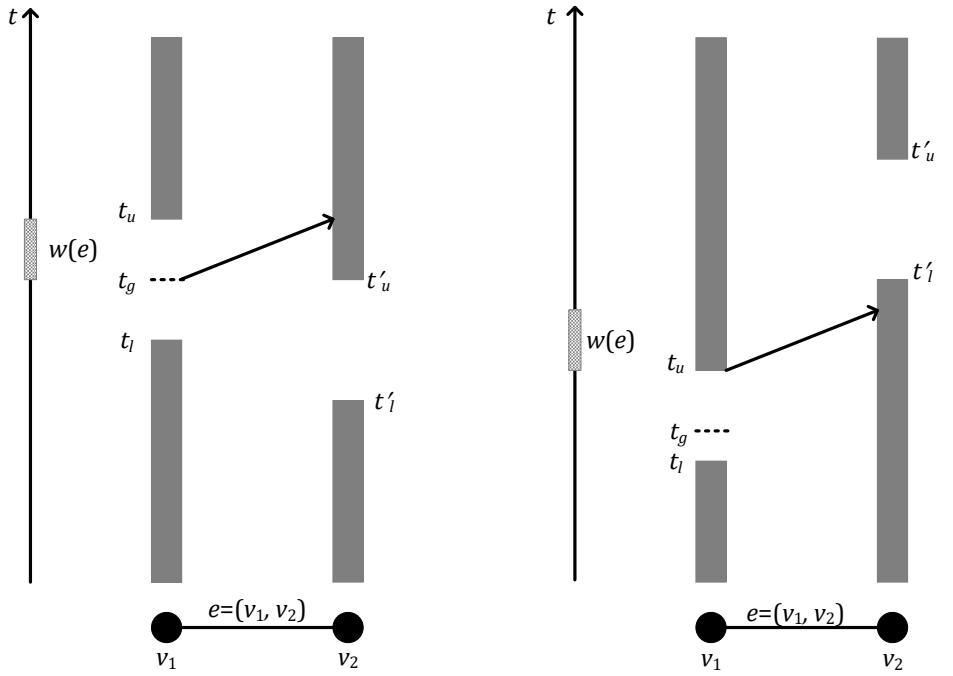


Рисунок 2.7 — Примеры того, когда переход из безопасного интервала $[t_l,t_u]$ вершины v в безопасный интервал $[t'_l,t'_u]$ вершины v' по ребру e с весом $w(e)$ не может быть осуществлен.

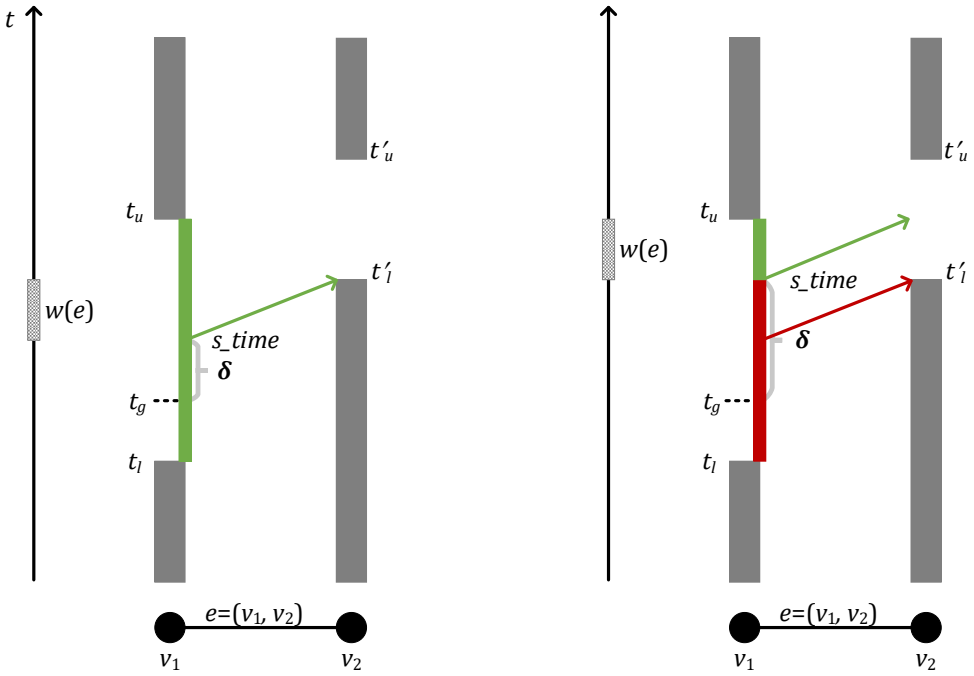


Рисунок 2.8 — Примеры того, как момент начала перехода из вершины v в вершину v' изменяется за счет добавления δ так, чтобы момент окончания перехода приходился безопасный интервал вершины v' и при этом был согласован с безопасными интервалами ребра $e = (v,v')$.

7–16. Именно здесь проверяется, что текущее значение s_time принадлежит одному из безопасных интервалов ребра e (строка 8). Если это так, то момент времени начала перехода и, соответственно, значение минимальной задержки в исходной вершине найдены – флагу $found$ выставляется значение $true$ (строки 8-10). В противном случае (строки 11–16) момент начала перехода сдвигается вправо и становится равным начальному моменту следующего безопасного интервала ребра e – см. Рис. 2.8 справа. При этом случай, когда следующего безопасного интервала e не существует обрабатывается отдельно в строках 13–14. Таким образом, внутри цикла `while` последовательно перебираются безопасные интервалы ребра e, чтобы совершить переход в момент начала одного из них. При этом момент времени начала перехода, s_time, должен лежать слева от конца безопасного интервала вершины v (первое условие цикла `while`), и переход, совершенный в это время, должен заканчиваться в момент времени, принадлежащий безопасному интервалу вершины v' (второе условие цикла `while`). Только при соблюдении всех этих условий флаг $found$ принимает значение $true$ (строка 9) и происходит прерывание цикла.

Итак, если после завершения цикла `while` флаг $found$ равен $false$, то переход по ребру e не согласован с заданными безопасными интервалами. Если же флаг $found$ равен $true$, то это означает, что переход корректен и согласован со всеми заданными безопасными интервалами ($[t_l, t_u]$, $[t'_l, t'_u]$, $T_{safe}(e)$), при этом известно минимально возможное время совершения такого перехода, s_time и минимально возможная задержка (перед совершением такого перехода), δ. Последняя и возвращается в строке 19.

В завершение описания алгоритма `SIPP` приведем его псевдокод – см. Рис. 2.9. Основной цикл алгоритма идентичен основному циклу алгоритмов `A*` и `A*-WithTime`. Основная разница заключается в процедуре генерации потомков, `GenerateSIPPSuccesors`, которая в свою очередь опирается на процедуру вычисления минимальной задержки перед совершением перехода – `GetMinDelay`. Стоит отметить, что в стандартной для алгоритмов поиска процедуре `UpdateOpenClosed` для определения совпадающих состояний в `SIPP` используется пара *(вершина графа, безопасный интервал)* (в соответствии с определением состояний как $n = (v, [t_l, t_u])$). Так же как и ранее, алгоритм подразумевает использование эвристической функции для оценки стоимости пути от произвольного состояния до целевого (равно как и алгоритмы `A*` и`A*-WithTime`). Если эта эвристическая функция удовлетворяет свойству мо-

нотонности (о свойствах эвристических функций см. Раздел 1.3), то SIPP гарантирует отыскание решения минимальной стоимости, т.е. оптимального решения (при этом если решения не существует, то алгоритм корректно завершается, возвращая специальный символ $failure$).

Алгоритм SIPP($\mathcal{G}$, v_{start}, v_{goal}, $\{T_{safe}(v)\}$, $\{T_{safe}(e)\}$, h):

Входные данные: Граф $\mathcal{G}$, начальная и целевая вершины v_{start}, v_{goal}, множества безопасных временных моментов (интервалов) для вершин и ребер графа $\{T_{safe}(v)\}$, $\{T_{safe}(e)\}$, монотонная эвристическая функция h

Выходные данные: Путь π

1 $n_{start} \leftarrow$ GenerateSearchNode$(v_{start}, SI_{first}(v_{start}))$
2 $g(n_{start}) \leftarrow 0$; $parent(n_{start}) \leftarrow$ null
3 $OPEN \leftarrow \{n_{start}\}$; $CLOSED \leftarrow \varnothing$
4 **while** $OPEN \neq \varnothing$ **do**
5 $n \leftarrow arg\min_{n \in OPEN}(f(n) = g(n) + h(n))$
6 $OPEN = OPEN \setminus \{n\}$
7 $CLOSED = CLOSED \cup \{n\}$
8 **if** $n.v = v_{goal}$ **and** $g(n) \in SI_{last}(v_{goal})$ **then**
9 **return** ReconstructPath(n)
10 $SUCC \leftarrow$ GenerateSIPPSuccesors$(n, \mathcal{G}, \{T_{safe}(v)\}, \{T_{safe}(e)\})$
11 UpdateOpenClosed$(SUCC)$
12 **return** *failure*

Рисунок 2.9 — Алгоритм эвристического поиска SIPP.

2.2.3 Безопасно-интервальное планирование для поиска оптимальных решений задачи AA-PFD

Для получения оптимальных решений задачи AA-PFD теоретически может использоваться описанный выше алгоритм SIPP. Задача AA-PFD отличается от PFD тем, что допускаются переходы между произвольными вершинами

в графе (а не только между вершинами, связанными ребрами) в соответствии с заданной функцией los. Если эта функция на паре вершин (u,v) возвращает $true$, то переход возможен, в противном случае переход не возможен. Следовательно, для модификации алгоритма `SIPP` для решения задачи AA-PFD необходимо модифицировать функцию генерации состояний-потомков, таким образом, чтобы при рассмотрении состояния n происходила итерация не только лишь по смежным вершинам, но по всем вершинам графа v', т.ч. $los(n.v, v') = true$. Псевдокод такой процедуры, `GenerateAllSuccesors`, представлен на Рис. 2.10.

Отметим здесь, что сигнатура вызова функции `GetMinDelay` (строка 4) незначительно поменялась по сравнению с таким вызовом из процедуры `GenerateSIPPSuccesors` (см. Рис 2.5), которая использовалась для создания состояний-потомков ранее. Теперь первый аргумент `GetMinDelay` есть пара интересующих нас вершин (вдоль которых осуществляется переход), а последний – набор безопасных интервалов для этой пары. При этом логика работы самой функции расчета минимальной задержки не изменяется.

Одним из недостатков предложенной процедуры генерации состояний-последователей является тот факт, что в ходе её работы необходимо $O(|V|)$ вызовов функции los – см. Строку 2 на Рис. 2.10. На практике эта функция как минимум проверяет проходит ли отрезок прямой, соединяющий вершины v и v', через непроходимые клетки, ассоциированные с вершинами ГРД. Даже если проверка выполняется консервативно (т.е. допускается некоторое количество ложных срабатываний) по, например, алгоритму Брезензема [217][1], то временная сложность функции los составляет $O(|V|)$. Таким образом, временная сложность получения вершин $\{v'\}$, таких что переход в них из вершины v возможен, т.е. $los(v,v') = true$, составляет $O(|V|^2)$, что весьма существенно увеличивает вычислительные трудозатраты и затрудняет применение алгоритма для решения практических задач.

Одним из возможных вариантов решения проблемы можно назвать предварительный расчет значений функции los на всех парах вершин ГРД и сохранение (кэширование) результатов. Однако на графах, содержащих большое число вершин, такая предварительная обработка может занять существенное

[1]Алгоритм Брезенхема – один из наиболее известных в компьютерной графике алгоритмов рисования линий на растровых дисплеях. По сути этот алгоритм определяет какие пиксели дисплея подсветить, чтобы они воспринимались как прямая линия.

Процедура `GenerateAllSuccesors`(n, $\mathcal{G}$, los, $\{T_{safe}(v)\}$, $\{T_{safe}(u,v)\}$):

Входные данные: Состояние поиска $n = (v,[t_l, t_u])$, граф $\mathcal{G}$, функция, определяющая возможность перехода los, множества безопасных интервалов для вершин графа $\{T_{safe}(v)\}$, множества безопасных интервалов для пар вершин графа $\{T_{safe}(u,v)\}$

Выходные данные: Множество состояний-последователей $SUCC$ для состояния n

```
1  SUCC ← ∅
2  foreach v' ∈ G : los(v, v') = true do
3      foreach SI' = [t'_l, t'_u] ∈ T_safe(v') do
4          δ ← GetMinDelay((v,v'), g(n), [t_l, t_u], [t'_l, t'_u], T_safe(v, v'))
5          if δ ≠ ∅ then
6              n' ← GenerateSearchNode(v', SI')
7              g(n') ← g(n) + δ + w(e)
8              parent(n') ← n
9              SUCC ← SUCC ∪ {n'}
10 return SUCC
```

Рисунок 2.10 — Процедура генерации всех состояний-потомков для решения задачи AA-PFD.

время (и некоторое количество памяти для сохранения результатов). Более того, это не решит другую, не менее важную проблему описанного выше подхода к генерации потомков, а именно проблему существенного повышения коэффициента ветвления дерева поиска. Теперь, при решении задачи AA-PFD, для каждого состояния существует $O(|V|)$ вершин, переход в которые возможен, а не 4(8), когда решалась задача PFD на 4(8)-связном ГРД - см. Рис. 2.11. Число же состояний-потомков может быть ещё больше, т.к. в безопасно-интервальном планировании состояние поиска определяется парой *(вершина, безопасный интервал)*.

Таким образом, поиск решения становится трудозатратным и целесообразным представляется разработка метода поиска, использующего, с одной стороны, принцип безопасно-интервального планирования, с другой – опира-

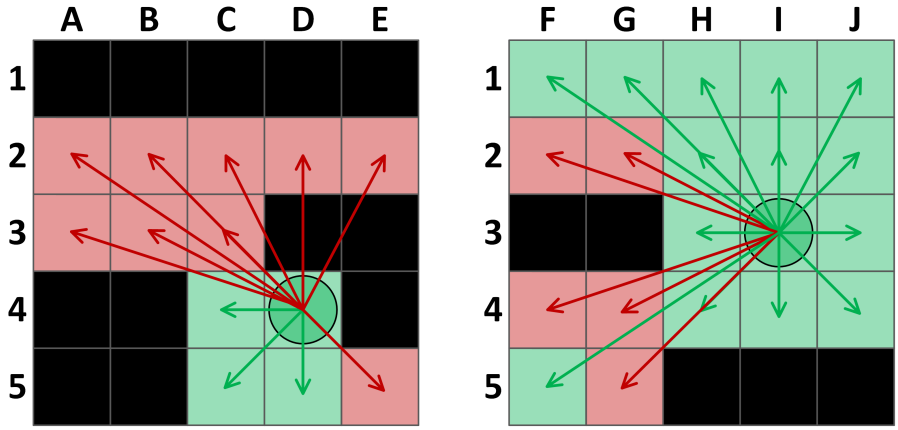


Рисунок 2.11 — Пример множества допустимых и недопустимых переходов из различных вершин ГРД при решении задачи AA-PFD.

ющегося на иной подход к организации самого поиска. Опишем далее такой метод.

Идея Основная идея предлагаемого метода состоит в отказе от использования стандартной для алгоритмов эвристического поиска процедуры раскрытия, т.е. процедуры генерации всех состояний-потомков для текущего состояния (состояния, рассматриваемого на текущий итерации основного цикла алгоритма поиска), и переходе к подходу, который уместно назвать *обратным раскрытием*. Кратко его идея состоит в том, чтобы сгенерировать все возможные состояний поиска на этапе инициализации (это возможно, благодаря тому, что безопасно-интервальное планирование оперирует не отдельными моментами времени, а интервалами) и затем устанавливать корректные отношения “родитель-потомок” между состояниями, восстанавливая таким образом искомый путь. Опишем далее эту идею более подробно и затем перейдём к псевдокоду и доказательству формальных свойств алгоритма (корректность, полнота, оптимальность отыскиваемых решений).

Пусть все состояния поиска, т.е. пары $n = (v, [t_l, t_u])$ созданы на этапе инициализации. Их g-значения равны ∞, т.к. путь до них неизвестен. Состояния-родители для них не определены, т.е. $\forall n : \ parent(n) = null$. Представим теперь, что после некоторого числа итераций алгоритма поиска для отдельных состояний становятся известны минимально-возможные g-значения, обозначаемые как $g^*(n)$, и родительские состояния, благодаря которым эти значения достигаются (т.е. известны пути минимальной стоимости до этих состояний).

Будем называть такие состояния *согласованными*. Рассмотрим теперь оставшиеся, несогласованные состояния (будем считать, что целевое состояние в их числе).

Минимальная стоимость пути до любого несогласованного состояния, очевидно, неизвестна, равно как и состояние-родитель, определяющее такой путь. Так же очевидно, что путь наименьшей стоимости до любого такого состояния возможен только за счет перехода (или последовательности переходов) из согласованных состояний. Таким образом, согласованные состояния являются *потенциальными родителями* для несогласованных. Если организовать процесс рассмотрения переходов из согласованных состояний в несогласованные таким образом, чтобы последние становились согласованными (за конечное число итераций), то задача будет решена. А именно, как только целевое состояние станет согласованным, можно будет остановить процесс и восстановить искомый путь с помощью родительских указателей, как это обычно делается в стандартных алгоритмах эвристического поиска.

Итак, исходная задача сведена к задаче преобразования несогласованных состояний в согласованные, и её предлагается решать за счет систематического рассмотрения переходов из последних в первые и установления корректных родительских отношений. На этот процесс предлагается ссылаться как на *обратное раскрытие*. Можно заметить, что идея обратного раскрытия в определенной степени перекликается с идей инвертированного поиска (т.е. поиска от цели) в планировании [67; 69; 76], несмотря на то, что это различные техники.

Реализация на концептуальном уровне Для воплощения предложенной идеи на каждой итерации основного цикла алгоритма поиска должны выполняться следующие шаги:

1. Выбрать среди множества несогласованных состояний наиболее перспективное состояние n для дальнейшего рассмотрения и выбрать среди множества его потенциальных родителей, $PARENTS(n)$, наиболее перспективного – $bpp(n)$ (от англ. best potential parent).
2. На основе рассмотрения перехода $bpp(n) \to n$ обновить стоимость достижения состояния n, т.е. $g(n)$, если такая стоимость ниже, чем ранее известная.

3. Проверить является ли теперь состояние n согласованным и, если да, то добавить его в множество потенциальных родителей для оставшихся несогласованных состояний.

Шаг 1 наиболее простой, на нём необходимо выбрать для рассмотрения состояние, которое наиболее вероятно приведет к скорейшему отысканию решения. Для этого, как в алгоритмах A* и SIPP, предлагается выбирать состояние с минимальным f-значением. Однако в отличие от упомянутых алгоритмов, f-значение определим следующим образом:

$$f(n) = g_{low}(n) + h(n) = g(bpp(n)) + h(bpp(n), n) + h(n) \tag{2.13}$$

Здесь $h(bpp(n),n)$ – эвристическая оценка стоимости пути из состояния $bpp(n)$ в состояние n. Как и ранее, предполагается, что эвристическая функция h является монотонной.

Заметим, что при таком способе определения f-значения, имеются два различных g-значения для состояния n: $g(n)$ – наилучшая стоимость пути от начального состояния до n, известная к текущей итерации алгоритма; $g_{low}(n)$ – оценка стоимости пути до n через текущего наиболее перспективного родителя, $bpp(n)$. При расчете g-значения состояния используется полная последовательность корректных переходов от начального состояния, а при расчете g_{low}-значения – неполная, т.к. стоимость последнего перехода лишь аппроксимируется с помощью эвристической функции (но не является гарантированно точной). Изначально $g_{low}(n) = h(n_{start}, n)$ для всех n, для которых $los(n_{start}.v, n.v) = true$, $g_{low}(n_{start}) \equiv 0$, для остальных состояний $g_{low} = \infty$.

На Шаге 2 рассматривается переход $bpp(n) \rightarrow n$ и происходит вызов процедуры расчета минимального времени достижения n из $bpp(n)$, GetMinDelay[2], которая раньше вызывалась при раскрытии состояния и генерации состояний-потомков. Если переход возможен и новое время достижения состояния n меньше, чем ранее известное, т.е. $g(bpp(n)) + \delta + w(e) < g(n)$[3], то g-значение состояния n соответствующим образом обновляется (приравнивается к значению левой части неравенства).

[2]Описание этой процедуры приведено в предыдущем разделе, когда описывался стандартный алгоритм SIPP. Псевдокод процедуры приведен на (Рис. 2.6).

[3]Здесь $w(e)$ – это стоимость перехода по ребру, соединяющему вершины, а δ – это минимальная задержка, рассчитанная с помощью функции GetMinDelay.

Шаг 3 заключается в проверке, является ли теперь g-значение состояния n минимально возможным. Если да, то кратчайший путь до n найден и это состояние становится согласованным. Следовательно оно может быть добавлено в качестве потенциального родителя для несогласованных состояний n', таких что $los(n.v, n'.v) = true$. Более подробно о том, как именно реализуется Шаг 3 будет сказано далее.

Псевдокод Будем называть описываемый метод поиска (и алгоритм его реализующий) `TO-AA-SIPP` (от англ. Time-Optimal Any-Angle Safe Interval Path Planning). Для удобства восприятия разобьем псевдокод алгоритма на два последовательно выполняющихся блока – инициализация и основной цикл алгоритма. Приведем сначала псевдокод процедуры инициализации – см. Рис. 2.12, а затем и основного цикла – см. Рис. 2.13.

В строках 2–5 процедуры инициализации создаются все состояния поиска и добавляются в список OPEN. Далее в этом списке ищется начальное состояние (строка 6), его g-значение приравнивается к 0, и это состояние извлекается из OPEN и добавляется в список CLOSED, становясь его единственным элементом. В строках 10–16 для каждого состояния n из OPEN проверяется возможность перехода в него из начального состояния (строка 11). Если переход возможен, то происходит инициализация его g_{low}-значения, в результате которой оно становится конченым (строка 12). При этом начальное состояние добавляется в список потенциальных родителей PARENTS(n), оно же становится наиболее перспективным родителем n (строки 13–14). Если же переход не возможен, то список PARENTS(n) полагается пустым, g_{low}-значение – равным бесконечности, а наиболее перспективный родитель n – неизвестным (строка 16). В строках 17–19 происходит инициализация оставшихся атрибутов состояния n: f-значения, g-значения и родителя.

Итак, в результате инициализации созданы все состояния поиска и заполнены их атрибуты. Эти состояния разбиты на два списка, OPEN и CLOSED, при этом список CLOSED содержит лишь начальное состояние (т.к. только оно является согласованным в текущий момент).

Поскольку набор атрибутов, характеризующих состояние поиска n предлагаемого алгоритма, отличается от ранее описываемых алгоритмов поиска, приведем здесь полный список этих атрибутов:

Процедура `TO-AA-SIPP-Init`($\mathcal{G}$, v_{start}, los, $\{T_{safe}(v)\}$, h)**:**

Входные данные: граф $\mathcal{G}$, начальная вершина v_{start}, функция, определяющая возможность переходов los, множества безопасных интервалов для вершин графа $\{T_{safe}(v)\}$, монотонная эвристическая функция h

Выходные данные: Все состояния пространства поиска, разбитые на списки $OPEN$ и $CLOSED$

1 $OPEN \leftarrow \varnothing$; $CLOSED \leftarrow \varnothing$
2 **foreach** $v \in \mathcal{G}$ **do**
3 **foreach** $[t_l, t_u] \in T_{safe}(v)$ **do**
4 $n \leftarrow$ `GenerateSearchNode`$(v,[t_l, t_u])$
5 $OPEN \leftarrow OPEN \cup \{n\}$
6 $n_{start} \leftarrow$ `FindNode`$(OPEN, v_{start}, SI_{first}(v_{start}))$
7 $g(n_{start}) \leftarrow 0$; $g_{low}(n_{start}) \leftarrow 0$
8 $OPEN \leftarrow OPEN \setminus \{n_{start}\}$
9 $CLOSED \leftarrow CLOSED \cup \{n_{start}\}$
10 **foreach** $n = (v, [t_l, t_u]) \in OPEN$ **do**
11 **if** $los(n.v, n_{start}.v) = true$ **then**
12 $g_{low}(n) \leftarrow h(n_{start},n)$
13 $bpp(n) \leftarrow n_{start}$
14 $PARENTS(n) \leftarrow \{n_{start}\}$
15 **else**
16 $g_{low}(n) \leftarrow \infty$; $bpp(n) \leftarrow$`null`
17 $PARENTS(n) \leftarrow \varnothing$
18 $f(n) \leftarrow g_{low}(n) + h(n)$
19 $g(n) \leftarrow \infty$
20 $parent(n) \leftarrow$ `null`
21 **return** $OPEN$, $CLOSED$

Рисунок 2.12 — Процедура инициализации алгоритма TO-AA-SIPP.

– $g(n)$ – g-значение состояния, вес пути из начального состояния в текущее, известный к текущему моменту (текущей итерации алгоритма поиска);

- $parent(n)$ – состояние-родитель в дереве поиска, с помощью которого рекурсивно может быть восстановлен путь (с весом $g(n)$) из начального состояния в текущее;
- $h(n)$ – h-значение состояния, эвристическая оценка веса пути из текущего состояния в целевое;
- $PARENTS(n)$ – список согласованных состояний, являющихся потенциальными родителями для n, т.е. таких согласованных состояний n', для которых $los(n',n) = true$;
- $bpp(n)$ – наилучший потенциальный родитель n, т.е. согласованное состояние n', т.ч. сумма $g(n') + h(n',n)$ минимальна (по всем возможным потенциальным родителям);
- $g_{low}(n)$ – g_{low}-значение состояния, величина, рассчитываемая по формуле $g(bpp(n)) + h(bpp(n), n)$, т.е. вес пути до наилучшего родителя плюс эвристическая оценка веса пути от него до непосредственно n;
- $f(n)$ – f-значение состояния, величина рассчитываемая по формуле 10, использующаяся для определения порядка рассмотрения состояний.

После того как инициализация завершена выполняется основной цикл алгоритма `TO-AA-SIPP`, представленный на Рис. 2.13.

В начале цикла (строка 2) выбирается наиболее перспективное состояние поиска для рассмотрения, т.е. состояние с минимальным f-значением в OPEN. Оно извлекается из OPEN (строка 2) и его лучший потенциальный родитель, $bpp(n)$, извлекается из списка PARENTS(n), если он там находится (строки 3–4). В строке 5 происходит оценка перехода из наилучшего родителя в текущее состояние. Под оценкой здесь подразумевается расчет минимально возможного времени достижения, который ранее выполнялся на этапе создания создания состояний-потомков – см. строки 4 и 7 процедуры `GenerateAASIPPSuccesors` (Рис. 2.10). Рассчитанное время достижения n из $bpp(n)$ сохраняется в переменную g_new. Если это время достижения лучше (меньше), чем известное ранее (строка 6)[4], то g-значение соответствующим образом обновляется, и лучший потенциальный родитель, $bpp(n)$ становится теперь настоящим родителем, $parent(n)$. Таким образом, если строка 7 выполнена, то нам известен корректный путь из начального состояния до состояния n, стоимость которого равняется $g(n)$. Дальнейшие строки основного цикла, строки 8–24, предназна-

[4]Если переход не возможен с точки зрения заданных безопасных интервалов, то g_new равно ∞ и проверка в строке 6 всегда возвращает $false$.

Процедура `TO-AA-SIPP-Main`**:**

```
1  while min_{n∈OPEN} f(n) < ∞ do
2      n ← arg min_{n∈OPEN} f(n); OPEN ← OPEN \ {n}
3      if bpp(n) ∈ PARENTS(n) then
4          PARENTS(n) ← PARENTS(n) \ {bpp(n)}
5      g_new ← ValidateTransition(n, bpp(n))
6      if g_new < g(n) then
7          g(n) ← g_new; parent(n) ← bpp(n)
8      if SetBestPotentialParent(n) then
9          OPEN ← OPEN ∪ {n}
10         continue
11     if g(n) + h(n) ⩽ min_{n∈OPEN} f(n) then
12         CLOSED ← CLOSED ∪ {n}
13         if n.v = v_goal and g(n) ∈ SI_last(v_goal) then
14             return ReconstructPath(n)
15         foreach n' ∈ OPEN do
16             if los(n',n) = true then
17                 PARENTS(n') ← PARENTS(n') ∪ {n}
18                 if g(n) + h(n,n') < g_low(n') then
19                     g_low(n') ← g(n) + h(n,n')
20                     bpp(n') ← n
21                     f(n') ← g_low(n') + h(n')
22                     Update n' in OPEN
23     else
24         OPEN ← OPEN ∪ {n}
25 return failure
```

Рисунок 2.13 — Основной цикл алгоритма `TO-AA-SIPP`.

чены для выбора нового наиболее перспективного родителя n и определения, является ли это состояние согласованным.

Поиск нового наилучшего потенциального родителя осуществляется в строке 8 с помощью вспомогательной функции `SetBestPotentialParent`. Её

код представлен на Рис. 2.14. Она проверяет есть ли в списке потенциальных родителей такое состояние n', с помощью которого состояние n может быть потенциально достигнуто в более ранний временной момент (нежели $g(n)$). Если да, то n' становится лучшим потенциальным родителем, если нет, то $bpp(n)$ становится равным $parent(n)$, т.е. тому состоянию поиска, через которое рассчитано текущее g-значение состояния n. Последнее обстоятельство объясняет почему в строке 3 основного цикла алгоритма выполняется проверка условия $bpp(n) \in PARENTS(n)$: не исключена ситуация, что $bpp(n) = parent(n)$ и не находится в списке PARENTS(n), следовательно операция удаления $bpp(n)$ из этого списка не корректна.

Процедура `SetBestPotentialParent(`n`):`

```
1   g_low(n) ← g(n); bpp(n) ← parent(n)
2   f(n) ← g_low(n) + h(n)
3   BPPFound ← false
4   foreach n' ∈ PARENTS(n) do
5       if g(n') + h(n',n) < g_low(n) then
6           bpp(n) ← n'
7           g_low(n) ← g(n') + h(n',n)
8           f(n) ← g_low(n) + h(n)
9           BPPFound ← true
10  return BPPFound
```

Рисунок 2.14 — Определение наилучшего потенциального родителя для состояния поиска n в алгоритме `TO-AA-SIPP`.

Вернёмся к основному циклу алгоритма. Если в строке 8 найден новый потенциальный родитель для рассматриваемого состояния n, то это состояние помещается обратно в список OPEN и происходит переход к следующей итерации основного цикла (строки 9–10). Если же нового потенциального родителя (среди состояний, принадлежащих PARENTS(n)) найти не удалось, то выполняется проверка в строке 11. Выполнение условия этой проверки, т.е. неравенства $g(n)+h(n) \leqslant \min_{n \in OPEN} f(n)$, означает, что в списке OPEN нет ни одного состояния, которое бы потенциально могло уменьшить стоимость пути из начального состояния в текущее. Таким образом, можно заключить, что состояние n – со-

гласованно (т.е. известен наилучший путь до него). Формальное доказательство этого утверждения будет дано позже.

Итак, если состояние n стало согласованным, то оно добавляется в список CLOSED (строка 12) и происходит проверка на достижение цели (строка 13). Если согласованным стало целевое состояние (условие в строке 13 выполняется), то алгоритм завершает свою работу и возвращает путь, восстановленный с помощью родительских указателей (строка 14). Если нет, то текущее согласованное состояние добавляется в качестве потенциального родителя (т.е. добавляется в список PARENTS) для всех состояний n' из OPEN, переход в которые из n возможен согласно функции los. При этом если текущее g_{low}-значение состояния n' больше, чем $g(n) + h(n,n')$, то состояние n становится новым наиболее перспективным потенциальным родителем для n', соответствующим образом обновляются g_{low}- и f-значения состояния n', и оно обновляется в списке OPEN (строки 18–22).

Строка 24 основного цикла выполняется, если условие в строке 11 не выполнено, т.е. если g-значение состояния n потенциально может быть уменьшено за счет переходов из других состояний, принадлежащих OPEN. В этом случае состояние n не считается согласованным, не добавляется в CLOSED, а возвращается обратно в OPEN (строка 24).

Алгоритм `TO-AA-SIPP` прекращает свою работу (выходит из основного цикла) при выполнении следующих условий. Либо на очередной итерации установлено, что целевое состояние стало согласованным и тогда возвращается искомый путь (строка 14), либо список OPEN содержит лишь состояния, f-значения которых не являются конечными (условие в строке 1). В этом случае алгоритм возвращает специальный символ $failure$ (строка 25), указывая на то, что искомый путь найти не удалось.

Теоретические свойства алгоритма `TO-AA-SIPP`

Докажем основные свойства предложенного алгоритма решения задачи AA-PFD, а именно тот факт, что алгоритм `TO-AA-SIPP` является *полным*, т.е. гарантирует отыскание решения для любой корректно определенной задачи AA-PFD, имеющей решение, и при этом корректно завершается, возвращая спе-

циальный символ $failure$, если входная задача не имеет решения; а тот факт, что решение, возвращаемое алгоритмом является оптимальным, т.е. найденный путь является минимальным по стоимости (целевая вершина достигается в минимально возможный момент времени). Дополнительно, установим временную сложность алгоритма.

Лемма 1. *Для любого состояния поиска n либо $g_{low}(n) = g(bpp(n)) + h(bpp(n),n)$, либо $g_{low}(n) = g(n)$.*

Доказательство. Докажем по индукции что перед любой итерацией основного цикла алгоритма указанное соотношение имеет место.

$k = 1$ Перед первой итерацией алгоритма для начального состояния n_{start} имеем $g_{low}(n_{start}) = g(n_{start}) = 0$ – см. строку 7 процедуры инициализации `TO-AA-SIPP-Init`. Т.е. для начального состояние утверждение леммы выполняется.

Для любого другого состояния поиска n, не являющегося начальным, но такого что $los(n_{start}, n) = true$, g_{low}-значение задается в строке 12 процедуры `TO-AA-SIPP-Init` равным $h(n_{start},n)$, а $bpp(n)$ полагается равным n_{start} (строка 13 этой же процедуры). Таким образом, $g_{low}(n) = h(n_{start},n) = 0 + h(n_{start},n) = g(n_{start}) + h(n_{start},n) = g(bpp(n)) + h(bpp(n),n)$. То есть доказываемое утверждение для таких состояний выполняется.

Рассмотрим теперь оставшиеся состояния поиска, т.е. состояния n, не являющиеся начальным, для которых не выполнено $los(n_{start}, n) = true$. Для них в строке 16 процедуры инициализации `TO-AA-SIPP-Init` выполняется приравнивание $g_{low}(n) \leftarrow \infty$, а в строке 18 $g(n) \leftarrow \infty$. Таким образом, $g_{low}(n) = g(n)$, т.е. утверждение Леммы для таких состояний выполняется.

Итак, перед 1-й итерацией основного цикла алгоритма поиска утверждение Леммы выполнено.

$k = i$ Пусть условие Леммы выполнено перед i-й итерацией основного цикла алгоритма.

$k = i + 1$ Докажем теперь, что Лемма выполняется и перед $i + 1$ итерацией основного цикла алгоритма. Для этого заметим, что g_{low}-значение состояния n – текущего рассматриваемого состояния поиска, – изменяется лишь внутри вызова процедуры `SetBestPotentialParent`. В строке 1 этой процедуры выполняется приравнивание $g_{low}(n) \leftarrow g(n)$, т.е. после этой строки утверждение

Леммы выполняется. Далее, если условие в строке 5 выполняется, то осуществляются приравнивания (строки 6–7): $bpp(n) \leftarrow n'$ и $g_{low}(n) \leftarrow g(n') + h(n',n)$. Т.е. $g_{low}(n) \leftarrow g(bpp(n)) + h(bpp(n), n)$. То есть утверждение Леммы выполняется. Итак, после выполнения процедуры `SetBestPotentialParent`, вызываемой в строке 8 основного цикла алгоритма поиска, утверждение Леммы выполнено. Более в основном цикле g_{low}-значение состояния n – текущего рассматриваемого состояния поиска, – нигде не меняется.

Заметим, что в строках 19–20 основного цикла алгоритма может измениться g_{low}-значение состояния n' отличного от текущего состояния n. А именно могут быть выполнены следующие присваивания. $bpp(n') \leftarrow n$ и $g_{low}(n') \leftarrow g(n) + h(n,n')$. Таким образом $g_{low}(n') \leftarrow g(bpp(n')) + h(bpp(n'), n)$. То есть и для состояния n' после этих присваиваний утверждение Леммы выполнено.

Таким образом, на i-й итерации цикла выполняемые присваивания и изменения g_{low}-значений состояний поиска не нарушают выполнения утверждения Леммы. Следовательно, перед $i+1$ итерацией основного цикла алгоритма утверждение Лемма выполняется. ■

Лемма 2. *Для любого состояния поиска n в всегда выполняется $g_{low}(n) \leqslant g(n)$.*

Доказательство. После выполнения процедуры инициализации это соотношение верно для всех имеющихся состояний поиска n, т.к. для начального состояния $g_{low}(n) = g(n) = 0$ (строка 7 процедуры `TO-AA-SIPP-Init`), а для всех остальных состояний $g(n) = \infty$ (строка 18 той же процедуры).

Далее, в основном цикле алгоритма, `TO-AA-SIPP-Init`, g-значение рассматриваемого состояния n может быть изменено только в строке 7. В этой строке $g(n)$ приравнивается к величине g_new, которая после выполнения строки 5 (функции `ValidateTranstion`) равна $g(bpp(n)) + \delta + w(bpp(n),n)$. Здесь δ – минимальная задержка, вычисляемая по описанной раннее процедуре `GetMinDelay` (Рис. 2.6), а $w(bpp(n),n)$ – стоимость (продолжительность) перехода из $bpp(n)$ в n. Заметим, что $\delta + w(bpp(n),n) \geqslant h(bpp(n), n)$, т.к. в алгоритме используется допустимая, монотонная эвристика. Таким образом, $g\_new > g(bpp(n) + h(bpp(n),n)$. В то же время по Лемме 1 $g_{low}(n) = g(bpp(n)) + h(bpp(n),n)$ либо $g_{low}(n) = g(n)$. Следовательно $g_{low}(n) \leqslant g(n)$ после выполнения строки 7 основного цикла. Более нигде в основном цикле g-значение состояния n измениться не может.

Однако, в строке 8 может измениться g_{low}-значение состояния n, в рамках процедуры выбора наилучшего потенциального родителя `SetBestPotentialParent`. При этом в строке 1 этой процедуры выполняется приравнивание $g_{low}(n) \leftarrow g(n)$, т.е. неравенство $g_{low}(n) \leqslant g(n)$ продолжает выполняться. Далее в этой процедуре $g_{low}(n)$ может лишь уменьшится (строки 5–6), но $g(n)$ при этом не меняется. Таким образом выполнение процедуры `SetBestPotentialParent` в строке 8 основного цикла алгоритма не нарушает выполнения неравенства $g_{low}(n) \leqslant g(n)$. После этой строки ни g-значение состояния n, ни его g_{low}-значение не изменяются.

Осталось обратить внимание на то, что в строке 19 основного цикла алгоритм поиска может обновиться g_{low}-значение состояния n' (не того, что было извлечено из OPEN в начале итерации), но при этом оно может лишь уменьшится (см. условие обновления $g_{low}(n')$ в строке 18), в то время как g-значение этого состояния не изменяется. Это означает, что и для состояния n' утверждение Леммы выполняется. ■

Лемма 3. *Для любого i выполняется $g^i_{low}(n) \leqslant g_{low}(n)$, где $g^i_{low}(n)$ – это g_{low}-значение состояния n извлеченного из списка OPEN в начале i-й итерации основного цикла алгоритма поиска, а $g_{low}(n)$ – это g_{low}-значение состояния n в конце i-й итерации.*

Доказательство. Пусть в начале очередной итерации алгоритма поиска, в строке 2 процедуры `TO-AA-SIPP-Main`, из списка OPEN извлечено состояние n с минимальным f-значением. По определение его g_{low}-значение в этот момент равно $g^i_{low}(n)$, а f-значение равно $f(n) = g^i_{low}(n) + h(n)$ (см. определение f-значения в формуле 10).

Единственное место где g_{low}-значение состояния n может измениться – это внутри вызова процедуры `SetBestPotentialParent` (в строке 8). В 1-й строке этой процедуры выполняется присваивание $g_{low}(n) \leftarrow g(n)$, т.е. g_{low}-значение может лишь увеличиться или остаться таким же (по Лемме 2). Таким образом, если далее g_{low}-значение состояния n более не меняется, то утверждение Леммы выполняется.

Допустим теперь, что g_{low}-значение состояния n изменяется. Это может произойти в строке 7 процедуры `SetBestPotentialParent`. Докажем от обратного, что новое g_{low}-значение состояния n, обозначаемое как $g'_{low}(n)$, больше либо равно $g^i_{low}(n)$.

Допустим, что $g'_{low}(n) < g^i_{low}(n)$. Это означает, что $f'(n) = g'_{low}(n)+h(n) < f(n) = g'_{low}(n) + h(n)$. Т.е. на текущей итерации основного цикла алгоритма поиска среди состояний находящихся в списке PARENTS(n), нашлось такое состояние n', f-значение которого меньше, чем $f(n)$. Следовательно, с этим f-значением, равным $g'_{low}(n) + h(n)$ состояние n должно было быть извлечено из OPEN на текущей итерации поиска, т.к. (см. строку 2 процедуры `TO-AA-SIPP-Main`) извлекается состояние с минимальным f-значением, что ведет к противоречию. Либо же состояние n с этим f-значением должно было извлечено из OPEN на одной из предыдущих итераций поиска (ведь в начале каждой итерации основного цикла из списка OPEN извлекается состояние минимизирующее f-значение). Однако в этом случае состояние n' должно было быть удалено из списка PARENTS(n) (в строке 4 основного цикла), что ведет к противоречию. ■

Лемма 4. *Последовательность* $\{f^1_{min}, f^2_{min}, ..., f^K_{min}\}$, *где* f^i_{min} *есть* f*-значение состояния поиска, извлеченного из OPEN на* i*-й итерации основного цикла алгоритма поиска, является невозрастающей последовательностью. Т.е.* $f^1_{min} \geqslant f^2_{min} \geqslant ... \geqslant f^K_{min}$.

Доказательство. Заметим, что все состояния поиска создаются на этапе инициализации с помощью процедуры `TO-AA-SIPP-Init` и далее внутри основного цикла алгоритма поиска новые состояния не создаются. Далее заметим, что на очередной итерации основного цикла (обозначим ее как i) из списка OPEN извлекается единственное состояние, а именно состояние с минимальным f-значением, f^i_{min}, и это состояние может быть либо добавлено обратно в OPEN (строки 9, 24), либо добавлено в CLOSED (строка 12), при этом f-значения некоторых состояний поиска (отличных от текущего рассматриваемого) могут быть обновлены (строки 21–22). Рассмотрим эти три случая по отдельности и покажем, что в любом из них утверждение Леммы выполняется.

Случай 1 (строки 8–10). В этом случае процедура `NewBestPotentialParent` вернула *true*, т.е. g_{low}-значение состояния n было изменено. Это значение по Лемме 3 не может быть меньше, чем g_{low}-значение, которое использовалось для расчета f^i_{min} в начале итерации основного цикла. Это означает, что n заново добавляется в OPEN с f-значением, которое больше либо равно f^i_{min}. Следовательно список OPEN содержит состояния поиска, f-значения которых не

меньше f^i_{min}, что, в свою очередь означает, что на следующей итерации будет извлечено состояния с f-значением f^{i+1}_{min}, для которого выполняется $f^{i+1}_{min} \geqslant f^i_{min}$.

Случай 2 (строка 24). В этом случае процедура `SetBestPotentialParent` вернула $false$, что означает, что в списке потенциальных родителей PARENTS(n) нет состояния, которое бы потенциально могло уменьшить g-значение состояния n. При этом в ходе выполнения этой процедуры (в строке 1) выполнено присваивание $g_{low}(n) \leftarrow g(n)$, т.е. g_{low}- и g-значения для рассматриваемого состояния теперь совпадают, а f-значение состояния теперь равно (строка 2 процедуры `SetBestPotentialParent`) $f(n) = g_{low}(n) + h(n) = g(n) + h(n)$.

Для того, чтобы строка 24 основного цикла выполнилась, проверка условия $g(n) + h(n) \leqslant \min_{n \in OPEN} f(n)$ в строке 11 должна вернуть $false$. Это означает, что $g(n) + h(n) = f(n) \geqslant f_{min} \geqslant f^i_{min}$, где $f_{min} = \min_{n \in OPEN} f(n)$ – это минимальное f-значение по состояниям поиска, находящимся в OPEN в данный момент выполнения основного цикла алгоритма поиска. Таким образом в строке 24 состояние n будет добавлено в OPEN с f-значением, которое больше, чем минимальное f-значение среди вершин, находящихся в OPEN в данный момент. Следовательно на следующей итерации поиска, $i+1$, из списка OPEN будет извлечено состояние с минимальным f-значением, f^{i+1}_{min}, которое больше либо равно f_{min}, которое в свою очередь больше либо равно f^i_{min}, т.е. $f^{i+1}_{min} \geqslant f_{min} \geqslant f^i_{min}$.

Случай 3 (строки 11–22). В этом случае текущее рассматриваемое состояние n помещается в CLOSED и, следовательно, не может быть извлечено из OPEN на последующих итерациях основного цикла. При этом, однако, g-значение этого состояния может быть использовано (см. строку 19) для обновления g_{low}-значения некоторого другого состояния n'. Это в свою очередь влечет изменение $f(n')$ (строка 21) и обновление n' в OPEN. Таким образом, необходимо доказать, что $f(n') \geqslant f^i_{min}$.

Заметим, что после выполнения строк 19–21 $f(n') = g_{low}(n') + h(n') = g(n) + h(n,n') + h(n')$. Поскольку эвристическая функция является монотонной, справедливо неравенство $h(n,n') + h(n') \geqslant h(n)$. То есть $f(n') \geqslant g(n) + h(n)$. По Лемме 2 $g(n) \geqslant g_{low}(n)$. Следовательно $f(n') \geqslant g_{low}(n) + h(n) = f^i_{min}$. Доказательство для рассматриваемого случая завершено.

Итак, были рассмотрены все возможные случаи обновления f-значений состояний поиска в OPEN на i-й итерации основного цикла и показано, что в

результате этих обновлений ни одно f-значение в OPEN не становится меньше, чем f^i_{min}. Таким образом на следующей итерации из OPEN будет извлечено состояние, f-значение которого больше либо равно f-значению состояния, извлеченного из OPEN на текущей итерации. То есть $f^{i+1}_{min} \geqslant f^i_{min}$. ■

Лемма 5. *Список* CLOSED *содержит состояния поиска, g-значение которых не может быть уменьшено, т.е. такие состояния до которых известен путь минимальной стоимости.*

Доказательство. Докажем по индукции, что перед каждой итерацией основного цикла алгоритма поиска утверждение Леммы выполнено.

$k = 1$ Перед первой итерацией основного цикла список CLOSED содержит лишь одно состояние, которое было в него добавлено на этапе инициализации (процедура `TO-AA-SIPP-Init`). Это – начальное состояние, g-значение которого равно 0. То есть утверждение Леммы, очевидно, выполнено.

$k = i$ Пусть утверждение Леммы выполнено перед i-й итерацией основного цикла алгоритма.

$k = i+1$ Докажем теперь, что Лемма выполняется и перед $i+1$ итерацией основного цикла алгоритма. Для этого, необходимо доказать, что если в список CLOSED добавлялись состояния поиска на i-й итерации, то только такие, для которых g-значение не может быть уменьшено.

Для начала заметим, что на i-й итерации основного цикла в CLOSED может добавиться лишь одно состояние (строка 12), а именно текущее рассматриваемое состояние n, которое было извлечено из OPEN в строке 2. Пусть это состояние добавлено в CLOSED. Необходимо доказать, что его g-значение не может быть уменьшено, т.е. среди всех состояний поиска не найдется состояний, переход в n из которых может уменьшить текущее $g(n)$.

Отметим, что поскольку n добавлено в CLOSED в 12-й строке основного цикла алгоритма поиска, процедура `SetBestPotentialParent`, вызываемая в строке 8, вернула $false$ (иначе был бы совершен переход к следующей итерации в строке 10). Это означает, что в ходе рассмотрения потенциальных родителей из списка PARENTS(n) (строки 4–9 процедуры `SetBestPotentialParent`) не было обнаружено ни одного состояния n', для которого бы выполнялось (строка 5 этой же процедуры) неравенство $g(n') + h(n',n) < g_{low}(n)$, что равносильно (с учетом присваивания $g_{low}(n) \leftarrow g(n)$ в строке 1 процедуры) неравенству $g(n') + h(n',n) < g(n)$. Заметим теперь, что $g(n')$ не может уменьшиться, т.к. n'

было добавлено в PARENTS(n) лишь одновременно с добавлением в CLOSED (строки 12–17 основного цикла алгоритма) на одной из предыдущих итераций, а по индуктивному предположению g-значения всех состояний в CLOSED перед i-й (текущей рассматриваемой) итерацией не могут более быть уменьшены. Дополнительно заметим, что поскольку эвристика является монотонной, то $h(n',n) \geqslant w(n',n)$, где $w(n',n)$ – это реальная стоимость перехода из n' в n. Таким образом, на основе вышесказанного можно сделать вывод – ни одно из состояний, входящих в PARENTS(n), не может быть использовано для уменьшения $g(n)$.

Итак, мы показали, что часть состояний из списка CLOSED, а именно состояния, входящие в список PARENTS(n), не могут уменьшать g-значения состояния n, которое сейчас (на i-й итерации основного цикла, в строке 12) добавляется в CLOSED. Рассмотрим оставшиеся состояния в списке CLOSED. Они могут быть разбиты на два подмножества, $S1$ и $S2$. В множество $S1$ входят состояния, переход из которых в n не возможен, т.е. $\forall n_{s_1} \in S1 : \ los(n_{s_1},n) = false$. Очевидно они не могут уменьшить $g(n)$. В $S2$ входят оставшиеся состояния, т.е. такие, что $los(n_{s_2},n) = true$ и при этом $n_{s_2} \notin PARENTS(n)$. Заметим, что тогда любое состояние n_{s_2} из $S2$ обязано было входить в PARENTS(n) на одной из предшествующих итераций основного цикла, т.к. любое состояние добавляемое в CLOSED добавляется и в список PARENTS для всех состояний, переход до которых возможен (функция los на которых возвращает $true$) – строка 17 основного цикла. Это означает, что возможность перехода из n_{s_2} в n и уменьшения $g(n)$ за счет этого перехода уже была рассмотрена ранее, следовательно, эти состояния не могут более уменьшить $g(n)$.

Итак, было показано, что ни одно из состояний, входящих в список CLOSED, не может быть использовано для уменьшения g-значения состояния n (которое добавляется в CLOSED на текущей итерации основного цикла в строке 12).

Остается доказать, что ни одно из состояний, входящих в OPEN, так же не может уменьшить $g(n)$.

Для этого в начале обратим внимание на условие в строке 11 основного цикла. Оно выполнено (т.к. иначе речь о добавлении n в CLOSED в строке 12 не шла бы). Перепишем это условие в виде

$$g(n) + h(n) \leqslant f(n_{best}) = g_{low}(n_{best}) + h(n_{best}),$$

где $n_{best} = arg\min_{n \in \text{OPEN}} f(n)$ – это состояние в OPEN, которое в данный момент обладает наименьшим f-значением.

Рассмотрим произвольное состояние $n' \in$ OPEN. Для него, очевидно, справедливо следующее неравенство: $f(n') = g_{low}(n') + h(n') \geqslant f(n_{best}) = g_{low}(n_{best}) + h(n_{best})$. В силу того, что эвристика является монотонной, для любых состояний поиска n и n' справедливо неравенство $h(n') \leqslant h(n, n') + h(n)$. Таким образом, имеем следующую цепочку неравенств:

$$\begin{aligned} g(n) + h(n) &\leqslant g_{low}(n_{best}) + h(n_{best}) \\ &\leqslant g_{low}(n') + h(n') \\ &\leqslant g_{low}(n') + h(n, n') + h(n). \end{aligned}$$

То есть:

$$g(n) + h(n) \leqslant g_{low}(n') + h(n, n') + h(n).$$

Сокращая левую и правую части на $h(n)$, имеем:

$$g(n) \leqslant g_{low}(n') + h(n, n').$$

Таким образом, состояние $n' \in$ OPEN не может быть использовано на текущей итерации, чтобы уменьшить $g(n)$ а в силу Леммы 4 состояние n' не сможет быть использовано, чтобы уменьшить $g(n)$ и на любой последующей итерации. Поскольку n' было выбрано из OPEN произвольно, это означает что *ни одно* состояние в OPEN не может быть использовано, чтобы уменьшить $g(n)$.

При этом ранее было показано, что ни одно состояние $n' \in$ CLOSED, $n' \neq n$ не может быть использовано, чтобы уменьшить $g(n)$.

Таким образом, g-значение состояния n добавляемого в строке 12 в CLOSED является минимально-возможным. ■

Важным следствием Леммы 5 является следующая Теорема об оптимальности решения, отыскиваемого алгоритмом `TO-AA-SIPP`.

Теорема 1. *Решение задачи* AA-PFD*, возвращаемое алгоритмом* `TO-AA-SIPP`*, является оптимальным.*

Доказательство. Алгоритм возвращает решение в строке 14, после проверки того, что а) текущее рассматриваемое состояние соответствует целевому, б) текущее рассматриваемое состояние добавлено в CLOSED, следовательно (см.

Лемму 5) g-значение этого состояния не может быть уменьшено (пути с более низкой стоимостью до целевого состояния не существует). ■

Продолжим исследование свойств алгоритма `TO-AA-SIPP`. Докажем следующие леммы.

Лемма 6. *Когда состояние поиска n, такое что $bpp(n) = parent(n)$, извлекается из* OPEN *на очередной итерации основного цикла, то это состояние будет добавлено в список* CLOSED *на этой же итерации.*

Доказательство. Пусть состояние n извлекается из OPEN в начале очередной итерации основного цикла (строка 2). Равенство из условия Леммы, $bpp(n) = parent(n)$, означает, что на одной из прошлых итераций основного цикла это состояние уже рассматривалось, для него вызывалась процедура `SetBestPotentialParent`, которая вернула *false*, т.к. присваивание $bpp(n) \leftarrow parent(n)$ происходит только внутри этой процедуры (строка 1). Поскольку процедура вернула *false*, то $g_{low}(n)$ в строке 7 этой процедуры ни разу не обновлялось, следовательно $g_{low}(n) = g(n)$ (см. строку 1 процедуры `SetBestPotentialParent`) и $f(n) = g_{low}(n) + h(n) = g(n) + h(n)$ (см. строку 2 этой же процедуры).

Итак, на текущей итерации основного цикла, в момент извлечения состояния n из OPEN его g_{low}- и g-значения совпадают. В строке 5 будет оценен переход из $bpp(n)$ в n, то есть по факту из $parent(n)$ в n. Следовательно, после выполнения строки 5, имеем $g\_new = g(n)$ (т.к. $g(n)$ уже включает в себя фактическую стоимость перехода из $bpp(n) = parent(n)$ в n); условие в строке 6 не выполняется и $g(n)$ в строке 7 не обновляется. Т.е. к строке 8 основного цикла имеем: $bpp(n) = parent(n)$, $g(n) = g_{low}(n)$. Вызов процедуры `SetBestPotentialParent` вернет *false*, т.к. если бы эта функция вернула *true*, то это означало бы, что $g_{low}(n)$ уменьшилось, что противоречит Лемме 3.

Таким образом алгоритм перейдёт к строке 11. Проверка в этой строке условия $g(n) + h(n) \leqslant \min_{n \in \text{OPEN}} f(n)$ вернет *true*, т.к. $g(n) = g_{low}(n)$ и это значение не менялось с начала итерации. Следовательно, будет выполнена строка 12 и состояние n будет добавлено в список CLOSED. ■

Лемма 7. *Если в основном цикле опустить проверку на достижение целевого состояния в строке (а именно убрать строки 13–14 из процедуры*

`TO-AA-SIPP-Main`*), то алгоритм завершит свою работу за конечное число итераций.*

Доказательство. Заметим, во-первых, что все возможные состояния поиска создаются на этапе инициализации (процедура `TO-AA-SIPP-Init`) и далее в основном цикле никакие состояния не создаются и не удаляются. Далее заметим, что любое состояние поиска может быть добавлено в список CLOSED только один раз (т.к. извлечение состояния из списка CLOSED не предусмотрено ни в одной строке ни одной из процедур, образующих алгоритм). Следовательно, это состояние может быть добавлено в список PARENTS для других состояний лишь однажды. Таким образом для произвольного состояния n список PARENTS(n) может содержать максимум $M-1$ елементов, где M – это общее число состояний поиска.

Далее, каждый раз когда состояние n извлекается из списка OPEN, список его потенциальных родителей сокращается на один элемент (на $bpp(n)$), если только этот список уже не пуст (строки 3–4 основного цикла). Таким образом, число итераций основного цикла, на которых произвольное состояние n извлекается из OPEN с непустым списком PARENTS конечно.

Покажем теперь, что после конечно числа извлечений произвольного состояния n из OPEN, это состояние либо оказывается в CLOSED, либо оказывается в OPEN с $f(n) = \infty$. Для этого рассмотрим два случая.

Случай 1 $g(n)$ не становится равным некоторому конечному числу каждый раз, когда n извлекается из OPEN. Это может быть только в том случае, если переход в n из каждого $bpp(n)$ невозможен (функция `ValidateTransition` в строке 5 возвращается *false*). Рассмотрим теперь итерацию основного цикла, когда n извлекается с пустым списком PARENTS (это обязательно произойдет, т.к. бесконечно извлекать n из OPEN и добавлять его обратно туда же, не сокращая при этом список PARENTS не получится, как показано выше). На этой итерации при вызове процедуры `SetBestPotentialParent` будут выполнены лишь первые 3 строки, в том числе присваивания: $g_{low}(n) \leftarrow g(n)(= \infty)$ и $f(n) \leftarrow g_{low}(n) + h(n) = \infty$. После этого процедура вернет *false*. Далее проверка в строке 11 основного цикла аналогично вернет *false*, т.к. левая часть выражения $g(n) + h(n) \leqslant \min_{n \in OPEN} f(n)$ равна ∞[5]. Следовательно в строке 24 основного цикла состояние n будет добавлено в OPEN с $f(n) = \infty$.

[5]Предполагается, что выражение $\infty \leqslant \infty$ интерпретируется как *false*.

Случай 2 $g(n)$ становится конечным на какой-то итерации основного цикла. Это означает, что родитель для n определен, т.е. выполнено присваивание $parent(n) \leftarrow bpp(n)$ в строке 7. Следовательно, он уже не сможет стать равным `null` ни на одной из последующих итераций (т.к. присваивание $parent(n) \leftarrow$ `null` выполняется для состояния n лишь на этапе инициализации и более нигде). Таким образом, даже если список PARENTS(n) иссякнет, либо же будет содержать состояния, которые не могут даже потенциально уменьшить g-значение (т.е. проверка условия в строке 5 процедуры `SetBestPotentialParents` возвращает *false*), то настанет итерация, когда состояние n будет извлечено из OPEN с $parent(n) = bpp(n)$. Следовательно, по Лемме 6, на этой итерации n будет добавлено с CLOSED и более не будет содержаться в OPEN (и извлекаться из этого списка на последующих итерациях основного цикла).

Таким образом, в результате рассмотрения обоих случаев показано, что каждое состояние поиска n, созданное на этапе инициализации, после конечного числа итераций основного цикла либо находится в CLOSED, либо находится в OPEN с $f(n) = \infty$. В любом из вариантов проверка условия в строке 1 основного цикла вернет *false*[6] и алгоритм завершит свою работу. ■

Прямым следствием Леммы 7 является следующая теорема, характеризующая вычислительную трудоемкость алгоритма `TO-AA-SIPP`.

Теорема 2. *Максимальное число итераций основного цикла алгоритма* `TO-AA-SIPP` *составляет* $M(M-1)$*, где* M *– это общее число состояний поиска.*

Доказательство. Лемма 7 устанавливает, что за конечное число итераций основного цикла алгоритм завершается. При этом, в рамках доказательства этой Леммы было установлено что, во-первых, мощность множества потенциальных родителей для произвольного состояния поиска n ограничена величиной $|M-1|$, во-вторых, число итераций основного цикла, на которых произвольное состояние n извлекается из OPEN конечно, в-третьих, каждый раз когда когда n извлекается из OPEN с непустым списком PARENTS из него удаляется один элемент. Таким образом произвольное состояние поиска n индуцирует максимум

[6]Мы подразумеваем, что оператор min на пустом множестве возвращает ∞ и, как и ранее, предполагаем, что выражение $\infty < \infty$ интерпретируется как *false*.

$|M-1|$ итераций основного цикла. Следовательно общее число таких итераций (индуцируемых различными состояниями поиска) равняется $M(M-1)$. ■

В литературе, посвященной алгоритмам эвристического поиска, обычно полагают, что временная сложность одной итерации основного цикла алгоритма есть $O(1)$. Таким образом, согласно Теореме 2, временная сложность алгоритма `TO-AA-SIPP` есть $O(M^2)$, где M – это число состояний поиска, которое, в свою очередь равняется числу безопасных интервалов по всем вершинам графа: $M = \sum_{i=1}^{|V|} |T_{safe}(v_i)|$. Здесь запись $|T_{safe}(v_i)|$ означает число безопасных интервалов вершины (а не мощность множества безопасных моментов времени, образующих эти интервалы).

Докажем теперь ещё одну лемму, которая понадобится далее.

Лемма 8. *Если в основном цикле опустить проверку на достижение целевого состояния в строке (а именно убрать строки 13–14 из процедуры* `TO-AA-SIPP-Main`*), то за конечное число итераций всякое достижимое из начальной вершины состояние будет добавлено в список* CLOSED*.*

Доказательство. Докажем Лемму от противного. Допустим некоторое достижимое состояние n не добавлено в CLOSED, а находится в OPEN, после того как алгоритм завершил свою работу (это гарантировано произойдет по Лемме 7). Это значит, что его f-значение бесконечно (т.к. если бы $f(n)$ было конечным, то проверка условия в строке 1 основного цикла не позволила бы завершить его выполнение). Это означает, что для состояния n не был найден ни один родитель (из числа других состояний поиска), переход из которого в n был бы возможен, и в результате такого перехода g-значение состояния n (а вместе с ним и f-значение) стало бы конечным. Тем не менее по условию Леммы как минимум одно такое состояние имеется (то, через которое достигается n по условию Леммы). Обозначим это состояние как n'. Единственной причиной, по которой достижимость n из n' оказалось неустановленной, может быть только то, что n' не было добавлено в ходе выполнения основного цикла алгоритма в список CLOSED, т.к. если бы n' попало в CLOSED, то тут же было бы добавлено в список потенциальных родителей для всех состояний из OPEN, в том числе и для n (и на одной из последующих итераций переход $n' \to n$ был бы рассмотрен и, т.к. этот переход по условию Леммы валиден, $g(n)$ и $f(n)$ стали бы конечными). Рекурсивно применяя эти рассуждения к состоянию n', приходим

к выводу, что начальное состояние поиска, n_{start}, не было добавлено в список CLOSED. Однако это не так, т.к. добавление n_{start} в CLOSED происходит на этапе инициализации (строка 9 процедуры `TO-AA-SIPP-Init`). Это противоречие и доказывает Лемму. ■

Прямым следствием Лемм 7 и 8 является следующая теорема.

Теорема 3. *Алгоритм* `TO-AA-SIPP` *является полным, т.е. гарантирует отыскание решения задачи* AA-PFD*, если оно существует, и гарантирует корректное завершение работы за конечное число итераций и возврат специального символа* failure *(решение не найдено), если решения задачи* AA-PFD *не существует.*

Доказательство. Если решение существует, то целевое состояние достижимо. Это значит (см. Лемму 8), что на одной из итераций основного цикла целевое состояние будет добавлено в список CLOSED, т.е. будет выполнена строка 12 основного цикла. Далее в строке 13 будет проведена проверка на то, что текущее состояние является целевым, эта проверка, очевидно, вернет *true*, и будет выполнена строка 14, т.е. с помощью родительских указателей будет реконструирован путь от начального состояния до целевого, и этот путь будет возвращен в качестве решения.

Если решения не существует, то целевое состояние недостижимо. Это означает, что проверка в строке 13 никогда не вернет *true*. Таким образом, можно считать, что проверка на достижение целевого состояния отсутствует. В этом случае по Лемме 7 будет выполнено конечно число итераций основного цикла, после чего будет выполнена строка 25 процедуры `TO-AA-SIPP-Main` – алгоритм вернет *failure* и завершит свою работу. ■

На этом описание теоретических свойств предложенного алгоритма решения задачи AA-PFD, `TO-AA-SIPP`, завершено.

2.2.4 Безопасно-интервальное планирование для эффективного поиска субоптимальных решений задачи AA-PFD

Предложенный выше алгоритм TO-AA-SIPP гарантирует отыскание оптимальных решений задачи AA-PFD, однако он обладает следующими ограничениями. Во-первых, алгоритм является достаточно ресурсоемким, т.к. его сложность квадратична относительно числа состояний поиска (которое линейно зависит от числа вершин в графе и количества безопасных интервалов). В худшем случае алгоритму необходимо рассмотреть все пары пространства состояний, при этом на практике процедура рассмотрения одной пары (т.е. выполнение одной итерации основного цикла TO-AA-SIPP) может быть весьма трудозатратной. Во-вторых, логика его работы достаточно нетривиальна и существенно отличается от логики работы стандартных алгоритмов эвристического поиска, обычно применяемых для поиска пути на ГРД. Это может затруднить практическую реализацию и использование TO-AA-SIPP произвольным пользователем. Таким образом, целесообразной представляется разработка и исследование методов, позволяющих получать допустимые задачи AA-PFD за меньшее время (по сравнению с TO-AA-SIPP), пусть и за счет отказа от гарантии оптимальности отыскиваемых решений. Желательно, однако сформулировать в том или ином виде общие оценки качества таких методов. Так же желательно, чтобы эти методы были основаны на стандартном для алгоритмов систематического поиска принципе последовательного рассмотрения состояний поиска и инкрементального построения дерева поиска. Именно разработке и исследованию подобных методов и посвящен этот раздел.

Безопасно-интервальное планирование с механизмом переназначения родителя в дереве поиска

Вспомним принцип работы алгоритма SIPP, предназначенного для решения задачи PFD, – см. Рис. 2.9. SIPP использует стандартный для алгоритмов эвристического поиска подход инкрементального построения дерева поиска. На каждой итерации из листов дерева, т.е. из списка OPEN, выбирается наиболее

перспективное состояние поиска, создаются состояния-потомки, которые затем, после ряда проверок и фильтраций, добавляются в дерево поиска в качестве листов, а само состояние помечается как раскрытое, т.е. помещается в список CLOSED (и более повторно не рассматривается[7]).

В оригинальном виде алгоритм `SIPP` строит решение только с использованием имеющихся ребер графа и никак не задействует переходы между произвольными вершинами графа (т.е. решает задачу PFD, но не AA-PFD). Предложим теперь оригинальную модификацию алгоритма `SIPP`, предназначенную для эффективного суботимального для решения задачи AA-PFD.

Идея алгоритма При рассмотрении текущего состояния n и генерации состояний-потомков n', т.ч. $(n,n') \in E$, будем дополнительно рассматривать возможность перехода в n' из состояния-родителя для n – $parent(n)$. Это позволяет из цепочки переходов $parent(n) \to n \to n'$ исключать по возможности промежуточное состояние n, спрямляя (в геометрическом смысле) таким образом путь – см. Рис. 2.15.

Заметим, что возможность перехода из $parent(n)$ в n' может быть установлена с помощью функции los. При этом момент достижения n' устанавливается, как и ранее, с помощью функции `GetMinDelay`, псевдокод которой был представлен ранее на Рис. 2.6.

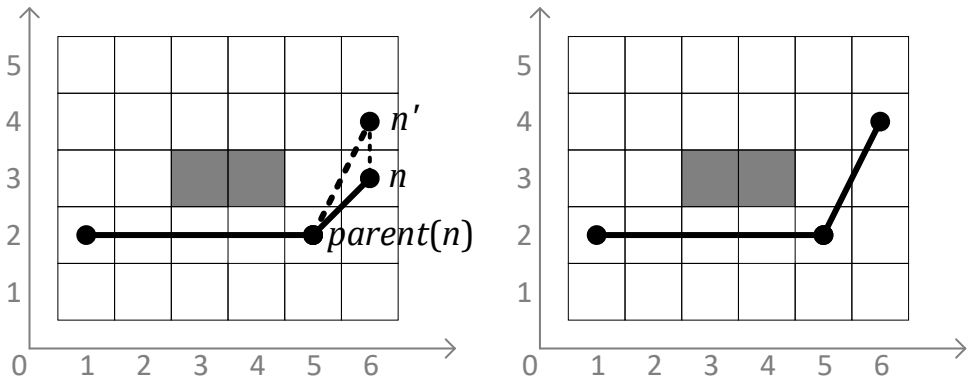


Рисунок 2.15 — Переназначение родителя в алгоритме `AA-SIPP`.

При указанной реализации процедуры создания состояний-потомков становится возможным переиспользовать как общий принцип работы алгоритма `SIPP` (который достаточно прост в понимании и практической реализации), так

[7]Отсутствие необходимости повторного рассмотрения гарантирует использование монотонной эвристической функции – см. Главу 1.

и отдельных его функций. При этом, поддерживается возможность перехода между произвольными вершинами графа, что обеспечивает возможность решения задачи AA-PFD.

Будем именовать алгоритм, опирающийся на описанную идею – `AA-SIPP` (от англ. any-angle safe interval path planning). Отсутствие приставки `TO` (time-optimal) означает, что алгоритм не гарантирует отыскание оптимальных решений задачи AA-PFD (при этом качество отыскиваемых алгоритмом `AA-SIPP` решений можно оценить, и такая оценка будет далее установлена).

Реализация алгоритма Ключевое отличие `AA-SIPP` от `SIPP` состоит в способе генерации состояний-потомков. Представим псевдокод этой процедуры – см. Рис. 2.16.

Основной внешний цикл (Строка 2) идёт по ребрам, исходящим из вершины v, определяющей текущее состояние n (то, для которого необходима генерация состояний-потомков). Для конечной вершины ребра, v', происходит перебор всех безопасных интервалов SI' (Строка 3). В Строке 4 устанавливается стандартным способом (через вызов функции `GetMinDelay`) минимальная задержка, необходимая для достижения v' из v в самый ранний момент времени, принадлежащий интервалу SI'.

В Строке 5 устанавливается родительское состояние (это строка является вспомогательной). В Строке 6 проверяется может ли быть достигнута вершина v' напрямую из вершины родительского состояния – $\tilde{v}$. Для этого вызывается функция los. Если она возвращает $true$, то вычисляется (через вызов функции `GetMinDelay`) минимальная задержка, необходимая для достижения v' из $\tilde{v}$.

Поскольку граф вложен в метрическое пространство (на плоскость), то очевидно, что геометрически путь $\tilde{v} \rightarrow v'$ не длиннее, чем путь $\tilde{v} \rightarrow v \rightarrow v'$, поэтому разумно надеяться, что время достижения v' из $\tilde{v}$ окажется ниже. Однако не стоит забывать, что возможна ситуация, когда переход из $\tilde{v}$ требует предварительной задержки, которая по факту может приводить к достижению v' в более поздний момент. Поэтому в Строках 8–17 устанавливается, через какого родителя вершина v' достигается раньше и соответствующим образом устанавливается родительский указатель и g-значение (время достижения) создаваемого состояния n'. Далее состояние n' добавляется в множество состояний-последователей для n (Строка 18).

Процедура `GenerateAASIPPSuccesors`(n, $\mathcal{G}$, los, $\{T_{safe}(v)\}$, $\{T_{safe}(u,v)\}$):

Входные данные: Состояние поиска $n = (v,[t_l, t_u])$, граф $\mathcal{G}$, функция, определяющая возможность перехода los, множества безопасных интервалов для вершин пар вершин $\{T_{safe}(v)\}$, $\{T_{safe}(u,v)\}$

Выходные данные: Множество состояний-последователей $SUCC$ для состояния n

1 $SUCC \leftarrow \varnothing$; $\delta \leftarrow \varnothing$; $\tilde{\delta} \leftarrow \varnothing$; $v \leftarrow n.v$
2 **foreach** $e = (v, v') \in \mathcal{G}$ **do**
3 **foreach** $SI' = [t'_l, t'_u] \in T_{safe}(v')$ **do**
4 $\delta \leftarrow$ `GetMinDelay`$((v,v'), g(n), [t_l, t_u], [t'_l, t'_u], T_{safe}(v,v'))$
5 $\tilde{n} = (\tilde{v}, [\tilde{t_l}, \tilde{t_u}]) \leftarrow parent(n)$
6 **if** $los(\tilde{v},v') = true$ **then**
7 $\tilde{\delta} \leftarrow$ `GetMinDelay`$((\tilde{v},v'), g(\tilde{n}), [\tilde{t_l}, \tilde{t_u}], [t'_l, t'_u], T_{safe}(\tilde{v},v'))$
8 **if** $\delta = \tilde{\delta} = \varnothing$ **then**
9 **continue**
10 **else**
11 $n' \leftarrow$ `GenerateSearchNode`(v', SI')
12 **if** $\delta < \tilde{\delta}$ **then**
13 $g(n') \leftarrow g(n) + \delta + w(v,v')$
14 $parent(n') \leftarrow n$
15 **else**
16 $g(n') \leftarrow g(\tilde{n}) + \tilde{\delta} + w(\tilde{v},v')$
17 $parent(n') \leftarrow \tilde{n}$
18 $SUCC \leftarrow SUCC \cup \{n'\}$
19 **return** $SUCC$

Рисунок 2.16 — Процедура генерации состояний-потомков с использованием техники переназначения родительского состояния.

С использование описанной процедуры генерации состояний-потомков, псевдокод алгоритма `AA-SIPP` представляется следующим образом – см. Рис. 2.17.

Алгоритм `AA-SIPP`($\mathcal{G}$, v_{start}, v_{goal}, $\{T_{safe}(v)\}$, $\{T_{safe}(u,v)\}$, los, h)**:**

Входные данные: Граф $\mathcal{G}$, начальная и целевая вершины v_{start}, v_{goal}, множества безопасных временных моментов (интервалов) для вершин и пар вершин графа $\{T_{safe}(v)\}$, $\{T_{safe}(u,v)\}$, функция, определяющая возможность переходов los, монотонная эвристическая функция h

Выходные данные: Путь π

1 $n_{start} \leftarrow$ `GenerateSearchNode`$(v_{start}, SI_{first}(v_{start}))$

2 $g(n_{start}) \leftarrow 0$; $parent(n_{start}) \leftarrow$ `null`

3 $OPEN \leftarrow \{n_{start}\}$; $CLOSED \leftarrow \varnothing$

4 **while** $OPEN \neq \varnothing$ **do**

5 $n \leftarrow arg\min_{n \in OPEN} f(n) \{= g(n) + h(n)\}$

6 $OPEN \leftarrow OPEN \setminus \{n\}$

7 $CLOSED \leftarrow CLOSED \cup \{n\}$

8 **if** $n.v = v_{goal}$ **and** $g(n) \in SI_{last}(v_{goal})$ **then**

9 **return** `ReconstructPath`(n)

10 $SUCC \leftarrow$ `GenerateAASIPPSuccesors`$(n, CLOSED, \mathcal{G}, los, \{T_{safe}(v)\}, \{T_{safe}(u,v)\})$

11 `UpdateOpenClosed`$(SUCC)$

12 **return** *failure*

Рисунок 2.17 — Алгоритм эвристического поиска `AA-SIPP` для решения задачи AA-PFD.

Теоретические гарантии алгоритма `AA-SIPP` Введем следующие обозначения. Пусть $\mathbf{P} = \mathbf{P}^+ \cup \mathbf{P}^-$ – множество корректно поставленных задач PFD, где $\mathbf{P}^+$ – подмножество задач, имеющих решение, $\mathbf{P}^-$ – подмножество задач, не имеющих решение. Для отдельной задачи $p \in \mathbf{P}^+$ обозначим множество решений как $\mathbf{\Pi}(p)$, подмножество оптимальных решений обозначим как $\mathbf{\Pi}_{opt}(p)$. Решение задачи p, отыскиваемое алгоритмом `Alg`, обозначим как $\pi_{p,\texttt{Alg}}$.

Докажем теперь следующую теорему об основных теоретических гарантиях алгоритма `AA-SIPP`.

Теорема 4. *Для любой решаемой задачи* PFD *алгоритм* `AA-SIPP` *найдёт решение, при этом стоимость решения не будет превосходить стоимость решения, найденного алгоритмом* `SIPP`. *Формально:* $\forall p \in \mathbf{P}^+ : cost(\pi_{p,\texttt{AA-SIPP}}) \leqslant cost(\pi_{p,\texttt{SIPP}})$.

Доказательство. Рассмотрим модификации алгоритмов `SIPP` и `AA-SIPP`, в которых отсутствует проверка на достижение целевого состояния (не проверяется условие в Строке 8 для обоих алгоритмов), то есть процесс рассмотрения и генерации состояний поиска происходит до тех пор, пока множество кандидатов на рассмотрение OPEN не исчерпает себя. Обозначим эти модификации как `SIPP`′ и `AA-SIPP`′. Докажем, что любое состояние поиска, созданное алгоритмом `SIPP`′ будет создано и алгоритмом `AA-SIPP`′.

Пусть n_k – произвольное состояние, сгенерированное алгоритмом `SIPP`′ в процессе работы. Обозначим цепочку состояний-родителей n_k следующим образом: $\{n_{start}, n_1, n_2, ..., n_{k-1}\}$. Убедимся, что каждое из этих состояний, равно как и само состояние n_k, будет создано алгоритмом `AA-SIPP`′.

Очевидно, что начальное состояние n_{start} будет создано алгоритмом `AA-SIPP`′ (Строка 1 идентична у `SIPP`, `AA-SIPP`, `SIPP`′, `AA-SIPP`′). При рассмотрении этого состояния на первой итерации алгоритма `AA-SIPP`′ будет создано состояние-потомок n_1, с таким же временем достижения $g(n_1)$ и таким же родителем (n_{start}), как и в алгоритме `SIPP`′ (т.к. процедура сброса родителя – Строки 5–7 процедуры `GenerateAASIPPSuccessors`, – не приведёт к этому сбросу из-за того, что родитель для n_{start} отсутствует). Состояние n_1 будет добавлено алгоритмом `AA-SIPP`′ в список OPEN. Рано или поздно это состояние будет извлечено из списка OPEN для рассмотрения (т.к. алгоритм `AA-SIPP`′ продолжает свою работу до тех пор пока список OPEN не пуст). При генерации состояний-потомков для n_1 будет рассмотрен переход из n_1 в n_2, и вместе с этим переход из $parent(n_1)$ в n_2 (Строки 4–7 процедуры `GenerateAASIPPSuccessors`). При этом переход $n_1 \rightarrow n_2$, очевидно, возможен (т.к. иначе состояние n_2 не было бы добавлено в качестве потомка состояния n_1 в алгоритме `SIPP`′). Следовательно, условие в Строке 8 процедуры `GenerateAASIPPSuccessors` не будет истинным и выполнятся строки 11–18. При выполнении этих строк будет создано состояние-последователь n_2, g-значение которого меньше либо-равно g-значению состояния n_2 в алгоритме SIPP′ (т.к. состояние в алгоритме `AA-SIPP`′ создается с наименьшим из двух возможных g-значений – доставляемого с помощью пе-

рехода из рассматриваемого состояния и доставляемого с помощью перехода из состояния-родителя для рассматриваемого состояния). Для этого состояния, n_2, справедливы рассуждения, аналогичные приведенным выше: рано или поздно это состояние будет извлечено из OPEN для рассмотрения, переход $n_2 \rightarrow n_3$ будет рассмотрен, состояние n_3 будет создано и добавлено в дерево поиска алгоритма AA-SIPP$'$ с g-значением меньшим либо равным g-значению этого состояния в дереве поиска алгоритма AA-SIPP$'$. Аналогично рассуждая о состояниях $n_3, n_4, ..., n_{k-1}$, приходим к тому, что состояние n_k будет создано алгоритмом AA-SIPP$'$, при этом $g(n_k)_{\texttt{AA-SIPP}'} \leqslant g(n_k)_{\texttt{SIPP}'}$. Здесь запись $g(n)_{\texttt{Alg}}$ означает g-значение состояния n установленное алгоритмом Alg.

Итак, любое состояние поиска, созданное алгоритмом SIPP$'$ будет создано алгоритмом AA-SIPP$'$, при этом время достижения этого состояния в AA-SIPP$'$ (равное g-значению состояния) будет меньше, чем в SIPP$'$. Таким образом, если целевое состояние достижимо, а это так по условию теоремы (т.к. задача – решаема), то оно будет добавлено алгоритмом AA-SIPP$'$ в дерево поиска. Следовательно, при возвращении на место проверки на достижение целевого состояния, т.е. при выполнении алгоритма AA-SIPP, эта проверка в определенный момент выполнится, т.к. целевое состояние будет извлечено из OPEN на очередной итерации. Таким образом, во-первых, алгоритм AA-SIPP завершит свою работу, во-вторых, в момент завершения работы g-значение целевого состояния, т.е. стоимость решения, будет меньше-либо равно g-значению этого состояния в алгоритме AA-SIPP. ■

Безопасно-интервальное планирование с использованием решетки переходов

Один из наиболее простых способов адаптации алгоритма SIPP к решению задачи AA-PFD, описанный в начале раздела 2.2.3, заключается в модификации процедуры генерации состояний-потомков, таким образом, чтобы при рассмотрении состояния n происходила итерация не только лишь по смежным вершинам, но по всем вершинам графа v', т.ч. $los(n.v, v') = true$ (см. Рис. 2.10). Как было отмечено ранее, этот подход обладает низкой эффективностью из-за

того, что число таких вершин v' для произвольного состояния n очень велико. Одним из возможных способов решения этой проблемы, наряду с теми, что были предложены в работе ранее, может быть принудительное сокращение числа рассматриваемых переходов, т.е. рассмотрение всех переходов вида $n.v \rightarrow v'$, т.ч. $(n.v, v') \in E$ (т.е. в исходном ГРД есть соответствующее ребро) и *части* переходов $n.v \rightarrow v'$, т.ч. $(n.v, v') \notin E$, но при этом $los(n.v, v') = true$. Назовем множество таких переходов *решеткой переходов*.

Одним из известных способов формирования решетки переходов для ГРД является, способ, проиллюстрированный на Рис. 2.18.

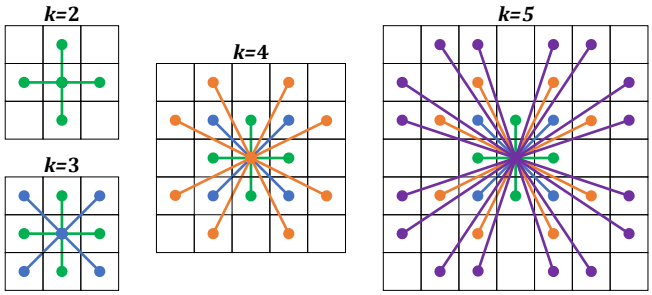


Рисунок 2.18 — Примеры различных решеток переходов для ГРД.

Такие решетки переходов в литературе, посвященной поиску пути на ГРД (см., например, [6; 218]), носят название 2^k-решеток, где $k \in \mathbb{N}$ – это параметр, определяющий вид и число переходов. При $k = 2{,}3$ мы имеем стандартный 4- и 8-связный ГРД соответственно. Нас интересует случай $k > 3$, т.к. именно такие решетки задают переходы, выходящие за изначальную топологию ГРД, с помощью которых может быть получено решение интересующей нас задачи AA-PFD.

Очевидно, при этом, что с ростом k экспоненциально растёт число переходов, которые необходимо рассмотреть для генерации состояний-потомков. В общем случае при увеличении параметра k к предыдущей решетке переходов добавляется 2^{k-1} новых переходов (добавляемые переходы на Рис. 2.18 выделены цветом). Поэтому, с одной стороны, на практике разумно ограничивать k некоторым невысоким значением, например 4 или 5 (в этих случаях для каждого состояния необходимо рассмотреть 16 и 32 перехода, соответственно). С другой стороны, целесообразно разработать модификации алгоритма `SIPP`, такие что общее число состояний, рассмотрение которых необходимо для отыскания решения, сокращалось. Именно такие модификации и будут описаны далее.

Алгоритм Weighted SIPP Одним из распространенных способов сокращения числа рассматриваемых состояний в алгоритмах эвристического поиска (к которым относится и алгоритм SIPP), является изменение способа расчета f-значений состояний поиска n следующим образом:

$$f(n) = g(n) + w \cdot h(n), \tag{2.14}$$

где $w > 1$ – это настраиваемый параметр.

Такой прием обычно называют *взвешиванием* эвристики. Формально можно дать следующее определение.

Определение 29. Пусть $h(n)$ – некоторая эвристическая функция (эвристика), используемая алгоритмом эвристического поиска. Тогда взвешенная эвристика – это функция вида:

$$\hat{h}(n) = w \cdot h(n), \tag{2.15}$$

где $w > 1$ – задаваемый пользователем параметр, именуемый весом эвристики.

Алгоритм поиска, использующий взвешенную эвристику и, следовательно, использующий для расчета f-значений состояний формулу 2.14, называют взвешенным. Например алгоритм A*, использующий взвешенную эвристику называют Weighted A* (weighted переводится с англ. как “взвешенный”) или, сокращенно, WA*.

Использование взвешенной эвристики в общем случае приводит к изменению порядка рассмотрения состояний при поиске. Интуитивно, поиск становится более жадным, т.е. б*о*льшее предпочтение отдается состояниям, которые расположены ближе к целевому состоянию (с точки зрения эвристики). Благодаря этому на практике во многих задачах, например, в задачах поиска пути на статических ГРД, взвешивание эвристики ведет к сокращению числа итераций алгоритма и, соответственно, к снижению времени работы. Однако, изменение порядка рассмотрения вершин влияет и на стоимость отыскиваемого решения.

В этой связи полезно ввести в рассмотрение следующее определение.

Определение 30. Пусть $w > 1$ – некоторая наперёд заданная константа, p – корректно-определенная, решаемая задача PFD (AA-PFD), π_p^* – оптимальное решение этой задачи, а $\pi_{p,\mathtt{Alg}}$ – решение этой задачи, найденное алгоритмом

Alg. Тогда решение $\pi_{p,\texttt{Alg}}$ называется ограниченно-субоптимальным, если выполняется следующее неравенство:

$$cost(\pi_{p,\texttt{Alg}}) \leqslant w \cdot cost(\pi_p^*) \tag{2.16}$$

Известно, что для задачи поиска пути на статическом ГРД алгоритм WA* является ограниченно субоптимальным [14], то есть:

$$cost(\pi_{p,\texttt{WA*}}) \leqslant w \cdot cost(\pi_p^*). \tag{2.17}$$

Более того обычно на поиск решения алгоритму WA* требуется гораздо меньше итераций по сравнению с A*, следовательно, алгоритм работает быстрее и расходует меньше памяти. Таким образом, возникает естественное предположение о том, что для повышения эффективности решения задачи AA-PFD возможно использовать алгоритм, который, во-первых, использует 2^k-решетку примитивов с разумным числом k (например, 4 или 5, как было обосновано выше), во-вторых, использует взвешенную эвристическую функцию (т.е. является взвешенным). Будем обозначать такой алгоритм как Weighted SIPP или просто как WSIPP.

Важно отметить, что изменение порядка рассмотрения состояний при взвешенном поиске приводит, в общем случае, не только к изменению стоимости итогового решения, но и к изменению стоимости путей до промежуточных состояний. Известно, например, что даже при использовании монотонной эвристической функции h, алгоритм WA* не гарантирует, что стоимость пути до извлекаемого из списка OPEN состояния не может быть уменьшена такая гарантия есть в случае использования невзвешенной монотонной эвристики [13]). Таким образом, возникает необходимость корректной реализации механизма повторного рассмотрения состояния (т.н. перераскрытия), что усложняет алгоритм. С другой стороны известно, что при поиске ограниченно-суботимального решения на статическом ГРД с помощью алгоритма WA*, можно обойтись и без реализации процедур повторного рассмотрения вершин, если h – монотонна. Возникает резонный вопрос – можно ли обойтись без реализации механизма перераскрытия в алгоритме WSIPP (т.е. когда решается задача поиска пути на ГРД, возможность прохождения по ребрам/вершинам которого не гарантируется в любой момент времени)? Для ответа на этот вопрос сформулирвем следующее утверждение.

Утверждение 2. *Алгоритм* `WSIPP`*, использующий для своей работы взвешенную монотонную эвристику, не гарантирует отыскания решения задачи поиска пути на динамическом графе (т.е. на графе, возможность прохождения по ребрам/вершинам которого зависит от времени) в случае, если процедура повторного рассмотрения состояний поиска не предусмотрена.*

Доказательство. Докажем утверждение конструктивно, т.е. предъявив пример задачи поиска, которая имеет решение, которое не будет найдено алгоритмом `WSIPP`. Для этого рассмотрим граф, изображенный на Рис. 2.19.

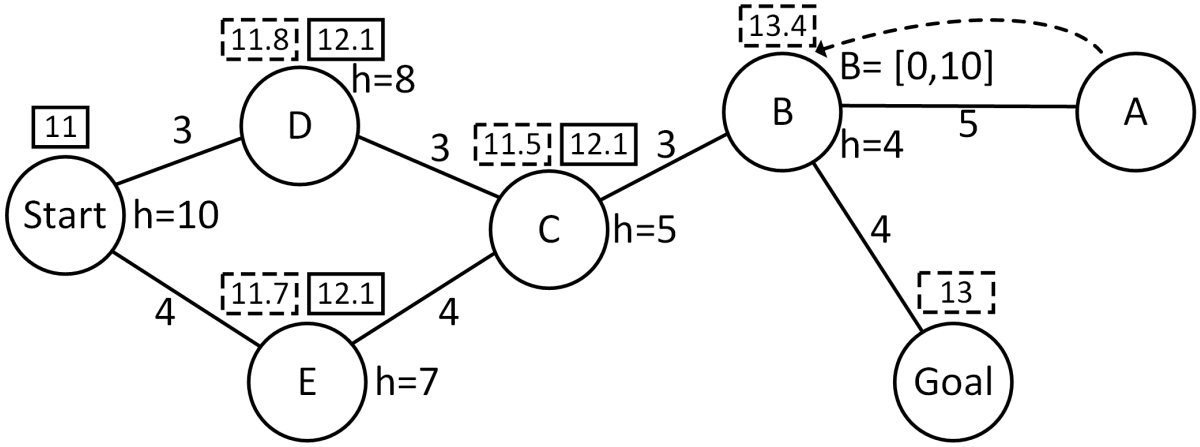


Рисунок 2.19 — Пример задачи поиска пути на динамическом ГРД, которая не может быть решена алгоритмом `WSIPP`.

На Рис. 2.19 вершины графа обозначены буквами. Безопасные интервалы для всех вершин, кроме вершины B, равны $[0, +\infty)$ и для облегчения восприятия не обозначены на рисунке. Для вершины B существует единственный безопасный интервал, равный $[0, 10]$[8]. Безопасные вершины всех ребер равны $[0, +\infty)$, веса ребер – обозначены на рисунке. Например, вес ребра, соединяющего вершины A и B равен 5. Значения эвристической функции для каждой из вершин графа также приведены на рисунке. Например, $h(C) = 5$.

Допустим теперь, что нужно найти путь из вершины *Start* в вершину *Goal*, и для поиска решения мы используем алгоритм `WSIPP` с весом 2 ($w = 2$) при этом повторное рассмотрение состояний поиска не предусмотрено, т.е. если при рассмотрении очередной вершины создается состояние-потомок n, которое

[8]Можно считать, что небезопасный интервал $(10, \infty)$ вершины B возникает из-за того, что эту вершину занимает динамическое препятствие, которое перемещается из вершины A. Это перемещение показано пунктирной стрелкой на рисунке.

находится в CLOSED (т.е. уже было рассмотрено ранее на одной из предыдущих итераций алгоритма), то это состояние-потомок отбрасывается.

Рассмотрим работу алгоритма `WSIPP` по шагам.

На этапе инициализации будет создано состояние $n_{start} = (Start, [0, +\infty))$ (корень дерева поиска) и одно будет помещено в список OPEN (список CLOSED при этом пуст).

На 1-й итерации из OPEN будет извлечено состояние n_{start} и помещено в список CLOSED. Далее будут рассмотрены переходы из вершины, определяющей текущее рассматриваемое состояние, т.е. из $Start$, в смежные вершины: D и E. Эти переходы возможны (т.к. безопасные интервалы вершин D, E и ребер $(Start, D)$, $(Start, E)$ равны $[0, +\infty)$. Соответственно, будут созданы состояния поиска $n_D = (D, [0, +\infty])$, $n_E = (E, [0, +\infty])$. Их g-значения будут равны 3 и 4 соответственно. Эти состояния будут помещены в список OPEN.

Перед началом второй итерации алгоритма дерево поиска будет выглядеть следующим образом: OPEN=$\{n_D, n_E\}$, CLOSED=$\{n_{start}\}$.

На 2-й итерации алгоритма из OPEN будет извлечено состояние с минимальным f-значением. Это n_E, т.к. $f(n_E) = g(n_E)+w\cdot h(n_E) = 4+2\cdot 7 < 3+2\cdot 8 = g(n_D) + w \cdot h(n_D) = f(n_D)$. Это состояние будет добавлено в CLOSED. Переход $E \to Start$ рассматриваться не будет, т.к. n_{start} находится в CLOSED. Переход $E \to C$ будет рассмотрен, в результате чего будет создано и добавлено в OPEN состояние $n_C = (C,[0,+\infty])$ с g-значением: $g(C) = g(E)+cost(C,E) = 4+4 = 8$.

Перед началом третьей итерации алгоритма дерево поиска будет выглядеть следующим образом: OPEN=$\{n_D, n_C\}$, CLOSED=$\{n_{start}, n_E\}$.

На 3-й итерации алгоритма из OPEN будет извлечено состояние с минимальным f-значением. Это n_C, т.к. $f(n_C) = g(n_C) + w \cdot h(n_C) = 8 + 2 \cdot 5 < 3 + 2 \cdot 8 = g(n_D) + w \cdot h(n_D) = f(n_D)$. Это состояние будет добавлено в CLOSED. Переход $C \to D$ рассматриваться не будет, т.к. n_D находится в CLOSED. Переход $C \to B$ будет рассмотрен, однако состояние-потомок n_B создано не будет, т.к. минимально-возможное время его достижения из C, равное $g(C) + cost(C,B) = 8 + 3 = 11$, не принадлежит безопасному интервалу B, равному $[0, 10]$.

Перед началом четвертой итерации алгоритма дерево поиска будет выглядеть следующим образом: OPEN=$\{n_D\}$, CLOSED=$\{n_{start}, n_E, n_C\}$.

На 4-й итерации алгоритма из OPEN будет извлечено состояние n_D, как единственное, находящееся там. Это состояние будет добавлено в CLOSED. Пе-

реход $D \rightarrow Start$ рассматриваться не будет, т.к. n_{start} находится в CLOSED. Аналогично не будет рассматриваться и переход $D \rightarrow C$, т.к. n_C находится в CLOSED. Итерация завершится.

Перед началом пятой итерации алгоритма дерево поиска будет выглядеть следующим образом: OPEN={}, CLOSED={n_{start}, n_E, n_C ,n_D}. То есть список OPEN опустеет. Поэтому алгоритм завершит свою работу на 5-й итерации, вернув специальный символ *failure*, означающий, что путь найти не удалось.

Тем не менее путь из $Start$ в $Goal$, согласованный со всеми безопасными интервалами, существует:

$$\pi = (Start \rightarrow D, 0), (D \rightarrow C, 3), (C \rightarrow B, 6), (B \rightarrow Goal, 9). \tag{2.18}$$

Его можно записать и в виде пар *(вершина, момент посещения этой вершины)*:

$$\pi = (Start,0), (D,3), (C,6), (B,9), (Goal,13). \tag{2.19}$$

Очевидно, вершина B (единственная, безопасный интервал которой ограничен) посещается в момент времени, принадлежащий безопасному интервалу.

Этот путь был бы найден, если бы в алгоритме `WSIPP` поддерживалось повторное рассмотрение вершин. Тогда на 4-й итерации алгоритма переход $D \rightarrow C$ был бы рассмотрен, несмотря на то, что состояние n_C уже находилось бы в CLOSED. При рассмотрении этого перехода g-значение состояния n_C было бы уменьшено с 8 до 6 и n_C было бы повторно добавлено с OPEN с новым g-значением. Далее это состояние было извлечено из OPEN, рассмотрен переход $C \rightarrow B$ и создано состояние n_B, т.к. оно теперь достижимо в момент времени, принадлежащий безопасному интервалу: $g(B) = g(C) + cost(C,B) = 6 + 3 = 9$. Далее при рассмотрении состоянии n_B было бы достигнуто целевое состояние и затем, после его рассмотрения, был бы восстановлен искомый путь. ■

Таким образом, для корректной реализации алгоритма `WSIPP`, т.е. модификации `SIPP`, использующей 2^k-решетку переходов и взвешенную монотонной эвристикой, необходимо реализовать механизм повторного рассмотрения состояний (перераскрытий). Обозначим подобный алгоритм как `WrSIPP` (от “W” – weighted, “r” - re-expansions).

Можно предложить и другой способ решения проблемы повторного рассмотрения вершин, а именно – введение в рассмотрение дубликатов состояний

с разными способами вычисления f-значения. Этот способ был предложен авторами оригинального алгоритма SIPP в [152]. Идея заключается в следующем. При рассмотрении начального состояния для каждого согласованного с временными интервалами перехода создаются две копии состояния-потомка: копия 1-го типа и копия 2-го типа. Эти копии считаются различными, более того у состояния 1-го типа f-значение вычисляется по формуле $f(n) = w \cdot (g(n) + h(n))$, а у состояния 2-го типа – по формуле $f(n) = g(n) + w \cdot h(n)$. Далее, каждый раз, когда рассматривается состояние типа 1 для него создаются потомки обоих типов, а когда рассматривается состояние типа 2, то для него создаются только потомки 2-го типа. В [152] было показано, что подобный подход позволяет, во-первых, гарантировать отыскание решения ограниченной суб-оптимальности, во-вторых, позволяет избежать повторного рассмотрения копий состояний, т.е. каждую копию достаточно рассмотреть лишь однажды. Будем ссылаться на такой вариант взвешенного SIPP как на WdSIPP (от "W" – weighted, "d" – dublicates).

Алгоритм FocalSIPP Помимо взвешивания эвристики получение ограниченно-субоптимальных решений за счет сокращения числа итерация алгоритма может быть реализовано следующим образом [12]. Помимо списков OPEN и CLOSED вводится в рассмотрение дополнительный список состояний FOCAL. Этот список содержит часть состояний из списка OPEN, таких что их f-значения не превосходят минимального f-значения (по всем состояний списка OPEN) более чем в w раз, где w – это (как и ранее) заданный пользователем фактор субоптимальности. Формально:

$$\begin{aligned} &\text{FOCAL} = \{n | n \in \text{OPEN}, f(n) \leqslant w \cdot f_{min}\}, \\ &f_{min} = \min_{n \in \text{OPEN}}(f(n)). \end{aligned} \tag{2.20}$$

Неформально, список FOCAL определяет такие состояния, каждое из которых потенциально гарантирует отыскание ограниченно-субоптимального решения.

После построения/обновления списка FOCAL на каждой итерации выбор состояния для дальнейшего рассмотрения осуществляется именно из этого списка по правилу:

$$n_{best} = \text{argmin}_{n \in FOCAL}(h_{FOCAL}), \tag{2.21}$$

где h_{FOCAL} – это дополнительная (вторичная) эвристическая функция, которая в отличие от основной (первичной) эвристической функции не обязана быть монотонной или даже допустимой. Более того эвристика h_{FOCAL} не обязательно должна оценивать стоимость пути до цели, она лишь должна лишь быть корректно определена на всех состояниях поиска. Интуитивно h_{FOCAL} – это любая функция, которая позволяет дискриминировать, т.е. различать, состояния поиска на предмет их "перспективности" для поиска решения. Например, в задачах поиска пути на статических ГРД h_{FOCAL} может быть определена как оценка числа ребер, входящих в путь до целевого состояния из текущего.

Алгоритм эвристического поиска, опирающийся на описанный выше принцип использования дополнительного списка FOCAL и дополнительной функции выбора из этого списка, будем называть (в соответствии с имеющейся литературой по эвристическому поиску) `FocalSearch`. Соответствующую модификацию алгоритма `SIPP` – `FocalSIPP`.

Обсуждение Выше были предложены две оригинальные модификации алгоритма `SIPP`, а именно `WrSIPP` и `FocalSIPP`, направленные на повышение быстродействия за счет сокращения числа итераций основного цикла. Наряду с ними был описан и известный ранее алгоритм `WrSIPP`. Каждый из этих алгоритмов предполагает задание пользователем фактора субоптимальности w. Проведем мысленный эксперимент по анализу числа итераций алгоритмов `WrSIPP`, `WdSIPP` и `FocalSIPP` в зависимости от значения w.

Пусть w мало (т.е. достаточно близко к 1 сверху). В этом случае весьма вероятно, что `WdSIPP` будет совершать большое число итераций, т.к. в этом алгоритме создаются две копии состояния, отличающиеся способом расчета f-значения: $w \cdot (g(n) + h(n))$ и $g(n) + w \cdot h(n)$. При этом при малых, т.е. близких к 1, значениях w оба этих выражения будут иметь близкие значения (особенно для состояний, расположенные близко к корню дерева поиска). Значит каждое их этих состояний (вероятно) будет рассматриваться, т.е. сначала будет рассматриваться одна копия состояния, затем, на одной из последующих итераций, – другая. `WrSIPP` скорее всего будет предпочтителен для малых w, т.к. он не оперирует дубликатами состояний. При больших значениях w, вероятно, высокую эффективность поиска будет показывать алгоритм `FocalSIPP` (при наличии информативной вторичной эвристики), т.к. в этом случае список FOCAL будет состоять на каждой итерации из большого числа состояний, следовательно, на

каждой итерации будет выбираться состояние, рассмотрение которого обеспечивает наиболее быстрый прогресс к цели. В дальнейшем эти гипотезы будет проверены (и подтверждены) эмпирически.

2.3 Экспериментальные исследования

Экспериментальные исследования предложенных в работе алгоритмов решения задачи AA-PFD были организованы следующим образом. В первой (предварительной) серии экспериментов исследовались алгоритмы, использующие 2^k-решетку переходов: `WrSIPP`, `WdSIPP`, `Focal-SIPP`. Эти алгоритмы являются параметризованными и цель эксперимента заключалась в том, чтобы установить, как задаваемые пользователем параметры влияют на качество работы алгоритмов (скорость работы, стоимость отыскиваемых решений) и выбрать алгоритм, обеспечивающий наиболее подходящий баланс между скоростью работы и стоимостью решений для дальнейшего сравнения. Во второй (основной) серии экспериментов исследовались алгоритмы `TO-AA-SIPP` (гарантирует построение оптимальных решений задачи AA-PFD), `AA-SIPP` (гарантирует построение решений, стоимость которых не превосходит стоимость решения задачи поиска пути в исходной топологии графа, т.е. задачи PFD), и один из алгоритмов, использующий 2^k-решетку переходов, выбранный по итогам первой серии экспериментов. Целью этой серии экспериментов было, во-первых, установить насколько предложенный в работе алгоритм `TO-AA-SIPP` эффективней стандартной (наивной) адаптации принципа безопасно-интервального планирования к решению задачи AA-PFD, во-вторых, проанализировать, насколько изменяется (в сторону увеличения) стоимость решения при использовании субоптимальных алгоритмов и насколько при этом снижается время работы.

Общие принципы и методология проведения исследований Поскольку предлагаемые в работе алгоритмы с точки зрения их практического использования наиболее применимы в задачах планирования траектории мобильного агента (например, колесного робота) в среде с динамическими препятствиями (траектории движения которых предсказаны достаточно точно

или известны, например это могут быть траектории других роботов в многоагентной системе), то рассматривался именно этот сценарий.

Все эксперименты проводились на ГРД различной топологии и размера, которые либо были взяты из известной в этой области коллекции MovingAI [219], либо созданы самостоятельно. Отдельный эксперимент подразумевал решение задачи поиска на динамическом ГРД от заданной начальной вершины до заданной целевой вершины. Предполагалось, что планирование производится для агента с радиусом безопасности, равным шагу дискретизации ГРД (неформально – агент полностью помещается в одну клетку ГРД). В таком случае необходимая для решения задачи AA-PFD функция *los*, которая возвращает *true* на паре вершин ГРД, если агент может проследовать вдоль отрезка прямой, может быть реализована с помощью незначительной модификации широко-известного в компьютерной графике алгоритма Ву [220], который в исходной версии решает задачу подсветки пикселей на растровом дисплее, образующих отрезок некоторой минимальной (не нулевой) толщины[9]. Модификация заключается в том, что помимо клеток ГРД, выбираемых алгоритмом Ву, идентифицируется ряд дополнительных клеток, которые пересекает (хотя ты в одной точке) агент с заданным радиусом безопасности (равным шагу дискретизации), движущийся вдоль отрезка – см. Рис. 2.20.

Для каждого вовлеченного в эксперименты ГРД генерировался ряд заданий. Каждое задание отличается начальной и целевой вершинами (в некоторых экспериментах – совпадают) и, что более важно, распределением безопасных и небезопасных интервалов вершин и ребер. Эти интервалы индуцировались траекториями движения динамических препятствий. То есть, сначала генерировались траектории динамических препятствий, каждая из которых переставляет собой путь на ГРД. Затем определялись клетки ГРД, накрываемыми препятствиями при движении (с помощью вышеупомянутого модифицированного алгоритма Ву), и информация о сегменте траектории (начальная и конечная клетки ГРД и соответствующие моменты времени) сохранялась так, чтобы по индексу клетки ГРД её можно было извлечь впоследствии. Эта информация использовалась для того, чтобы рассчитывать безопасные интервалы ребер и вершин ГРД во время поиска. То есть предпосчет всех безопасных интер-

[9] Другим широко известным аналогичным алгоритмом является алгоритм Брезенхема [217]. Однако он чуть менее подходит для исследуемого в работе случая, т.к. он предназначен для подсветки пикселей, образующих отрезок, не имеющий толщины.

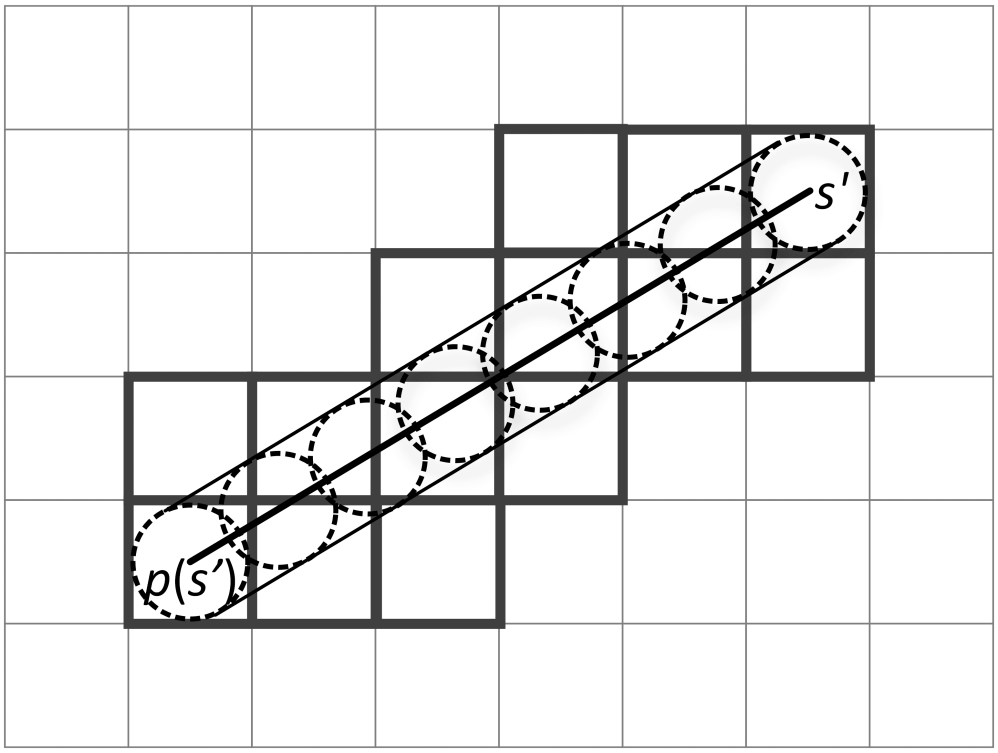


Рисунок 2.20 — Клетки ГРД, с которыми соприкасается агент при движение между двумя вершинами.

валов для всех вершин и пар вершин ГРД не проводился, т.к. это практически не целесообразно. Наоборот, во время поиска, когда генерировалось некоторое состояние поиска, ассоциированное с вершиной ГРД и переходом в неё, определялись безопасные интервалы самой вершины и перехода (пары вершин, не обязательно смежных), ведущего в него.

Основными показателями, отслеживаемыми в каждом эксперименте являлись время, затраченное алгоритмом на поиск пути и его стоимость пути. Эти два индикатора позволяют в полной мере сравнивать алгоритмы между собой по качеству и (вычислительной) эффективности. Заметим, что все проведенные эксперименты являлись сравнительными. Например, в случае времени работы целью было установить, как влияет увеличение числа безопасных интервалов (числа динамических препятствий) на скорость работы алгоритма и как времена работы различных алгоритмов соотносятся между собой. Абсолютные значения времени работы не представляли особого интереса, т.к. абсолютное время работы зависит от используемого аппаратного обеспечения, от того на каком языке программирования и с использованием каких техник и структур данных реализован метод и др. Здесь важно отметить, что все исследуемые алгоритмы были реализованы на языке программирования C++ с использова-

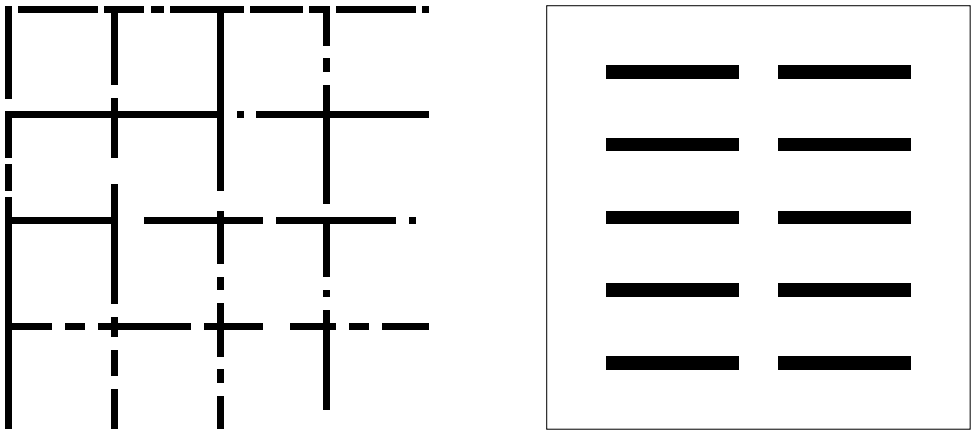
нием одинаковых техник и структур данных, что обеспечивает достоверность сравнения.

Предварительная серия экспериментов В этой серии экспериментов сравнивались между собой алгоритмы `WrSIPP`, `WdSIPP`, `Focal-SIPP` при решении задачи AA-PFD. Эти алгоритмы являются параметризованными, т.е. требуют от пользователя задания двух основных параметров: w – вес эвристической функции и k – параметр, задающий вид решетки переходов. Последний параметр был зафиксирован равным 5, т.е. использовалась 32-связная решетка переходов. Это значение было выбрано т.к. пути, образуемые переходами этой решетки наиболее приближены к решениям задачи AA-PDF (по сравнению с меньшими значениями k), и при этом время работы не столько велико как при использовании $k > 5$. Параметр w варьировался в диапазоне $w = \{[1.01, 1.05, 1.1, 1.25, 1.5, 1.75, 2, 3, 4, 5\}$. Чем больше w, тем ниже, как ожидается, будет время работы алгоритма, но тем выше будет стоимость отыскиваемых решений. Помимо этого в экспериментах использовался (для нормировки результатов) стандартный алгоритм `SIPP`, использующий 32-связную решетку переходов. Этот алгоритм гарантирует отыскание решений, стоимость которых не может быть уменьшена (при использовании фиксированной решетки переходов), но при этом время его работы весьма велико, по сравнению с предложенными в работе субоптимальными модификациями.

В качестве эвристической функции использовалось Евклидово расстояние (монотонная, допустимая эвристика для рассматриваемого случая). В качество вторичной эвристики для алгоритма `Focal-SIPP` использовалось число переходов в оптимальном пути до целевой вершины (образованном с использованием выбранной решетки переходов). Эта эвристика предрассчитывалась перед каждым экспериментом с помощью запуска алгоритма Дейкстры из целевой вершины.

В эксперименте использовались два различных ГРД, которые также будем именовать картами: `rooms` и `warehouse` – см. Рис. 2.21. Первая карта имеет размер 64×64 клетки и моделирует помещение, представляющее набор комнат, соединенных узкими проходами. Вторая карта имеет размер 64×64, однако моделирует складское помещение, в котором имеется определенное количество стеллажей, являющихся препятствиями для агента, путь которого необходимо найти. Выбор этих ГРД мотивирован практическими соображениями, т.к.

они моделируют навигационные робототехнические сценарии внутри помещений. При этом существенное отличие карты `rooms` заключается в том, что из-за геометрической структуры этой карты Евклидова эвристика (игнорирующая расположение препятствий) является менее информативной, т.к. она неверно оценивает расстояние между многими вершинами ГРД.



а) Карта rooms (64 × 64) б) Карта warehouse (64 × 64)

Рисунок 2.21 — Карты, используемые в экспериментальном исследовании алгоритмов `WrSIPP`, `WdSIPP`, `FocalSIPP`.

Для каждой карты было сгенерировано 100 различных заданий, в соответствии с описанной выше методологией эксперимента. Для генерации каждого задания использовалось 250 траекторий динамических препятствий, каждая из которых начиналась и заканчивалась в случайных вершинах графа. Эти траектории индуцируют безопасные интервалы вершин и переходов на ГРД. При этом, чем больше имеется динамических препятствий, тем больше имеется интервалов и, следовательно, тем сложнее становится поиск пути с их учетом. Каждое задание отличалось от аналогичного расположением начальной и целевой вершин (при этом безопасные интервалы вершин и переходов не изменялись). Каждый из исследуемых алгоритмов запускался на каждом задании, затем полученные данные аггрегировались и усреднялись для анализа. Результаты представлены далее на Рис. 2.22 – 2.24. Остановимся на них более подробно.

На Рис. 2.22 представлено среднее относительное время работы исследуемых алгоритмов. Нормировка произведена на время работы алгоритма `SIPP`. Использование относительного времени работы целесообразно, т.к. этот индикатор позволяет сконцентрироваться на прямом сравнении алгоритмов без

использования абсолютных величин, которые зависят от многих факторов (например, от используемого аппаратного обеспечения).

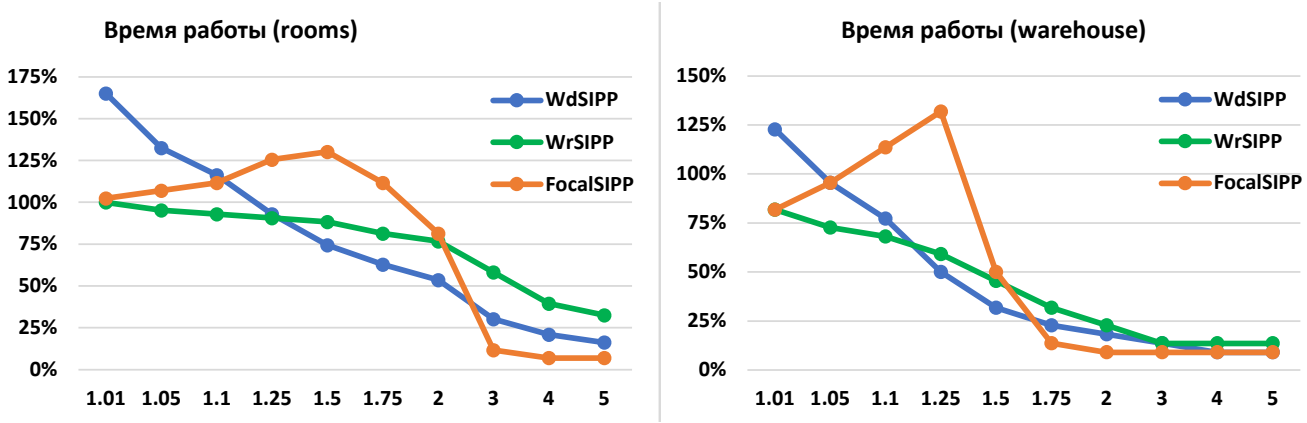


Рисунок 2.22 — Относительное время работы алгоритмов `WdSIPP`, `WrSIPP` и `FocalSIPP`. Ось X – значение параметра w, ось Y – время работы алгоритма нормированное на время работы `SIPP`.

Как видно из рисунка, при малых значения параметра w, т.е. $w \in [1.01, 1.1]$ наиболее эффективным является алгоритм `WrSIPP`; в среднем диапазоне значений, $w \in [1.25, 2]$, – алгоритм `WdSIPP`; при $w > 2$ – алгоритм `FocalSIPP`. Это подтверждает выдвинутые ранее в работе (в Разделе 2.2.4 гипотезы о том, что при малых значениях фактора суб-оптимальности алгоритм `WdSIPP` совершает на практике большое число повторных рассмотрений состояний поиска, т.н. переraскрытий состояний, из-за использования техники дублирования состояний. При этом эта техника показывает себя хорошо на б*о*льших значениях w. Алгоритм `FocalSIPP`, как и предполагалось, показывает наилучший результат при высоких значениях w, когда, благодаря информированной вторичной эвристике, ему удается избегать переraскрытий и максимально точно выбирать ветвь дерева поиска, ведущую к решению.

Более подробную информацию о числе переraскрытий содержит Рис. 2.23.

На Рис. 2.23 слева показана тепловая карта переraскрытий на одном из заданий на карте `rooms`. Чем насыщеннее цвет пикселя, тем более часто алгоритм переraскрывал соответствующую вершину графа. Справа на рисунке изображено усредненное число переraскрытий (ось Y) в зависимости от фактора суботимальности (ось X). Каждая точка соответствует усредненному значению по всей выборке заданий. Как видно из рисунка, выдвинутые ранее гипотезы подтверждаются как с помощью анализа отдельных заданий (слева) так и с помощью анализа всех решений (справа).

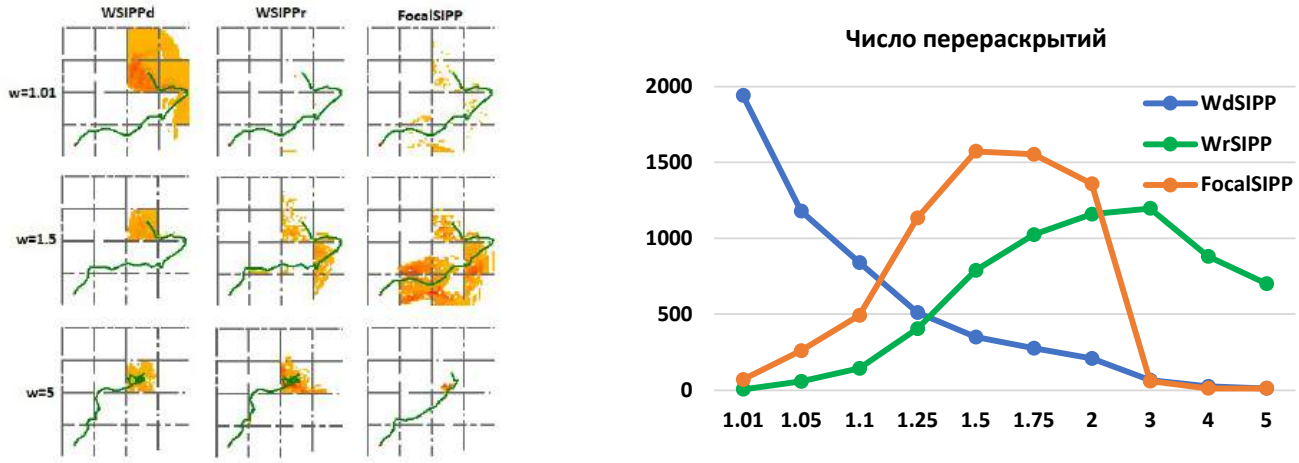


Рисунок 2.23 — Число повторно рассмотренных (перераскрытых) состояний алгоритмами `WdSIPP`, `WrSIPP` и `FocalSIPP`.

Проанализируем теперь стоимость, отыскиваемых алгоритмами решений – см. Рис. 2.24.

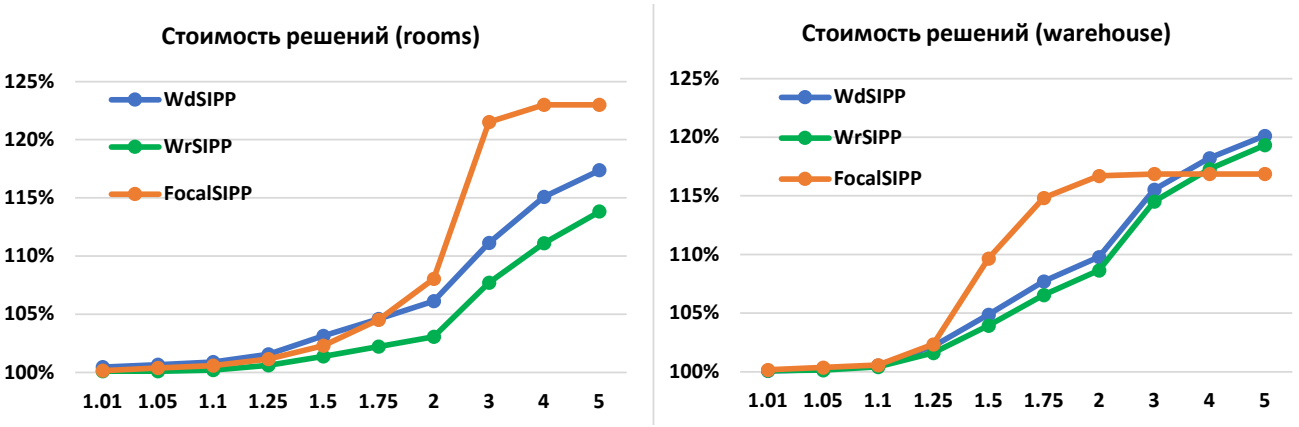


Рисунок 2.24 — Стоимость решений, отыскиваемых алгоритмами `WdSIPP`, `WrSIPP` и `FocalSIPP`.

Как и ранее, по оси X на графиках отложено значение параметра w, а по оси Y – усредненная относительная стоимость решений (нормировка произведена на стоимость решений алгоритма `SIPP`). Очевидно, что с увеличением w растет и стоимость решения. Наибольшая разница на карте `rooms` достигается при использовании алгоритма `FocalSIPP` и составляет 123%, т.е. решения отыскиваемые `FocalSIPP` превосходят оптимальные (при использовании данной решетки примитивов) в среднем на 23%. Наибольшая разница на карте `warehouse` достигается при использовании алгоритмов `WrSIPP` и `WdSIPP` и составляет 119% и 120% соответственно. Если же значение параметра w находится в диапазоне от 1.01 до 2, то стоимость отыскиваемых решений в среднем превосходит стоимость оптимальных решений менее чем на 10%, что является

хорошим результатов для решения практических задач (с учетом демонстрируемого алгоритмами сокращения по времени работы).

Обобщая полученные результаты можно сделать следующий общий вывод о том, что все рассмотренные алгоритмы имеют свои достоинства и недостатки и нельзя однозначно говорить утверждать, что какой-либо из алгоритмов превосходит остальные. При этом, однако, стоит заметить, что высокая эффективность алгоритма `FocalSIPP` при больших значениях w во многом обусловлена наличием информативной вторичной эвристики, построение которой не всегда возможно. Поэтому, с точки зрения практического использования в общем случае целесообразно применять алгоритмы `WrSIPP` и `WdSIPP`, которые не требуют наличия дополнительной эвристической функции и при этом демонстрируют высокую эффективность при решении задач поиска. При этом, если повышенное внимание уделяется качеству (т.е. низкой стоимости) решений, то предпочтительно использование алгоритма `WrSIPP` с низкими значениями параметра w, например $w \in (1; 1.25]$, если же стоимостью решения исходя из практических соображений можно поступиться (допустим получение решений, превышающих оптимальные на 10-15%), то имеет смысл применять алгоритм `WdSIPP` с $w \in [1.5; 3]$.

Основная серия экспериментов Эта серия экспериментов была направлена на исследование предложенных в работе алгоритмов решения задачи AA-PFD: `TO-AA-SIPP` (гарантирует построение оптимальных решений задачи AA-PFD), `AA-SIPP` (гарантирует построение решений, стоимость которых не превосходит стоимость решения задачи поиска пути в исходной топологии графа, т.е. задачи PFD). Для сравнения была реализована и наивная версия безопасно-интервального планирования, предполагающая рассмотрение всех возможных состояний-потомков на каждой итерации поиска. Будем ссылаться на такой алгоритм как на `nTO-AA-SIPP` (от. англ. Naive Time-Optimal Safe Interval Path Planning). Этот алгоритм, очевидно, гарантирует построение оптимальных решений задачи AA-PFD и является единственным (к моменту проведения данного исследования) аналогом алгоритма `TO-AA-SIPP`. В эксперимент был включен и алгоритм поиска, использующий 32-связную решетку переходов, который был выбран на основе предыдущей серии экспериментов, а именно – `WdSIPP` с $w = 2$. Будем такой алгоритм для краткости обозначать просто как `WSIPP`.

Основной целью эксперимента было, во-первых, установить, насколько предложенный в работе алгоритм `TO-AA-SIPP` эффективней (в вычислительном смысле) по сравнению с базовым алгоритмом `nTO-AA-SIPP`; во-вторых, оценить насколько переход от поиска оптимальных решений задачи AA-PFD к поиску суб-оптимальных (с помощью алгоритмов `AA-SIPP` и `WSIPP`) влияет на скорость работы и насколько субоптимальные решения, отыскиваемые алгоритмами `AA-SIPP` и `WSIPP`, проигрывают по стоимости оптимальным.

Для проведения экспериментов использовались 3 различных ГРД (карты) из известной коллекции MovingAI [219]:

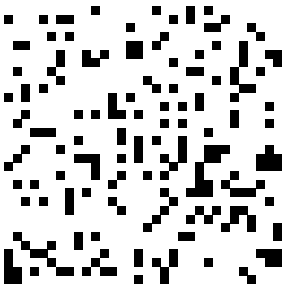
- random-32-32-20 – ГРД размером 32×32 с случайно заблокированными клетками (число заблокированных клеток – 20%);
- arena – ГРД размером 49×49, содержащий небольшое число рассеянных препятствий небольшого размера (карта из видео-игры Dragon Age: Origins);
- warehouse-10-20-10-2-2 – ГРД размером 170×84, моделирущий складское помещение со стеллажами и небольшими проходами между ними в центральной части карты (и свободным пространством сбоку от стеллажей).

Визуализация используемых в эксперименте карт представлена на Рис. 2.25. Здесь и далее для краткости будем на них ссылаться просто как на random, arena, warehouse.

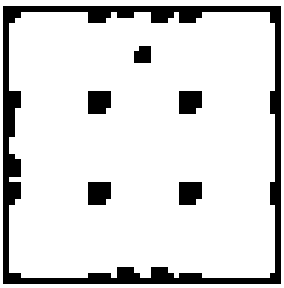
Согласно описанной ранее методологии проведения эксперимента, безопасные интервалы для вершин (и пар вершин) на каждом из ГРД индуцировались динамическими препятствиями. Всего для каждой из карт было сгенерированы траектории движения 32-64-96-128 динамических препятствий. На каждое число движущихся препятствий было сгенерировано 500 различных заданий, отличающихся начальной и конечной вершинами и безопасными интервалами (траекториями движения препятствий).

Отдельный эксперимент состоял в запуске алгоритма на отдельном задании (карта + старт-финиш + набор безопасных интервалов, индуцированных траекториями движения динамических препятствий). Отслеживалась скорость работы алгоритма (время, затраченное на поиск пути) и качество отыскиваемых решений (стоимость пути)[10]. Поскольку все исследуемые алгоритмы реализо-

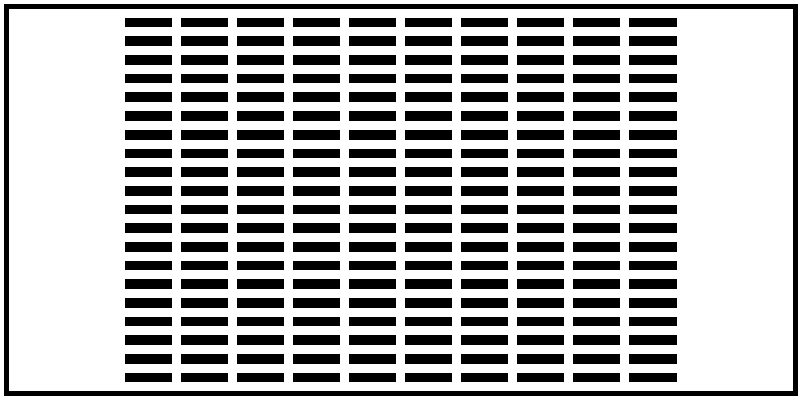
[10]Очевидно стоимость пути может варьироваться только для алгоритмов `AA-SIPP` и `WSIPP`, в то время как `TO-AA-SIPP` и `nTO-AA-SIPP` всегда строят пути минимально-возможной стоимости.



а) Карта random (32 × 32)



б) Карта arena (49 × 49)



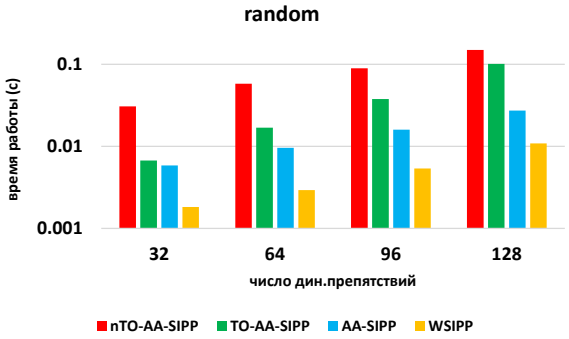
в) Карта warehouse (170 × 84)

Рисунок 2.25 — Карты, используемые в экспериментальном исследовании алгоритмов, прдназначенных для решения задачи AA-PFD.

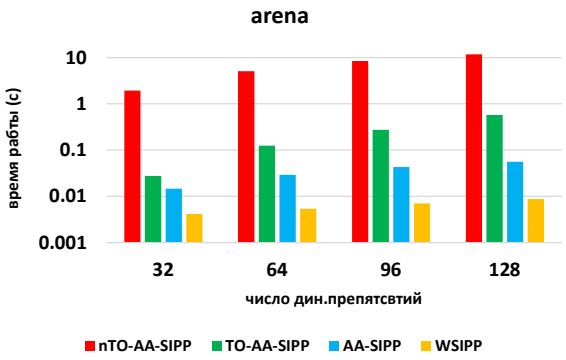
ваны самостоятельно на одном и том же языке программирования (C++) и с использованием одних и тех же техник и структур данных, то получаемые результаты по времени работы являются вполне сравнимыми. Результаты представлены на Рис. 2.26 – 2.28. Опишем их более подробно.

На Рис. 2.26 представлено медианное время работы всех исследуемых алгоритмов в зависимости от карты и числа динамических препятствий. На каждом графике по оси отложено число динамических препятствий, по оси Y – время работы алгоритма. Следует обратить внимание, что масштаб оси Y – логарифический.

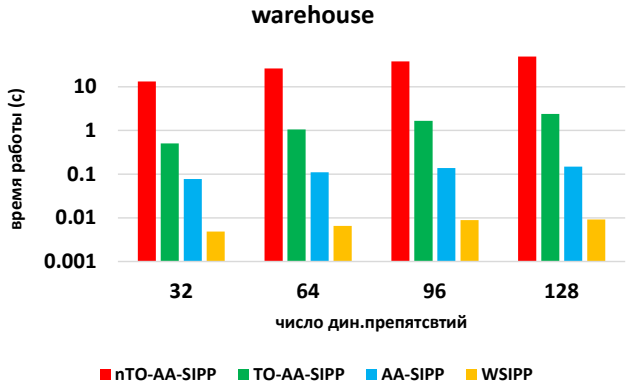
Как видно из рисунка, предлагаемый в работе алгоритм поиска оптимальных решений задачи AA-PDF – `TO-AA-SIPP`, существенно превосходит свой прямой аналог – алгоритм `nTO-AA-SIPP`. Первый быстрее второго почти на порядок в подавляющем большинстве случаев. Так, например, для карты warehouse алгоритм `TO-AA-SIPP` быстрее аналога в среднем в 23.8 раза, а для карты arena – в 40.6 раз. Наименьший прирост производительности наблюдается на карте random, что может быть объяснено высокой плотностью статических препятствий, при которой, вероятно, предлагаемые в работе техники исследования



а) Результаты на карте random (32 × 32)



б) Результаты на карте arena (49 × 49)



в) Результаты на карте warehouse (170 × 84)

Рисунок 2.26 — Медианное время работы алгоритмов `nTO-AA-SIPP`, `TO-AA-SIPP`, `AA-SIPP` и `WSIPP` на трёх различных картах с разным числом динамических препятствий.

пространства поиска, оказываются несколько менее эффективными по сравнению с базовой реализацией. Но даже в этом случае алгоритм `TO-AA-SIPP` быстрее конкурента в среднем в 3 раза.

Для сравнения эффективности алгоритмов `TO-AA-SIPP` и `nTO-AA-SIPP` может быть полезно проанализировать число вызовов одной из важнейших используемых процедур, `ValidateTranstion` (`VT`), каждым из алгоритмов. Эта процедура для двух заданных вершин ГРД вызывает функцию *los*, которая определяет не возникает ли коллизии со статическими препятствиями (заблокированными клетками на ГРД), и затем (если проверка пройдена успешно) определяет минимально-возможный момент достижения целевой вершины с учетом всех безопасных интервалов. Число вызовов этой процедуры для карты warehouse представлено в Табл. 2.1, наряду с числом итераций основного цикла (обозначено в таблице как It, от англ. iterations).

	TO-AA-SIPP		nTO-AA-SIPP	
#	It	VT	It	VT
32	7 348	7 348	856	391 845
64	11 953	11 953	871	461 357
96	16 884	16 884	908	484 819
128	24 088	24 088	931	535 076

Таблица 2.1 — Медианное число итераций (It) и вызовов процедуры ValidateTransition (VT) для алгоритмов TO-AA-SIPP и nTO-AA-SIPP на карте warehouse.

Из таблицы явно следует, что несмотря на б*о*льшее число итераций алгоритма TO-AA-SIPP по сравнению с nTO-AA-SIPP, число вызовов процедуры VT существенно ниже – в 22.2-53.3 раза в зависимости от числа динамических препятствий[11]. Это обстоятельство и обеспечивает заметное повышение быстродействия, наглядно иллюстрируя выгоду от предложенного при разработке алгоритма TO-AA-SIPP принципа “обратного раскрытия”, когда на каждой итерации не генерируются все возможные потомки рассматриваемой вершины, а наоборот ищется (один) подходящий родитель.

Возвращаясь к анализу времени работы алгоритмов (Рис. 2.26), можно заметить, что отказ от требования получения оптимальных решений существенно ускоряет поиск. Так предлагаемые в работе алгоритмы поиска суб-оптимальных решений задачи AA-PFD – AA-SIPP и WSIPP, на 1-2 порядка быстрее, чем TO-AA-SIPP. Более наглядно выигрыш от применения суб-оптимальных алгоритмов показан в Табл. 2.2.

Каждая ячейка таблицы показывает во сколько раз в среднем алгоритмы AA-SIPP и WSIPP быстрее отыскивают субоптимальное решение задачи AA-PFD, нежели алгоритм TO-AA-SIPP отыскиваем оптимальное. Наиболее существеное ускорение наблюдается на карте warehouse. На ней при достаточно большом числе динамических препятствий (> 96) наблюдается ускорение в более, чем 10-100 раз (для алгоритмов AA-SIPP и WSIPP соответственно). На карте random

[11]Стоит заметить, что совпадение числа итераций и вызовов процедуры VT для алгоритма TO-AA-SIPP не случайно, т.к. по дизайну алгоритма на каждой итерации основного цикла процедура VT вызывается один раз – для валидации перехода в текущую вершину из наиболее перспективного родителя

#	random		arena		warehouse	
	AA-SIPP	WSIPP	AA-SIPP	WSIPP	AA-SIPP	WSIPP
32	1.2	3.7	1.9	6.7	6.5	104.0
64	1.8	5.8	4.3	23.0	9.6	160.7
96	2.4	7.0	6.4	39.2	11.9	188.0
128	3.7	9.3	10.4	66.3	16.0	258.5

Таблица 2.2 — Ускорение по времени работы при переходе к поиску суб-оптимальных решений задачи AA-PFD.

выигрыш наименьший, но даже в этом случае ускорение весьма существенно, до 3-9 раз при 128 препятствиях.

Итак, очевидно, что поиск суб-оптимальных решений гораздо менее трудозатратен, чем поиск оптимальных. Проанализируем теперь насколько решения, конструируемые алгоритмами AA-SIPP и WSIPP хуже по качеству (т.е. выше по стоимости), чем оптимальные. Эта информация представлена в виде диаграмм разброса на Рис. 2.27.

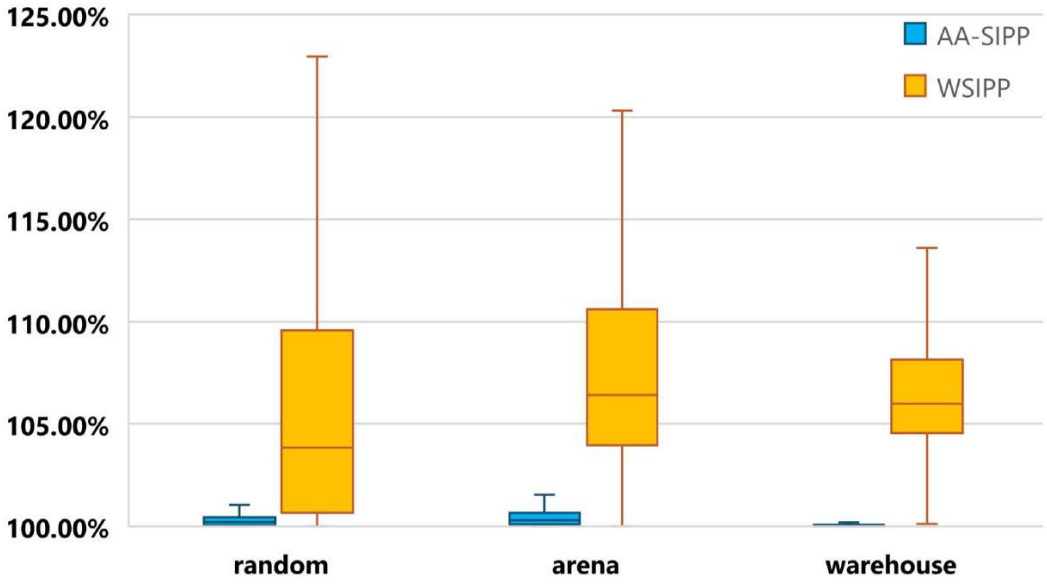


Рисунок 2.27 — Стоимость решений, отыскиваемых алгоритмами AA-SIPP и WSIPP (относительно стоимости оптимальных решений).

На диаграмме представлена стоимость путей, отыскиваемых алгоритмами AA-SIPP и WSIPP в процентах относительного стоимости оптимальных решений (отыскиваемых алгоритмом TO-AA-SIPP). Выбросы не изображены, т.к. в отдельных случаях относительная стоимость решений может превышать 200-300%.

Из диаграммы видно, что в подавляющем большинстве случаев решения, отыскиваемые алгоритмом AA-SIPP превосходят, по стоимости оптимальные, менее чем на 1-2%. Для алгоритма WSIPP превышение более существенно – порядка 2-12%. Следует отметить отметить, что в отдельных случаях превышение может быть заметно выше (точки данных, соответствующие этим случаям, т.н. выбросы, не показаны на диаграммах). Пример, такого пути изображен на Рис. 2.28.

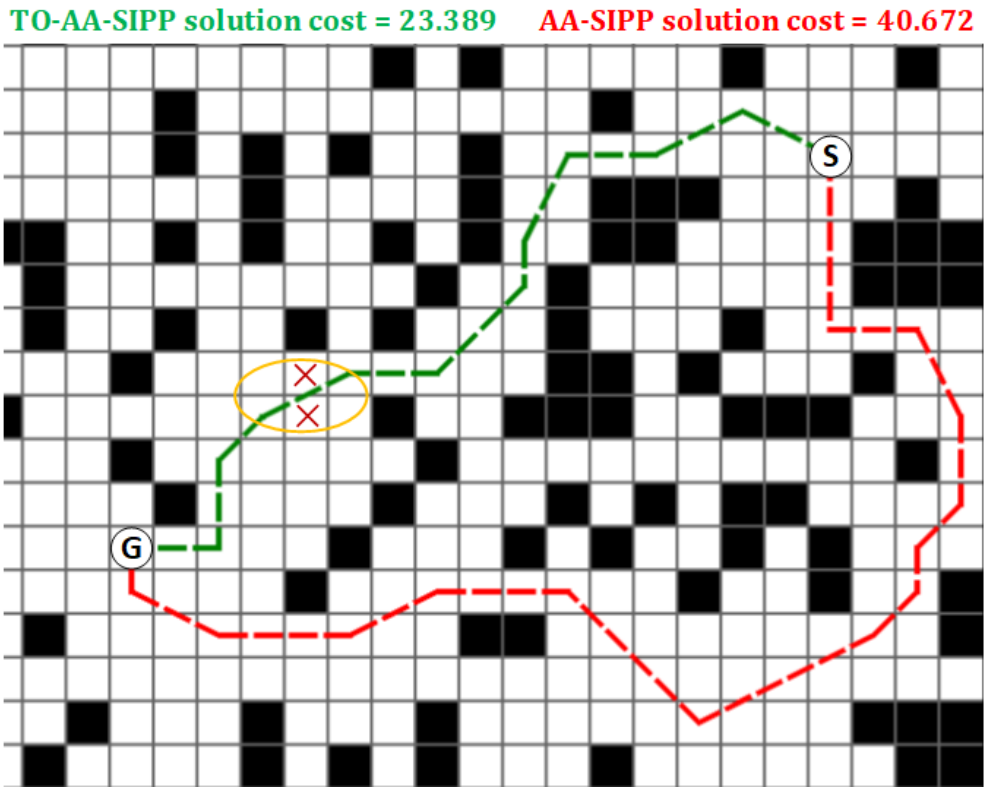


Рисунок 2.28 — пример путей, отыскиваемых алгоритмами AA-SIPP и TO-AA-SIPP, для одного из заданий на карте random (изображен фрагмент карты).

На рисунке изображены два пути для одного и того же задания, полученные с помощью алгоритмов TO-AA-SIPP и AA-SIPP (динамические препятствия и небезопасные интервалы не изображены для наглядности). В этом примере алгоритм AA-SIPP не смог, из-за применяемой техники сброса родительского указателя, обнаружить переход, показанные желтым овалом. В результате был построен путь из другого топологического класса, стоимость которого составила 173% от стоимости оптимального пути.

Для алгоритма WSIPP встречаются случаи 2- и 3-кратного превышения стоимости. Стоит, однако, помнить, что алгоритмы AA-SIPP и WSIPP работают на порядок (несколько порядков) быстрее, чем TO-AA-SIPP, поэтому наблюдаемое

превышение стоимости конструируемых ими решений является вполне допустимым с практической точки зрения, особенно с учетом того, что в большинстве случаев разница в стоимости минимальна.

2.4 Выводы по главе

В данной главе работы рассматривалась задача поиска пути на графе регулярной декомпозиции, проходимость вершин и ребер которого меняется с течением времени (т.е. одна и та же вершина/ребро графа может быть проходима в один момент времени и непроходима в другой, что нужно учитывать при поиске). Для решения этой задачи был предложен ряд новых алгоритмов, опирающихся на подход безопасно-интервального планирования. Предложенные алгоритмы теоретически исследованы и установлены их теоретические гарантии относительно качества предоставляемых решений. В частности впервые в мире предложен алгоритм, гарантирующий отыскание оптимальных решений рассматриваемой задачи, – `TO-AA-SIPP`. Этот алгоритм использует оригинальный механизм поиска, основанный на идее так называемых обратных раскрытий, когда в ходе поиска происходит определение наилучшего состояния-родителя, а не прямая генерация состояний-потомков, как в стандартном (эвристическом) поиске. Предложены алгоритмы поиска суб-оптимальных решений – `WSIPP`, `FocalSIPP` и `AA-SIPP`. Первые два оперируют так называемой решеткой переходов и используют техники взвешивания эвристики и фокусировки поиска для ускорения поиска. Для этой же цели последний алгоритм использует механизм переопределения состояния-родителя в ходе поиска. Доказаны утверждения, характеризующие качество конструируемых решений (длину путей). Все предложенные алгоритмы экспериментально исследованы на широком спектре задач, в том числе на заданиях из известной в сообществе коллекции MovingAI. Результаты экспериментального исследования подтвердили высокую вычислительную эффективность предлагаемых алгоритмов. В частности предложенный алгоритм поиска оптимальный решений `TO-AA-SIPP` на порядок быстрее алгоритма, опирающегося на стандартный подход к генерации состояний-потомков в ходе поиска. Предложенные в работе суб-оптимальные алгоритмы в свою очередь на порядок быстрее `TO-AA-SIPP`, при этом качество

отыскиваемых ими решений лишь незначительно уступает оптимальным алгоритмам (менее чем на 5% процентов для алгоритма AA-SIPP). Предлагаемые методы и алгоритмы могут использоваться в робототехнических приложений, в частности, когда необходимо обеспечивать автономную навигацию мобильного робота в среде с динамическими препятствиями и в много-агентных робототехнических системах, когда динамическими препятствиями являются другие агенты (роботы). Более подробно о последней задаче будет сказано в следующей главе работы.

Глава 3. Поиск совокупности неконфликтных путей на графах регулярной декомпозиции

Одним из наиболее активно развивающихся в настоящее время областей робототехники является автоматизация складской деятельности. В частности, в последнее время активно внедряются централизованные системы управления множеством мобильных роботов, которые участвуют в перемещении товаров по складским площадям. Для эффективной работы подобных систем необходима, в частности, разработка алгоритмов централизованного планирования безопасных траекторий перемещений группы роботов в общем рабочем пространстве. Эта задача в свою очередь может быть формализована как задача поиска совокупности путей на ГРД, удовлетворяющих определенным ограничениям. Именно этой задаче и будет посвящена данная глава работы. В ней будет предложен ряд новых методов решения указанной задачи, основным отличием которых от известных мировых аналогов будет являться поддержка возможности перемещения между произвольными вершинами графа, что положительным образом влияет на сокращение времени следования по траекториям. Предложенные методы будут исследованы теоретически и эмпирически.

3.1 Постановка задачи

Рассмотрим сначала упрощенную постановку задачи, наиболее часто встречающуюся в литературе.

3.1.1 Базовая постановка (задача MAPF)

Рассмотрим 4-связный взвешенный ГРД $\mathcal{G} = (V, E, w)$, вложенный в метрическое пространство $\mathcal{M} = \mathbb{R}^2$ так, как это описано в Главе 1. То есть каждой вершине ГРД соответствует точка на плоскости, а с каждым ребром графа $e = (u, v) \in E$ ассоциирован отрезок AB, т.ч. $coord(u) = A$, $coord(v) = B$, где

$coord : V \to \mathbb{R}^2$ – функция, сопоставляющая вершины графа точкам в $\mathbb{R}^2$ (функция вложения графа в пространство). Вес ребра определен как $w(e) = \|AB\|$.

Как и ранее будем измерять расстояние в условных единицах и, без ограничения общности, будем считать, что горизонтально- и вертикально-смежные вершины ГРД отстоят друг от друга на расстоянии 1, т.е. вес всех ребер в ГРД равен 1 (т.к. ГРД является 4-связным, то ребрами связаны только горизонтально- и вертикально-смежные вершины и никакие иные).

Введем в рассмотрение множество временных моментов $T = [0, 1, 2, 3, ...)$.

Зафиксируем n начальных и целевых вершин ГРД соответственно: $S = \{s^1, s^2, ..., s^n |\ s^i \in V, s^i = s^j \Leftrightarrow i = j\}$, $G = \{g^1, g^2, ..., g^n |\ g^i \in V, g^i = g^j \Leftrightarrow i = j\}$.

Путем π^i из s^i в g^i назовем последовательность пар вершин ГРД:

$$\begin{aligned} &\pi^i = (v_0^i, v_1^i), (v_1^i, v_2^i), ..., (v_{k-1}^i, v_k^i), \\ &v_{\{0,1,...,k\}}^i \in V, \\ &\text{т.ч.:} \\ &v_0^i = s^i \\ &v_k^i = g^i \\ &v_j^i = v_{j+1}^i \oplus (v_j^i, v_{j+1}^i) \in E \end{aligned} \tag{3.1}$$

Здесь знак $\oplus$ означает логическое исключающее ИЛИ, т.е. либо в паре вершины совпадают, либо являются различными и соответствуют одному из ребер ГРД.

Пары вершин образующих путь будем называть действиями: $a = (u, v)$. При этом, если вершины, образующие пару, различны (и, по определению, образуют ребро графа), то будет называть это действие *действием перемещения*, если же вершины совпадают, то будем говорить о *действии ожидания*.

Теперь путь можно записать в виде последовательности действий:

$$\pi^i = (a_1^i, a_2^i, ..., a_k^i) \tag{3.2}$$

С каждым действием свяжем продолжительность этого действия, которая одинакова для всех действий и равна 1 (совпадает в весом ребер ГРД):

$$dur(a) = 1 \tag{3.3}$$

Весом пути (стоимостью пути) будем называть суммарную продолжительность всех действий, образующих путь:

$$\begin{aligned} cost(\pi^i) &= dur(a_1^i) + dur(a_2^i) + ... + dur(a_k^i) \\ &= k \end{aligned} \tag{3.4}$$

Очевидно, что, т.к. продолжительность любого действий одинакова и равна 1, то вес пути равен количеству действий, его образующих.

Рассмотрим теперь два различных пути π^i и π^j.

Определение 31. Будем говорить, что пути π^i и π^j содержат *вершинный конфликт*, если выполняется следующее условие:

$$\exists t \leqslant \min\{cost(\pi^i), cost(\pi^j)\} : \ target(a_t^i) = target(a_t^j), \tag{3.5}$$

где $target(a) = v$ – это вторая вершина в паре (u, v), определяющей действие a.

Определение 32. Будем говорить, что пути π^i и π^j содержат *реберный конфликт*, если выполняется следующее условие:

$$\exists t \leqslant \min\{cost(\pi^i), cost(\pi^j)\} : \ source(a_t^i) = target(a_t^j), target(a_t^i) = source(a_t^j) \tag{3.6}$$

где $source(a) = u$ – это первая вершина в паре (u, v), определяющей действие a.

Неформально, вершинный конфликт возникает тогда, когда различные пути используют одну и ту же вершину в один и тот же такт времени, а реберный – тогда, когда используется одно и то же ребро (в один и тот же такт времени). Визуальный пример показан на Рис. 3.1.

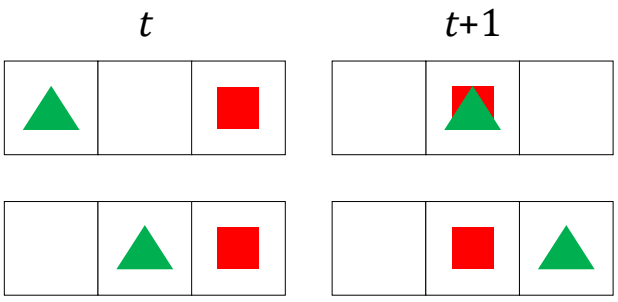


Рисунок 3.1 — Вершинный и реберный конфликт.

Обозначим множество путей из заданных начальных вершин, $s^1, s^2, ..., s^n$, в заданные целевые вершины, $g^1, g^2, ..., g^n$, как $\Pi = \{\pi^1, \pi^2, ..., \pi^n\}$.

Если для этого множества путей верно, что любые два пути π^i и π^j не содержат конфликтов (ни вершинных, ни реберных), то будем называть такое множество неконфликтным.

Совокупный вес (стоимость) этого множества путей определим как:

$$cost(\Pi) = \sum_{i=1}^{n} cost(\pi^i). \tag{3.7}$$

Теперь задачу построения множества неконфликтных путей на ГРД можно формально определить следующим образом.

Определение 33. Задача построения множества неконфликтных путей на ГРД – это набор:

$$\text{MAPF} = (\mathcal{G}, S, G). \tag{3.8}$$

Решение задачи MAPF (от англ. **M**ulti-**a**gent **P**ath**f**inding) – множество путей Π, являющееся неконфликтным (т.е. любые два пути не содержат конфликтов).

Оптимальным решением задачи считается множество неконфликтных путей минимальной стоимости (определенной в соответствии с 3.7).

Пример задачи MAPF изображен на Рис. 3.2 в левом верхнем углу. На этом же рисунке изображено решение этой задачи.

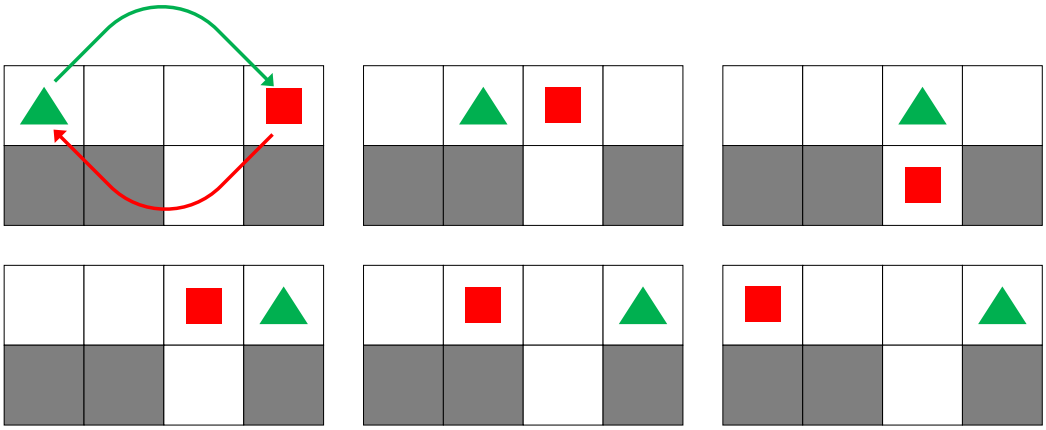


Рисунок 3.2 — Пример задачи MAPF и её решения.

Стоимость указанного решения равняется 8, т.к. первый путь (соответствующий перемещению зеленого треугольника на рисунке) содержит 3 действия, а второй – 5. Это решение является оптимальным. Естественно, существует и множество суботимальных решений этой задачи. Простейший пример

конструирования такого решения – добавление m действий ожидания в начало каждого пути.

Связь с задачей планирования траектории Описанная выше задача имеет очевидную связь с задачей (централизованного) планирования совокупности неконфликтных траекторий для группы (кооперативных) агентов, функционирующих в общем пространстве (на плоскости). Очевидно, что данное выше определение пути соответствует траектории перемещения агента в пространстве, при этом в процессе этого перемещения возможны ожидания в определенных местах (соответствующих вершинам графа). Определенное выше понятие конфликта соответствуют потенциальной ситуации столкновения агентов (или критического уменьшения расстояния между ними), которой нужно избегать. Таким образом, решение формально поставленной выше задачи может быть транслировано в совокупность траекторий для мобильных агентов, следуя по которым (без отклонений) они достигнут поставленных целей без столкновений.

При этом описанная формализация содержит ряд ограничивающих допущений и предположений, основное из которых состоит в том, что агенты могут перемещаться только в ортогональных направлениях, т.к. рассматривается 4-связный ГРД. Это существенным образом может увеличивать длину отыскиваемых путей и, следовательно, время следования по ним. При этом отказ от этого предположения сопряжен с рядом трудностей. Так предположение о том, что любое действие перемещения или ожидания обладает одинаковой продолжительностью и занимает 1 такт времени, обеспечивает возможность достаточно легко (как в теории так и на практике) определить и идентифицировать конфликты между путями. Конфликт в приведенной постановке всегда привязан к конкретному временному шагу и конкретной вершине (или ребру) ГРД. Если же допустить перемещение агентов в произвольном направлении, что может быть гораздо более целесообразным на практике (особенно когда рабочее пространство содержит большие области без статических препятствий), то необходим альтернативный подход к определению понятия конфликта и разработка (эффективных) методов методов поиска, поддерживающих разрешение подобных конфликтов. Именно такие методы и будут представлены и исследованы далее. Предварим описание этих методов расширенной постановкой задачи, допускающей возможность перемещения агентов в произвольном направлении.

3.1.2 Расширенная постановка (задача AA-MAPF)

Будем в явном виде предполагать, что n агентов функционируют в общем рабочем пространстве, а именно перемещаются на плоскости, в которую вложен ГРД $\mathcal{G}$. Начальные и целевые вершины графа, $S = \{s^1, s^2, ..., s^n |\ s^i \in V, s^i = s^j \Leftrightarrow i = j\}$, $G = \{g^1, g^2, ..., g^n |\ g^i \in V, g^i = g^j \Leftrightarrow i = j\}$, соответствуют начальным и целевым положениям агентов.

Определим действие перемещения (агента) как $a = (u, v)$, где $u, v \in V$ $(u \neq v)$, при этом необязательно $(u,v) \in E$. Т.е. теперь возможно перемещение между произвольными (различными) вершинами ГРД, необязательно смежными. Будем считать, что действие перемещения $a = (u,v)$ возможно тогда и только тогда, когда $los(u,v) = true$, где $los : V \times V \to \{true, false\}$ – это заданная функция, определяющая возможность перехода между произвольными вершинами ГРД.

Обозначим множество временных моментов как $T = [0, +\infty)$.

С каждым действием перемещения $a = (u,v)$ будем ассоциировать отрезок AB, т.ч. $coord(u) = A$, $coord(v) = B$, где $coord : V \to \mathbb{R}^2$ – функция, сопоставляющая вершины ГРД точкам в $\mathbb{R}^2$ (функция вложения графа в пространство). Продолжительность действия перемещения определим как $\|AB\|$.

Будем рассматривать и действия ожидания, т.е. действия вида $a = (v, v)$. По определению, с каждым действием ожидания ассоциирована ровно одна вершина ГРД, $v \in V$. При этом в отличие от упрощенной постановки задачи, будем считать что продолжительность действия ожидания может быть произвольной. Следовательно, возникает необходимость включить эту продолжительность в определение действия ожидания.

Действие теперь по определению задается тройкой:

$$\begin{aligned} &a = (u, v, \Delta), \\ &u, v \in V, \\ &\Delta \in T. \end{aligned} \tag{3.9}$$

Здесь v, u – это вершины ГРД, определяющие действие, а Δ – его продолжительность. При этом для любого действия перемещения значение Δ фиксировано и равно $\|AB\|$, где AB это отрезок, соединяющий вершины; а для действия ожидания Δ может принимать любое положительное значение.

Обозначим как A множество возможных действий. Оно содержит в себе два подмножества $A = \{A_{move}, A_{wait}\}$, которые не пересекаются, $A_{move} \cap A_{wait} = \varnothing$, – множество действий перемещения и множество действий ожидания. Первое – конечно, т.к. число пар вершин графа ограничено и продолжительность каждого действия перемещения фиксирована. Мощность этого множества ограничена величиной $|V| \cdot (|V| - 1)$, где $|V|$ – число вершин ГРД. Второе множество бесконечно, т.к. действие ожидания может иметь произвольную продолжительность (и для любой вершины графа можно задать сколь угодно большое число действий ожидания).

Рассмотрим теперь некоторый момент времени $t_{start} \in T$ и допустим, что агент начал выполнение действия $a = (u,v,\Delta)$ в этот момент. Обозначим через $pos(t, a, t_{start})$ положение агента на плоскости, в которую вложен ГРД, в момент времени t. Это положение определяется (в векторной форме) следующим образом:

$$\vec{pos}(t, a, t_{start}) = \begin{cases} \varnothing & t \notin [t_{start}, t_{start} + \Delta] \\ \vec{A} & a \in A_{wait} \\ \vec{A} + \frac{\vec{AB}}{\|\vec{AB}\|} \cdot (t - t_{start}) & a \in A_{move} \end{cases}, \qquad (3.10)$$

где $\vec{A}$ это вектор с координатами $coord(u)$, а $\vec{B}$ это вектор с координатами $coord(v)$.

То есть, при совершении действия ожидания позиция агента в течении всей продолжительности действия совпадает с позицией вершины ГРД, определяющей это действие. При перемещении между двумя различными вершинами графа, позиция агента меняется и соответствует прямолинейному и равномерному (т.е. ускорение и инерциальные эффекты отсутствуют) движению с единичной скоростью от одной вершины к другой. При этом позиция агента не определена в любой момент времени, выходящий за пределы продолжительности действия.

Определим понятие пути следующим образом. Путем π^i из $s^i \in S$ в $g^i \in G$ назовем последовательность пар (*действие, момент начала действия*):

$$\begin{aligned}
&\pi^i = (a_1^i, t_1^i), (a_2^i, t_2^i), ..., (a_k^i, t_k^i) \\
&\text{т.ч.} \\
&source(a_1^i) = s^i \\
&\forall j \in [1,k-1]: \ source(a_{j+1}^i) = target(a_j^i) \\
&target(a_k^i) = g^i \\
&\forall j \in [1,k-1]: \ t_j^i + dur(a_j^i) = t_{j+1}^i
\end{aligned} \tag{3.11}$$

Использованные записи $source$, $target$, dur для действия $a = (u,v,\Delta)$ обозначают, очевидно, следующее: $source(a) = u$, $target(a) = v$, $dur(a) = \Delta$.

Итак, путь это последовательность действий, каждое из которых (начиная со второго) начинается в тот момент и в той вершине, в которых заканчивается предыдущее действие, при этом первое действие начинается в начальной вершине, а последнее – заканчивается в целевой.

Продолжительность пути π^i определим как:

$$dur(\pi^i) = t_k^i + dur(a_k^i) \tag{3.12}$$

То есть, как и ранее, продолжительность равняется времени достижения целевой вершины.

Положение агента, следующего вдоль пути π^i, в момент времени t будем обозначать как $pos(\pi^i,t)$. Технически, для вычисления этого положения необходимо определить, какое действие запланировано к выполнению в этот момент (это можно вычислить, т.к. продолжительность каждого действия в заданном пути известна) и затем для данного действия a_j^i, момента его запланированного начала t_j^i воспользоваться формулой 3.10, т.е. вызвать функцию $pos(t, a_j^i, t_j^i)$.

Нетрудно заметить, что приведенные выше определения действия, продолжительности действия, пути отличаются от аналогичных определений упрощенной постановки задачи. Перечислим основные отличия. Во-первых, в приведенной постановке допускается произвольная продолжительность действий (в отличии от фиксированной продолжительности, одинаковой для всех действий). Во-вторых, допускаются действия перемещения между произвольными вершинами ГРД. Возникает необходимость модификации понятия конфликта.

В рассматриваемой постановке будем определять конфликт через действия и моменты начала их выполнения. Формально, введем в рассмотрение функцию:

$$con: \ A \times T \times A \times T \to \{true, false\} \tag{3.13}$$

Рассмотрим теперь четверку $(a_k^i, t_k^i, a_l^j, t_l^j)$, где a_k^i – это k-е действие i-го агента, a_l^j – это l-е действие j-го агента, а t_k^i и t_l^j – моменты начала совершения этих действий, соответственно.

Будем говорить, что имеет место конфликт, если $con(a_k^i, t_k^i, a_l^j, t_l^j) = true$.

Способ задании функции con может быть различным (исходя из особенностей решаемой задачи). Здесь и далее будем считать, что:

$$\begin{aligned} &con(a_k^i, t_k^i, a_l^j, t_l^j) = true \iff \\ &\exists t \in [0, \min(dur(a_k^i), dur(a_l^j))] : \\ &dist(pos(t, a_k^i, t_k^i), pos(t, a_l^j, t_l^j)) < 2 \cdot r, \end{aligned} \tag{3.14}$$

где $dur(a)$ – продолжительность действия a, $pos(t, a, t_{start})$ – положение агента в момент времени t при совершении действия a, начало которого было в момент t_{start} (вычисляется по 3.10), $dist$ – расстояние, r – положительная константа, задающая радиус безопасности агента.

То есть действия считаются конфликтными, если при их совершении существует такой момент времени, что агенты оказываются на расстоянии меньшем, чем $2r$ друг от друга.

Практическая трактовка понятия конфликта достаточно очевидна. Представим, что каждый агент – есть робот дискообразной формы, и радиус диска равняется r. Тогда конфликтность двух действий означает, что начав их выполнение роботы физически столкнутся между собой.

Пример, иллюстрирующий конфликт двух действий приведен на Рис. 3.3. В данном примере рассматриваются два действия одинаковой продолжительности, которую обозначим как Δ. Начальный момент совершения обоих действий совпадает: $t_k^i = t_l^j = 0$. В момент времени t_{col} расстояние между агентами составляет $2 \cdot r$. При этом, в любой из моментов времени $t \in (t_{col}, \Delta)$ расстояние между агентами меньше $2 \cdot r$, т.е. возникает конфликт (столкновение агентов).

Отдельно отметим, что конфликтность действий определяется не только составом самих действий, но и моментами времени их начала. Одни и те же действия могут быть конфликтными при одних моментах начала их совершения и – неконфликтыми при других. Так, если в примере, указанном выше, действие одного из агентов начнётся существенно позже, например, в момент времени

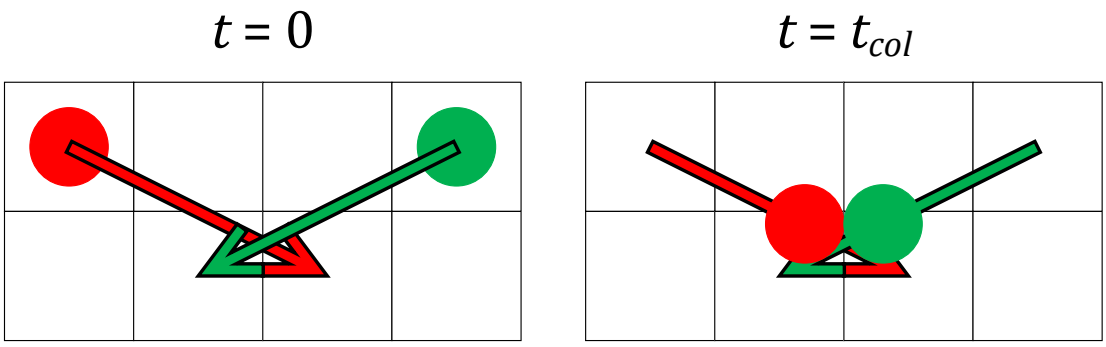


Рисунок 3.3 — Пример конфликта, возникающего в задаче AA-MAPF.

$t = \Delta$ (т.е. в момент завершения другим агентом действия перемещения), то по определению конфликта между такими действиями не возникнет.

Определение 34. Пути π^i и π^j, определенные в соответствии с 3.11, содержат конфликт если:

$$\exists(a^i_k, t^i_k) \in \pi^i, (a^j_l, t^j_l) \in \pi^j : \ con(a^i_k, t^i_k, a^j_l, t^j_l) = true \tag{3.15}$$

Заметим, что по этому определению конфликт с произвольным агентом, например i, не может возникнуть после завершения им последнего действия из π^i, т.к. согласно 3.14 после завершения действия конфликт не возникает.

Неформально это означает, что каждый агент, после достижения им целевого положения исчезает из рабочего пространства. Несмотря на кажущуюся иррациональность подобного допущения, именуемого в англоязычной литературе по много-агентному планированию как *disappear-at-target*, оно вполне разумно в определенных практических сценариях. Например, если речь идет о малых беспилотных летательных аппаратах, которые приземляются после достижения нужной позиции и более не создают помех другим аппаратам, оставшимся в общем воздушном пространстве.

Тем не менее, если от озвученного выше предположения необходимо отказаться и считать, что после достижения целевого состояния каждый агент остается в рабочем пространстве и, тем самым, ограничивает в действиях других агентов, то достаточно дополнить индивидуальный путь такого агента действием ожидания бесконечной продолжительности в целевой вершине. При этом это действие учитывается при определении конфликтом между путями, но не учитывается при подсчете продолжительности пути, в который оно добавлено. Такое предположение о поведении агентов на целевых вершинах обычно

именуется в англоязычной литературе по много-агентному планированию как *stay-at-target*. Именно это предположение считается верным в данной работе, если отдельно не оговорено иное.

После введения всех необходимых определений и предположений определим следующий вариант задачи построения множества неконфликтных путей на ГРД.

Определение 35. Задача построения множества неконфликтных путей, допускающих перемещения между произвольными вершинами ГРД, – это набор:

$$\text{AA-MAPF} = (\mathcal{G}, S, G, los, con) \tag{3.16}$$

Решение задачи AA-MAPF (от англ. **A**ny-**A**ngle **M**ulti-**a**gent **P**ath**f**inding) – множество путей П, допускающих перемещения между произвольными вершинами ГРД в соответствии с функцией los, т.ч. любая пара различных путей не содержит конфликтов (определяемых с помощью функции con).

Задача состоит в том, чтобы по заданному ГРД и набору начальных и целевых вершин, найти совокупность попарно неконфликтных путей, каждый из которых связывает одну из начальных вершин с соответствующей целевой вершиной и состоит из последовательности действий перемещения и ожидания. При этом для определения возможных действий перемещения используется функция los. Для определения неконфликтных действий (и, соответственно, путей) используется функция con. Важно отметить, что продолжительность каждого действия ожидания не фиксирована заранее (в отличие от действий перемещения), и определение необходимых действий ожидания и их продолжительностей является частью задачи. Оптимальным решением задачи, как и ранее, будем считать набор путей с минимальной суммарной продолжительностью, см. 3.7.

Пример задачи AA-MAPF и наглядное отличие её решения от решения задачи MAPF показаны на Рис. 3.4.

В этом примере результирующие пути для трёх агентов изображены разным цветом. Очевидно, что пути, образующие решение задачи MAPF (на рисунке – слева), содержат лишь переходы между смежными вершинами 4-связного ГРД, в то время как пути, образующие решение задачи MAPF (на рисунке – справа), содержат, в том числе, переходы между произвольными вершинами.

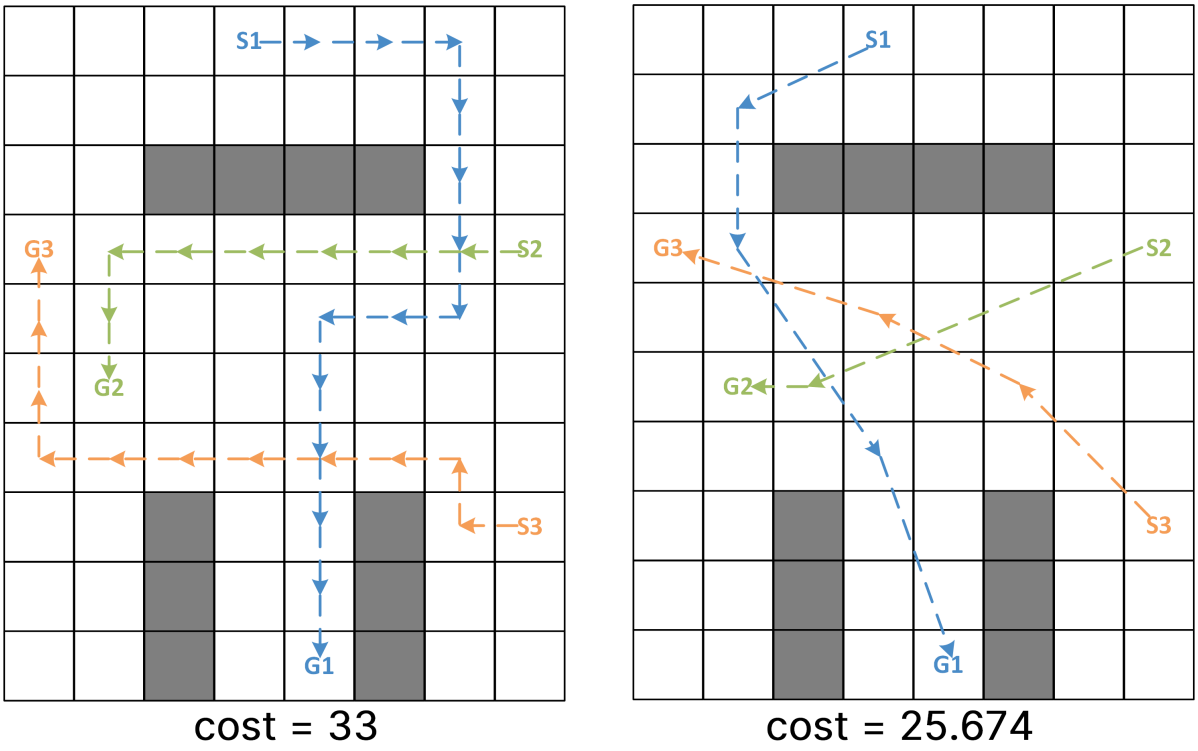


Рисунок 3.4 — Пример решения задач MAPF и AA-MAPF на одном и том же ГРД с одинаковым набором начальных и целевых вершин. Слева – решение задачи MAPF, справа – AA-MAPF.

Как результат стоимость решения задачи AA-MAPF (25.674) ниже, чем стоимость решения задачи MAPF (33). В данном случае разница составляет 22%.

3.2 Методы и алгоритмы

В предыдущем разделе была сформулирована расширенная постановка задачи поиска множества неконфликтных путей на ГРД – AA-MAPF и показана её связь с задачей (централизованного) много-агентного планирования траекторий. Опишем теперь предлагаемые в работе новые методы решения задачи AA-MAPF – метод поиска решения минимальной стоимости (оптимального решения), основанный на принципах конфликтно-ориентированного планирования, и более эффективный (в вычислительном смысле) метод поиска субоптимальных решений, основанный на принципе приоритизированного планирования. Заметим, что оба подхода опираются на использование ранее разработанных (и описанных в Главе 2) алгоритмов поиска пути на ГРД.

3.2.1 Конфликтно-ориентированное планирование для поиска оптимальных решений

Общая идея конфликтно-ориентированного планирования для поиска совокупности неконфликтных состоит в последовательном разбиении исходной задачи на ряд более легких (в вычислительном смысле) подзадач поиска индивидуальных путей, удовлетворяющих определенным ограничениям. Ограничения являются локальными, т.е. ограничивается совершение определенных действий в определенные интервалы времени. При этом за счет систематического разбиения задачи на подзадачи гарантируется, что полученное в итоге решение является минимальным по стоимости, т.е. оптимальным.

Изначально конфликтно-ориентированное планирование было предложено в работах [48; 179] для решения задачи MAPF. Описанный алгоритм был назван `CBS` (от англ. Conflict-Based Search). В дальнейшем было разработано целое семейство алгоритмов этого типа, таких как, например, `ICBS` [180], `CBS-H` [182], `ECBS` [185], `CBS-TAB` [221] и др. В подавляющем большинстве все они нацелены на решение задачи MAPF. Наиболее релевантным расширением `CBS` в контексте рассматриваемой задачи является алгоритм `CCBS` [222], в котором допускаются действия произвольной различной (в т.ч. произвольной) продолжительности. Именно этот вариант конфликтно-ориентированного поиска взят за основу предлагаемого в работе метода поиска оптимальных решений задачи AA-MAPF, который назван `AA-CCBS` (приставка AA обозначает any-angle).

Опишем далее предлагаемый в работе алгоритм `AA-CCBS` более подробно, начав с предварительного описания общих идей конфликтно-ориентированного планирования.

Ограничения в конфликтно-ориентированном планировании Одной из центральных идей конфликтно-ориентированного планирования является идея устранения конфликтов за счет наложения на агентов локальных *ограничений* и индивидуального перепланирования. Определение ограничения может отличаться от задачи к задаче. Так, например, когда речь идет о задаче MAPF, то обычно вводятся в рассмотрение ограничения вида (i, v, t) и (i, e, t), которые подразумевают невозможность для агента i занимать вершину v или переме-

щаться по ребру e на шаге t. Когда же речь идёт о задаче AA-MAPF, то целесообразно иное определение ограничений, которое приводится ниже.

Рассмотрим четверку $c = (a_k^i, t_k^i, a_l^j, t_l^j)$, такую что $con(c) = true$, т.е. между действиями a_k^i и a_l^j, совершаемыми агентами i и j в моменты времени t_k^i и t_l^j соответственно возникает конфликт. При этом будем считать, что $a_k^i \in A_{move}$ и $a_l^j \in A_{move}$, т.е. оба рассматриваемых действия является действиями перемещения (случай, когда имеется конфликт с действием ожидания, опишем позже).

Пусть теперь известна величина $\alpha > 0$, такая что:

$$\begin{aligned} &con(a_k^i, t_k^i + \alpha, a_l^j, t_l^j) = false \\ &\forall \alpha' \in (0,\alpha) : \ con(a_k^i, t_k^i + \alpha', a_l^j, t_l^j) = true \end{aligned} \tag{3.17}$$

То есть α – это минимальная задержка, которую агенту i необходимо совершить перед началом действия a_k^i, так чтобы конфликт с действием a_l^j агента j перестал существовать. Аналогично будем считать, что для агента j известна подобная величина β. Тогда *интервальное ограничение*, накладываемое на агента i для устранения конфликта $c = (a_k^i, t_k^i, a_l^j, t_l^j)$ определяется как:

$$\psi(i, c) = (a_k^i, [t_k^i, t_k^i + \alpha)). \tag{3.18}$$

Аналогично определяется и ограничение для агента j:

$$\psi(j, c) = (a_l^j, [t_l^j, t_l^j + \beta)). \tag{3.19}$$

Итак, перемещение агента запрещается к выполнению в определенный временной интервал. Длительность этого интервала минимальна с той точки зрения, что при начале совершения действия в любой из моментов интервала конфликт продолжает иметь место, а при совершении действия в момент, определяющий правую границу интервала, конфликта не происходит[1].

Расчет величин α и β на практике зависит от способа задания конфликта, которое в свою очередь зависит от способа задания функции, определяющей положения агента при совершении действия (см. 3.10). В рассматриваемом в работе случае, когда считается, что агенты движутся вдоль отрезков, соединяющих вершины ГРД, с постоянной скоростью, значения α и β могут быть

[1]Стоит отметить, что в целом подход конфликтно-ориентированного поиска допускает различные определения ограничений. Так в упрощенной постановке задачи (см. выше) когда время дискретно, и конфликты привязаны к конкретным дискретным тактам времени, то и ограничения могут быть определены не через интервалы, а через отдельные временные такты.

получены в аналитическом виде, т.е. рассчитаны по формулам – см. [223]. Здесь и далее будем предполагать, что способ расчета минимальных задержек, α или β, необходимых для устранения конфликта между парой действий перемещения известен.

Рассмотрим теперь случай, когда конфликт происходит между действиями ожидания и перемещения. Т.е. пусть теперь в четверке $c = (a_k^i, t_k^i, a_l^j, t_l^j)$, такой что $con(c) = true$, одно из действий, a_k^i или a_l^j, является действием ожидания[2]. Без ограничения общности будет считать, что $a_k^i \in A_{wait}$, т.е. действие ожидания совершает агент i (соответственно, агент j совершает действие перемещение). Продолжительность этого действия, как и раньше обозначим как $dur(a_k^i)$, а вершину, в которой совершается ожидание обозначим как $v_{wait}(a_k^i)$.

Пусть теперь $I(c) = (t_1,t_2)$ – временной интервал, т.ч.

$$\begin{aligned} &\forall t \in (t_1,t_2): \ dist(coord(v_{wait}(a_k^i), pos(t, a_l^j, t_l^j)) < 2 \cdot r \\ &\forall t \notin (t_1,t_2): \ dist(coord(v_{wait}(a_k^i), pos(t, a_l^j, t_l^j)) \geqslant 2 \cdot r \\ &(t_1, t_2) \cap [t_k^i, t_k^i + dur(a_k^i)] \neq \varnothing \end{aligned} \tag{3.20}$$

$I(c)$ – это конфликтный интервал, когда расстояние между агентами (один из которых ждёт, а другой – перемещается) меньше суммы их радиусов, что по определению (см. 3.14) означает наличие конфликта.

Расчет $I(c)$ на практике не представляет труда и осуществляется исходя из простых геометрических соображений: на плоскости определяются координаты пересечения прямой, вдоль которой движется агент j, и окружности радиуса $2 \cdot r$ с центром в $coord(v_{wait}(a_k^i))$. Моменты времени, в которые агент j достигает этих координат, и определяют $I(c)$. Заметим, что в общем случае эти моменты могут лежать как внутри, так и снаружи интервала действия ожидания $[t_k^i, t_k^i + dur(a_k^i)]$.

После определения $I(c)$ вводится в рассмотрение величина задержки (или же – смещения начала действия) δ следующим образом:

$$\delta = \min(\gamma \cdot |I(c)|, \ t_k^j + dur(a_k^j) - t_1), \tag{3.21}$$

где $\gamma \in (0,1)$ – произвольная константа.

[2]Случай, когда оба конфликтных действия являются действиями ожидания, не рассматривается, т.к. наличие такого конфликта необходимо влечет, что раньше по времени должен был возникнуть конфликт между действием перемещения и действием ожидания. И устранение последнего конфликта автоматически разрешает конфликт между двумя ожиданиями.

Теперь ограничения, накладываемые на агентов i и j могут быть определены следующим образом:

$$\begin{aligned}\psi(i,c) &= (v_{wait}(a_k^i), (t_1+\delta, t_2)) \\ \psi(j,c) &= (a_l^j, [t_l^j, t_l^j+\delta))\end{aligned} \tag{3.22}$$

Здесь $v_{wait}(a_k^i)$ – это вершина, в которой ожидает агент i, $I(c)$ – конфликтный интервал, определяемый в соответствии с 3.20, δ – задержка, определяемая в соответствии с 3.21.

Как и ранее (в случае конфликта между двумя действиями перемещения), накладываемые ограничения являются интервальными. Первое ограничение запрещает агенту i находится в вершине $v_{wait}(a_k^i)$ в любой из моментов времени $t \in (t_1+\delta, t_2)$. Второе ограничение запрещает агенту j начинать перемещение в любой из моментов времени $[t_l^j, t_l^j+\delta)$. Заметим, что первое ограничение запрещает как любое действие ожидания в вершине $v_{wait}(a_k^i)$, начинающееся в любой из моментов заданного конфликтного интервала, так и любое действие перемещения агента i, начинающееся в этой вершине (в этот же временной интервал).

После введения в рассмотрение интервальных ограничений (и способа их задания), опишем основные принципы конфликтно-ориентированного планирования.

Общая идея конфликтно-ориентированного планирования Конфликтно-ориентированное планирование представляет собой иерархический процесс, в котором разделяют верхний уровень и нижний уровень. Верхний уровень оперирует так называемым *деревом ограничений*. Узел (вершина) этого дерева – это пара: $N = (\Pi, \Psi)$, где Π – это набор из n индивидуальных путей, Ψ – это множество *ограничений*, в соответствии с которыми эти пути были построены. То есть любой из путей $\pi \in \Pi$ не содержит действий, определяемых ограничениями Ψ.

В начале процесса планирования создается корень дерева ограничений: $N_{root} = (\Pi_0, \varnothing)$, где список ограничений пуст, а Π_0 – есть совокупность индивидуальных путей, построенных без учета каких либо ограничений.

Далее происходит итеративный процесс роста дерева. На каждой итерации из листов дерева ограничений выбирается такой узел N_{best}, для которого стоимость набора индивидуальных путей минимальна (среди всех листов дерева). Затем набор путей, $N_{best}.\Pi$ проверяется на наличие конфликтов. Если

ни одного конфликта между любыми парами путей не существует, то решение найдено. Если же конфликты есть, то произвольным образом выбирается один из них $c = (a_k^i, t_k^i, a_l^j, t_l^j)$. Далее на вовлеченных в конфликт агентов накладываются ограничения: $\psi(i, c)$ и $\psi(j,c)$ (например так, как было показано выше). Создается два новых узла в дереве, которые будут являться потомками N_{best}: N_{left} и N_{right}. При этом в множество ограничений одного из потомков добавляется ограничение на агента i, т.е. $N_{left}.\Psi \leftarrow N_{best}.\Psi \cup \psi(i, c)$, а в множество ограничений другого из потомков добавляется ограничение на агента j: $N_{right}.\Psi \leftarrow N_{best}.\Psi \cup \psi(j, c)$

Далее для каждого из потомков происходит перепланирование индивидуального пути ограниченного агента. Т.е. для узла N_{left}, в котором ограничение накладывалось на агента i, происходит построение пути π^i с учетом всех ограничений из $N_{left}.\Psi$ (о том, как искать такой путь будет сказано отдельно). Если такого пути найти не удалось, то узел N_{left} не добавляется в дерево, если удалось – то N_{left} становится новым листом в дереве. Аналогичные операции выполняются для N_{right}.

Таким образом на каждой итерации конфликтно-ориентированного происходит попытка устранения конфликта за счет рассмотрения двух альтернативных ограничений: либо ограничивается один агент, при этом путь другого агента, входящего в конфликт, не меняется, либо – наоборот. Оба этих альтернативных варианта сохраняются в дереве ограничений для дальнейшего рассмотрения.

Рост дерева ограничений продолжается до тех пор, пока не будет создан лист дерева, содержащий набор неконфликтных путей (согласованный со всеми ограничениями, содержащимися в этом узле). Этот набор возвращается в качестве решения задачи.

Псевдокод Приведем ниже псевдокод алгоритма конфликтно-ориентированного планирования с интервальными ограничениями, обычно именуемый в литературе как `CCBS`.

Этап инициализации (Строки 1-4) включает в себя построение индивидуальных путей с помощью процедуры `FindPath`, которая принимает на вход ГРД, начальную и целевую вершины и множество ограничений, которое необходимо учесть при построении пути (на данном этапе это множество пусто). Также формируется дерево ограничений, содержащее одну корневую вершину,

Алгоритм `CCBS`($\mathcal{G}$, $s^1, s^2, ..., s^n$, $g^1, g^2, ..., g^n$)**:**

Входные данные: Граф $\mathcal{G}$, множество начальных вершин $s^1, s^2, ..., s^n$, множество целевых вершин $g^1, g^2, ..., g^n$

Выходные данные: Множество неконфликтных путей $\Pi = \{\pi^1, ..., \pi^n\}$

1 **foreach** $i \in \{1, ..., n\}$ **do**
2 $\pi_i \leftarrow$ `FindPath`$(\mathcal{G}, s^i, g^i, \varnothing)$
3 $\Pi_0 \leftarrow \{\pi_1, ..., \pi_n\}$; $\Psi_0 \leftarrow \varnothing$; $N_{root} \leftarrow (\Pi_0, \Psi_0)$
4 $OPEN \leftarrow \{N_{root}\}$
5 **while** $OPEN \neq \varnothing$ **do**
6 $N_{best} \leftarrow arg\min_{N \in OPEN} cost(N.\Pi)$
7 $c = (a_k^i, t_k^i, a_l^j, t_l^j) \leftarrow$ `GetConflict`$(N_{best}.\Pi)$
8 **if** $c = \varnothing$ **then**
9 **return** $N_{best}.\Pi$
10 **foreach** $z \in \{i, j\}$ **do**
11 $\psi_{new} \leftarrow$ `GetConstraint`(z, c)
12 $\Psi_{new} \leftarrow N_{best}.\Psi \cup \psi_{new}$
13 $\pi'_z \leftarrow$ `FindPath`$(\mathcal{G}, s^z, g^z, \Psi_{new})$
14 **if** $\pi'_z \neq \varnothing$ **then**
15 $\Pi_{new} \leftarrow N_{best}.\Pi \setminus \{\pi_z\} \cup \{\pi'_z\}$
16 $N_{new} \leftarrow (\Pi_{new}, \Psi_{new})$
17 $OPEN \leftarrow OPEN \cup \{N_{new}\}$

Рисунок 3.5 — Алгоритм конфликтно-ориентированного планирования.

которая добавляется в список OPEN (список листов дерева – кандидатов на дальнейшее рассмотрение).

Далее (Строки 5-17) осуществляется основной цикл алгоритма. Очередная итерация этого цикла начинается с выбора из множества OPEN узла N_{best} с минимальной стоимостью (Строка 6). Затем на множестве путей этого узла $N_{best}.\Pi$ ищется конфликт с помощью вспомогательной процедуры `GetConflict` (Строка 7). Если конфликт не обнаружен, то текущий набор путей $N_{best}.\Pi$ воз-

вращается в качестве решения задачи. В противном случае осуществляются шаги по устранению конфликта.

Поочередно, для каждого из агентов, формируется ограничение (Строка 11) с помощью вспомогательной процедуры `GetConstraint`, и это ограничение добавляется к уже имеющемуся множеству (Строка 12) – формируется обновленное множество ограничений для агента, Ψ_{new}. Далее ищется путь с учетом этих ограничений (Строка 13), и если этот путь найден, то он замещает собой старый путь. Формируется новый узел (листа) дерева ограничений (Строка 16), и он добавляется в список кандидатов на дальнейшее рассмотрение – список OPEN (Строка 17).

Итак, алгоритм `CCBS` реализует достаточно простую логику работы и опирается на 3 вспомогательные процедуры:

- `FindPath` – процедура поиска пути на ГРД, удовлетворяющего заданным ограничениям;
- `GetConflict` – процедура поиска конфликта (для последующего устранения) среди множества путей;
- `GetConstraint` – процедура формирования ограничения, необходимого для устранения найденного конфликта.

Процедура `GetConflict` может быть реализована произвольным способом, т.к. теоретические гарантии алгоритма не зависят от того, какой именно конфликт выбран для разрешения на очередной итерации основного цикла. Здесь и далее, будем предполагать, что эта процедура возвращает первый по времени конфликт.

Процедура формирования ограничений важна, т.к. от того, как именно формируются ограничения, зависят теоретические гарантии алгоритма. Интуитивно, наложение чрезмерно жестких ограничений может привести к тому, что решение не будет найдено (несмотря на то, что оно существует). Наложение же слишком мягких ограничений может приводить к отысканию не оптимальных решений.

На процедуру `FindPath`, осуществляющую поиск пути с учетом множества ограничений, накладывается требование по отысканию оптимального решения (т.к. только в этом случае можно доказать ряд свойств алгоритма конфликтно-ориентированного планирования).

Теоретические свойства алгоритма `CCBS` подробно рассматривались в [224—226]. В частности в этих работах доказано, что если процедура `FindPath`,

гарантирует построение индивидуального пути наименьшей стоимости с учетом сформированных ограничений, а сами интервальные ограничения формируются по формулам 3.18, 3.19, 3.22, то результирующий алгоритм гарантирует отыскание оптимального решения для любой решаемой задачи. Здесь и далее будем считать это верным (т.е. будем предполагать, что `FindPath` и `GetConstraint` реализованы соответствующим образом).

Ключевой процедурой является `FindPath` – процедура, которая осуществляет поиск пути с учетом множества ограничений. Важно, чтобы эта процедура гарантировала не просто отыскание пути, но построение оптимального пути (т.к. только в этом случае можно доказать ряд свойств алгоритма конфликтно-ориентированного планирования).

Алгоритм `AA-CCBS` Ранее в работе, в Главе 2, был представлен алгоритм `TO-AA-SIPP`, который гарантирует построение оптимального пути (с возможностью перемещения между произвольными вершинами) на динамическом ГРД, т.е. на ГРД с каждой вершиной и ребром которого ассоциировано множество безопасных интервалов. Заметим, что ограничения конфликтно-ориентированного поиска напрямую могут быть транслированы в эти интервалы. Опишем этот процесс более подробно.

В соответствии с определениями интервальных ограничений (задаваемыми формулами 3.18, 3.19, 3.22), любое из таких ограничений может быть представлено либо в виде $(a, [t_{start}, t_{end}))$, где $a = (u,v)$ – действие перемещения, либо в виде $(v_{wait}, (t_{start}, t_{end}))$, где v_{wait} – вершина ГРД, использование которой запрещается (в определенный интервал времени). Очевидно, что в первом случае ограничение индуцирует следующее множество безопасных интервалов для соответствующего перехода $u \rightarrow v$:

$$[0, t_{start}), [t_{end}, +\infty). \tag{3.23}$$

Во втором случае ограничение транслируется в аналогичную пару пару безопасных интервалов вершины v_{wait}.

Итак, становится возможной трансляция интервальных ограничений в множества безопасных и небезопасных интервалов, ассоциированных с вершинами ГРД и переходами между ними. Соответственно, возможно и использование алгоритма `TO-AA-SIPP` в качестве процедуры поиска индивидуальных путей

FindPath. Будем называть такой вариант алгоритма конфликтно-ориентированного планирования алгоритмом AA-CCBS (от англ. Any-angle Continuous-time Conflict Based Search).

Как было сказано ранее, для алгоритма CCBS доказано, что возвращаемое им решение является минимальным по стоимости, при условии того, что процедура FindPath всегда конструирует индивидуальный путь минимальной стоимости. Поскольку для алгоритма AA-CCBS это условие выполняется (что следует из доказанных в работе свойств алгоритма TO-AA-SIPP), справедливо следующее утверждение.

Утверждение 3. *Алгоритм* AA-CCBS *гарантирует отыскание решения задачи* AA-MAPF*, если оно существует, при этом данное решение является оптимальным (т.е. минимальным по стоимости).*

Повышение эффективности алгоритма AA-CCBS Алгоритмы конфликтно-ориентированного планирования, в том числе описанный выше алгоритм AA-CCBS, предполагает использование единственного ограничения, накладываемого на агента при обнаружении конфликта. То есть, если между агентами i и j, которые согласно своим текущим планам π_i, π_j совершают действия перемещения[3] a_k^i, a_l^j в моменты времени t_k^i, t_l^j, происходит конфликт $c = (a_k^i,t_k^i,a_l^j,t_l^j)$, то на агента i накладывается интервальное ограничение вида $\psi(i,c) = (a_k^i, [t_k^i, t_k^i + \alpha))$, где α – это минимальная задержка, которую необходимо выполнить перед началом совершения действия a_k^i, чтобы избежать конфликта с действием (a_l^j,t_l^j). На агента j аналогичным образом накладывается (единственное) интервальное ограничение $\psi(j,c) = (a_l^j, [t_l^j, t_l^j + \beta))$.

При этом известно, что при соблюдении определенных требований на агентов участвующих в конфликте могут быть наложены *мульти-ограничения* [221]. Формально, мульти-ограничение, ассоциированное с конфликтом c и накладываемое на агента i это множество индивиуальных ограничений:

$$\begin{aligned} &\boldsymbol{\Psi}(c,i) = \{\psi_1(i,c), \psi_2(i,c), ..., \psi_M(i,c)\}, \\ &\psi_m(i,c) = (a_m^i, [t_m^i, t_m^i + \alpha_m)), m \in \overline{1,M}, \\ &M \in \mathbb{N}. \end{aligned} \tag{3.24}$$

[3]Здесь и далее конфликты между действиями ожидания и перемещения не рассматриваются.

Здесь $\psi_m(i, c)$ – это индивидуальное интервальное ограничение, подразумевающее запрет для агента i совершения действия a^i_m в интервале $[t^i_m, t^i_m + \alpha_m)$, а M – общее число (различных) действий, входящих в мульти-ограничение.

То есть запрещается не единственное действие (перемещения), а сразу множество различных действий. При этом для каждого из ограничений, входящих в мульти-ограничение, определен свой собственный временной интервал, начало которого может не совпадать с t^i_k, т.е. с моментом времени начала совершения действия, приводящего к исходному конфликту (т.е. к тому конфликту, для устранения которого и накладывается мульти-ограничение).

Интуитивно мульти-ограничения имеют своей целью сокращение перебора вариантов действий, происходящего на верхнем уровне конфликтно-ориентированного планирования, что подтверждается результатами экспериментов [221]. В этой же работе приведен один из возможных способов формирования мульти-ограничений. Опишем его.

Пусть a^i_k, a^j_l – конфликтные действия перемещения. Будем перебирать все действия (перемещения), которые начинаются в тех же вершинах графа, что и a^i_k, a^j_l, для формирования двух множеств *взаимно-конфликтных действий* A_i и A_j. Множества действий будем называть взаимно-конфликтными, если любое действие из A_i, начинающееся в момент времени t^i_k приводит к конфликту с любым из действий A_j, начинающемуся в момент t^j_l. Теперь мульти-ограничение, накладываемое на агента i состоит из множества индивидуальных ограничений, каждое из которых имеет вид:

$$\psi_m(i, c) = (a^i_m, [t^i_k, t^i_k + \alpha')), \tag{3.25}$$

где действие a^i_m начинается в той же вершине, что и действие a^i_k (формально, $source(a^i_m) = source(a^i_k)$); конфликтный интервал начинается в момент времени t^i_k, который не зависит от m (т.е. начало конфликтного интервала является одинаковым для всех действий, входящих в мульти-ограничение); конфликтный интервал $[t^i_k, t^i_k + \alpha'))$ это максимальный по продолжительности интервал, такой что *полностью покрывает* все конфликтные интервалы индуцированные действием a_m и любым действием из A_j.

Будем ссылаться на подобное мульти-ограничение как на мульти-ограничение типа 1 или же MC1 (от multi-constraint type 1). Пример MC1 приведён на Рис. 3.6. Слева изображены два действия a_{11}, a_{12} (помечены желтым цветом) и два действия a_{21}, a_{22} (помечены красным), которые формируют множества

взаимно-конфликтных действий A_1 и A_2. Справа вверху перечислены индивидуальные конфликтные интервалы. Например, запись вида "a_{11} vs. a_{21}: $[0, 1.106)$" означает, что временной интервал ограничения, накладываемого на действие a_{11} относительно действия a_{21} составляет $[0, 1.106)$. По другим парам действий – аналогично. В правой нижней части рисунка перечислены результирующие мульти-ограничения.

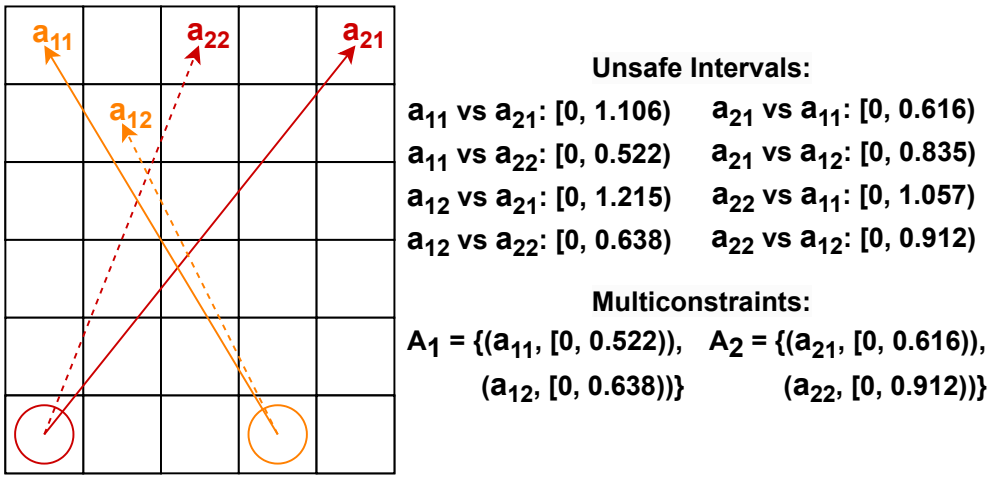


Рисунок 3.6 — Мульти-ограничение типа 1 (MC1).

Очевидно, что мульти-ограничение типа 1 с одной стороны запрещает выполнение сразу нескольких действий агенту i (j – аналогично), с другой стороны временной интервал каждого из образующих ограничений уменьшается. В приведенном выше примере у действия $(a_{11}, 0)$ конфликтный интервал относительно действия $(a_{21}, 0)$ равен $[0, 1.106)$, а относительно действия $(a_{22}, 0)$ – $[0, 0.522)$. По определению результирующий конфликтный интервал, ассоциированный с a_{11} должен покрывать оба этих интервала. То есть результирующий конфликтный интервал равен $[0, 0.522)$, что почти в два раза меньше, чем интервал относительно $(a_{21}, 0)$, продолжительность которого составляет 1.106. Эффект от наложения мульти-ограничения может нивелироваться – сокращения пространства поиска может не произойти из-за уменьшения продолжительности конфликтных интервалов (несмотря на то, что ограничения накладывается сразу на несколько действий).

Для купирования указанного недостатка предлагается следующий подход. Во-первых, предлагается включать в множество A_i (A_j – соответственно) не все действия, для которых начальная вершина совпадает с a_k^i, а лишь те, которые начинаются в той же вершине и заканчиваются в одной из вершин расположенных в непосредственной близости от отрезка $[source(a_k^i), target(a_k^i)]$.

Формально, пусть задана некоторая положительная константа $r > 0$, тогда множество A_i содержит действие перемещения a_m^i только, если:

$$\begin{aligned} &source(a_m^i) = source(a_k^i), \\ &dist(target(a_m^i), [source(a_k^i), target(a_k^i)]) \leqslant r, \end{aligned} \tag{3.26}$$

где $dist$ – это функция возвращающая кратчайшее расстояние (по Евклиду) от точки (первый аргумент функции) до отрезка (второй аргумент функции).

Иллюстрирующий этот принцип пример приведен на Рис. 3.7, в центральной части рисунка. В данном примере константа r соответствует радиусу безопасности агента, который задан изначально (а сам агент изображен в виде круга этого радиуса). Вершины, которые используются при формировании множества A_i, соответствуют клеткам красного цвета. Это те клетки, с которыми соприкасается агент при выполнении действия перемещения a_k^i, которое показано сплошной красной стрелкой.

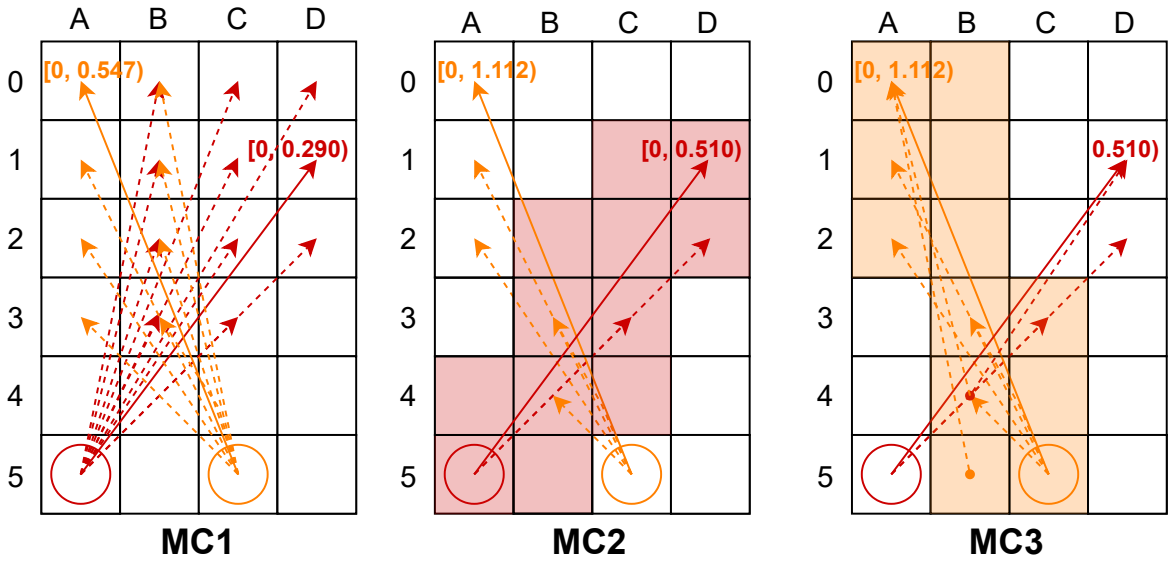


Рисунок 3.7 — Мульти-ограничение тима 1 (MC1) – слева, типа 2 (MC2) – по центру и типа 3 (MC3) – справа.

Далее, предлагается исключать из A_i (A_j – аналогично) те действия, которые приводят к сокращению исходного конфликтного интервала противоположного действия. То есть, если для некоторого действия a_m^i верно, что $unsafe(a_l^j, a_m^i) \not\subset unsafe(a_l^j, a_k^i)$, где $unsafe(a_l^j, a_m^i)$ – это конфликтный интервал (a_l^j, t_l^j) относительно (a_m^i, t_k^i), то a_m^i не включается в A_i. Таким образом в подобное мульти-ограничение входит, с одной стороны, меньше действий, с другой – исходный конфликтный интервал не сокращается. Достигается эффект

того, что добавляемые действия действительно дополняют исходное ограничение, не сокращая его интервал.

Будем называть мульти-ограничения, сформированные согласно описанным выше принципам, мульти-ограничениями типа 2 или MC2 (от англ. multi-constraint type 2). Псевдокод процедуры формирования MC2 приведён на Рис. 3.8. Для наглядности в псевдокоде используются обозначения, в которых индекс, указывающий на конкретное действие (k-е действие для агента i и l-е действие для агента j) опускается, т.е. вместо записей a_k^i, t_k^i, a_l^j, t_l^j используются записи a^i, t^i, a^j, t^j.

В Строках 1–3 происходит вычисление изначальных конфликтных интервалов и инициализация результирующих множеств действий A_i, A_j. Далее в Строке 4 происходит идентификация вершин графа, удаленных на расстояние меньшее либо равное r от отрезка, концы которого совпадают с начальной и конечной вершиной исходного действия, а именно действия a^i для агента i и действия a^j для агента j. С помощью идентифицированных вершин в Строке 5 происходит формирование множеств, состоящих из действий-кандидатов на дальнейшее включение в мульти-ограничение: $\hat{A}_i$, $\hat{A}_j$. Далее (Строка 6) эти множества объединяются в множество $\hat{A}$ так, что элементы из $\hat{A}_i$ и $\hat{A}_j$ чередуются в $\hat{A}$ (процедура такого объединения обозначена как *zip* в псевдокоде). Выполнение Строк 7–14 имеет своей целью отбрасывание действий-кандидатов, которые не удовлетворяют определению мульти-ограничения типа 2. Т.е. действие-кандидат $\hat{a} \in \hat{A}_i$ ($\hat{a} \in \hat{A}_j$ – аналогично) не отбрасывается, если оно имеет конфликт со всеми действиями из $\hat{A}_j$ (Строка 9)[4], во-вторых оно не сокращает исходный конфликтный интервал действия a^j относительно a^i (Строки 10–12). После того как фильтрация действий-кандидатов завершена, для каждого действия из $a \in \hat{A}_i$ (Строка 15) вычисляется результирующий конфликтный интервал (Строки 16–19) и с этим интервалом действие добавляется в A_i (Строка 20). Для действий из $\hat{A}_j$ – аналогично (Строка 21). После совершения вышеописанных операций образуются множества A_i, A_j, которые образуют мульти-ограничение типа 2 и эти мульти-ограничения и возвращаются процедурой (Строка 22).

Способы формирования мульти-ограничений, описанные выше, подразумевают включение в мульти-ограничение лишь тех действий, которые начинаются в тех же вершинах графа, что и исходные конфликтные действия a_k^i, a_l^j, т.е. в вершинах $source(a_k^i)$, $source(a_l^j)$. Однако, согласно определению 17, для

[4]Предполагается, что $\hat{a}$ начинается в момент t^i, а действие из $\hat{A}_j$ начинается в момент t^j.

Процедура `GetMultiContraints`$(a^i, t^i,\ a^j, t^j,\ r)$**:**

Входные данные: Действия и моменты времени, приводящие к конфликту: $a^i, t^i,\ a^j, t^j$, настраиваемый параметр r (радиус безопасности)

Выходные данные: Множества действий, формирующих мульти-ограничение типа 2: A_i, A_j

1 $[t^i, t') \leftarrow$ конфликтный интервал (a^i, t^i) относительно (a^j, t^j)

2 $[t^j, t'') \leftarrow$ конфликтный интервал (a^j, t^j) относительно (a^i, t^i)

3 $A_i \leftarrow \varnothing$; $A_j \leftarrow \varnothing$

4 $\hat{V}_{\{i,j\}} \leftarrow$ вершины графа, удаленные не более чем на расстояние r от отрезка $[source(a_{\{i,j\}}), target(a_{\{i,j\}})]$

5 $\hat{A}_{\{i,j\}} \leftarrow$ действия, начинающиеся в $source(a_{\{i,j\}})$ и заканчивающиеся в $\hat{V}_{\{i,j\}}$

6 $\hat{A} \leftarrow zip(\hat{A}_i, \hat{A}_j)$

7 **for each** $\hat{a} \in \hat{A}$ **do**

8 **if** $source(\hat{a}) = source(a_i)$ **then**

9 **if** *IsConflict*$(\hat{a}, \hat{A}_j)$=*true* **then**

10 $[t^j, \hat{t}) \leftarrow$ конфликтный интервал (a^j, t^j) отн. $(\hat{a}, t^i)$

11 **if** $t'' > \hat{t}$ **then**

12 $\hat{A}_i \leftarrow \hat{A}_i \setminus \{\hat{a}\}$

13 **else**

14 Обработка a^j аналогично (Строки 9-12)

15 **for each** $a \in \hat{A}_i$ **do**

16 $t_{min} \leftarrow \infty$

17 **for each** $a' \in \hat{A}_j$ **do**

18 $[t^i, t_{cur}) \leftarrow$ конфликтный интервал (a, t^i) отн. (a', t^j)

19 $t_{min} \leftarrow \min\{t_{min}, t_{cur}\}$

20 $A_i \leftarrow A_i \cup \{(a, [t^i, t_{min}))\}$

21 Обработка $\hat{A}_j$ аналогично (Строки 15-20)

22 **return** A_i, A_j

Рисунок 3.8 — Процедура формирования мульти-ограничения типа 2.

формирования мульти-ограничения это не обязательно. В связи с этим предлагается ещё один способ формирования мульти-ограничения для алгоритма AA-CCBS, который может рассматриваться как расширение мульти-ограничения типа 2. А именно предлагается включать в мульти-ограничение типа 2 для агента i (j – аналогично) действия, которые имеют своей целевой вершину вершину $target(a_k^i)$ (т.е. ту же вершину, в которой заканчивается исходное действие агента i), начальная вершина которых удалена не более, чем на расстояние r от отрезка $[source(a_k^i), target(a_k^i)$. Иллюстрирующий пример изображен на Рис. 3.7 справа.

Будем называть мульти-ограничения, сформированные согласно описанным выше принципам, мульти-ограничениями типа 3 или же MC3 (от англ. multi-constraint type 3). Псевдокод формирования MC3 аналогичен ранее приведенному коду формирования MC2 – см. Рис. 3.8, за исключением Строки 5. В случае построения MC3 эта строка выглядит так, как показано на Рис. 3.9.

$\hat{A}_{\{i,j\}} \leftarrow$ действия, начинающиеся в $source(a_{\{i,j\}})$ и заканчивающиеся в $\hat{V}_{\{i,j\}}$, либо – начинающиеся в $\hat{V}_{\{i,j\}}$ и заканчивающиеся в $target(a_{\{i,j\}})$

Рисунок 3.9 — Формирование множеств действий-кандидатов для последующего построения мульти-ограничения типа 3.

Итак, выше была рассмотрена техника, имеющая своей целью сокращение перебора на верхнем уровне алгоритма AA-CCBS и, следовательно, повышения быстродействия последнего, а именно техника формирования мульти-ограничений, когда ограничивается (для поиска нижнего уровня) не одно действие, а сразу несколько. При этом было предложено два новых способа формирования мульти-ограничений – MC2 и MC3, потенциально более эффективных нежели ранее известный способ, предложенные в [221] и называемый в данной работе MC1. Все три указанных способа опираются на общий подход, связанный с выделением двух множеств взаимно-конфликтных действий, и различаются в способах формирования этих множеств. Как было показано в [221] такой принцип формирования мульти-ограничений не влияет на гарантии отыскания алгоритмом конфликтно-ориентированного планирования оптимальных решений. Следовательно, справедливо следующее утверждение.

Утверждение 4. *Решение задачи* AA-MAPF, *возвращаемое алгоритмом* AA-CCBS+MCx, *является оптимальным.*

Здесь запись `AA-CCBS+MCx` означает, что алгоритм `AA-CCBS` использует мульти-ограничения типа x ($x \in \{1, 2, 3\}$).

Помимо механизма мульти-ограничений для повышения практической вычислительной эффективности конфликтно-ориентированного планирование может быть использована техника *взаимоисключающего разделения* – DS (от англ. disjoint splitting), см. [181]. В данной работе предлагается адаптировать эту технику для рассматриваемой постановки, т.е. для поиска решений задачи AA-MAPF (задачи построения совокупности неконфликтных путей на ГРД, которые могут содержать переходы между произвольными вершинами). Опишем в начале изначальную идею взаимоисключающего разделения, а затем её адаптацию для рассматриваемого случая.

Пусть на очередной итерации конфликтно-ориентированного планирования на верхнем уровне рассматриваемся узел дерева ограничений $N = (\Pi, \Psi)$, где $N.\Pi$ – это набор из n индивидуальных путей, $N.\Psi$ – множество ограничений, соответствии с которыми эти пути были построены. Пусть $N.\Pi$ содержит конфликт $c = (a_k^i, t_k^i, a_l^j, t_l^j)$ между агентами i и j, которые согласно своим текущим планам $\pi_i \in N.\Pi$, $\pi_j \in N.\Pi$ совершают действия перемещения a_k^i, a_l^j в моменты времени t_k^i, t_l^j. Затем формируются ограничения. В частности для агента i формируется интервальное ограничение вида $\psi(i, c) = (a_k^i, [t_k^i, t_k^i + \alpha))$, где α – это минимальная задержка, которую необходимо выполнить перед началом совершения действия a_k^i, чтобы избежать конфликта с действием (a_l^j, t_l^j). Для агента j аналогичным образом формируется интервальное ограничение $\psi(j, c) = (a_l^j, [t_l^j, t_l^j + \beta))$.

После формирования ограничений в дереве ограничений создаются два узла-потомка для N, а именно N_{left} и N_{right}, при этом $N_{left}.\Psi \leftarrow N.\Psi \cup \psi(i, c)$, а $N_{right}.\Psi \leftarrow N.\Psi \cup \psi(j, c)$. Т.е. в одном из потомков, N_{left}, для агента i запрещается выполнение действия a_k^i в интервал времени $[t_k^i, t_k^i + \alpha)$, а в другом из потомков, N_{right}, для агента j запрещается выполнение действия a_l^j в интервал $[t_l^j, t_l^j + \beta)$. То есть накладываемые ограничения можно атрибутировать как негативные (запрещается выполнение действия в определенный временной интервал).

При взаимоисключающем разделении накладываются другие ограничения. В узле-потомке N_{left} по-прежнему для агента i запрещается выполнение действия a_k^i в интервал времени $[t_k^i, t_k^i + \alpha)$, а в узле потомке N_{right} в дополнении к негативному ограничению на агента j дополнительно накладывается т.н. *по-*

зитивное ограничение на агента i: $\psi_{pos}(i,c) = (a^i_k, [t^i_k, t^i_k + \alpha))$, которое диктует, что агент i *должен* совершить действие a^i_k в один из моментов времени, принадлежащих интервалу $[t^i_k, t^i_k + \alpha)$. Таким образом, в одном из потомков, N_{left}, агент i не может совершать действие a^i_k в интервале $[t^i_k, t^i_k + \alpha)$, а в другом потомке, N_{right}, агент i обязан совершить это действие в этот же интервал времени. Такой способ наложения ограничений является, очевидно, более ограничивающим и потенциально ведущим к сокращению числа итераций верхнего уровня конфликтно-ориентированного планирования. На практике применение техники DS заметно повышает быстродействие алгоритма, что подтверждается результатами сравнительного экспериментального исследования, см. например [181; 227].

Описанная выше техника взаимоисключающего разделения может применяться и для решения задачи AA-MAPF предлагаемым алгоритмом `AA-CCBS`. Будем обозначать такой алгоритм как `AA-CCBS+DS`. Интерес так же представляет алгоритм `AA-CCBS`, использующий комбинацию взаимоисключающего разделения и мульти-ограничений. Один из способов такого комбинирования может быть связан с введением понятия позитивного мульти-ограничения, когда агенту необходимо выполнить любое из его формирующих действий. Такое позитивное ограничение легко определить и сформировать по описанным выше процедурам (см., например, Рис. 3.8). Однако учет такого ограничения на нижнем уровне планирования, т.е. встраивание такого позитивного мульти-ограничения в алгоритм `TO-AA-SIPP` – это нетривиальная и трудоемкая задача. Вместо этого предлагается более простой и элегантный (и при этом – эффективный) способ интеграции, использующий негативные мульти-ограничения и позитивные ограничения, но не позитивные мульти-ограничения.

Также как в алгоритме `AA-CCBS+DS`, при формировании узлов потомков N_{left} и N_{right} для текущего узла N, содержащего конфликт $c = (a^i_k, t^i_k, a^j_l, t^j_l)$ между агентами i и j, предлагается в узел N_{left} добавлять единственное негативное ограничение на действие a^i_k агента i: $\psi(i,c) = (a^i_k, [t^i_k, t^i_k + \alpha))$. Для узла же N_{right} предлагается добавлять позитивное ограничение на действие a^i_k агента i: $\psi_{pos}(i,c) = (a^i_k, [t^i_k, t^i_k + \alpha))$ и негативное (стандартное) мульти-ограничение для агента j, при формировании которого множество A_i состоит из единственного действия a^i_k (т.е. негативное мульти-ограничение для агента j формируется относительно единственного действия агента i, а не множества действий).

При таком подходе становится возможным комбинирование любой из предложенных выше техник формирования мульти-ограничений и техники взаимоисключающего разделения. Будем ссылаться на алгоритм, использующий такую комбинацию как на `AA-CCBS+DS+MCx`, где $x \in \{1,2,3\}$ – это тип мульти-ограничения, используемого алгоритмом.

3.2.2 Приоритизированное планирование и фокусировка поиска для эффективного построения субоптимальных решений задачи AA-MAPF

Ранее, в предыдущем разделе был предложен ряд алгоритмов, основанных на конфликтно-ориентированном планировании, обеспечивающих построение оптимальных решений задачи AA-MAPF. К сожалению, на практике применимость этих алгоритмов может быть ограничена, когда число отыскиваемых путей велико. При этом во многих практических применениях речь идёт о планировании сотен (или даже тысяч) путей. Это обстоятельство свидетельствует о целесообразности разработки методов решения задачи AA-MAPF, которые не обязательно гарантируют оптимальность отыскиваемых решений, но при этом являются более эффективными (в вычислительном плане) при решении задач с большим числом агентов. Ниже будут описаны предлагаемые в работе новые методы, предназначенные для решения этой задачи, а именно эффективные методы поиска субоптимальных решений задачи AA-MAPF.

Фокусировка поиска на верхнем уровне конфликтно-ориентированного планирования

Идея фокусировки поиска, и алгоритм её реализующий, известный в англоязычной литературе как `Focal Search` [12], уже описывалась в данной работе, в одном из пунктов раздела 2.2.4, применительно к задаче поиска пути на динамическом ГРД. Эта же идея может быть применена и для алгоритмов, реализующих подход конфликтно-ориентированного планирования, для ускорения

поиска за счет отказа от требования отыскания оптимального решения. Так, в работе [228] описывается, как фокусировка поиска может быть использована в комбинации с методом CBS [48] для решения задачи MAPF. Аналогичным образом может быть модифицирован предложенный в работе алгоритм AA-CCBS, предназначенный для решения задачи AA-MAPF. Псевдокод результирующего алгоритма, Focal-AA-CCBS, показан на Рис. 3.10.

Алгоритм Focal-AA-CCBS($\mathcal{G}$, $s^1, ..., s^n$, $g^1, ..., g^n$, w, h_{FOCAL}):

Входные данные: Граф $\mathcal{G}$, множество начальных $(s^1, ..., s^n)$, и целевых $(g^1, ..., g^n)$ вершин, фактор субоптимальности $w \geqslant 1$, эвристическая функция h_{FOCAL}

Выходные данные: Множество неконфликтных путей $\Pi = \{\pi^1, ..., \pi^n\}$

1 **foreach** $i \in \{1, ..., n\}$ **do**

2 $\quad \pi_i \leftarrow$ TO-AA-SIPP$(\mathcal{G}, s^i, g^i, \varnothing)$

3 $\Pi_0 \leftarrow \{\pi_1, ..., \pi_n\}$; $\Psi_0 \leftarrow \varnothing$; $N_{root} \leftarrow (\Pi_0, \Psi_0)$

4 $OPEN \leftarrow \{N_{root}\}$; $FOCAL \leftarrow \varnothing$

5 **while** $OPEN \neq \varnothing$ **do**

6 $\quad cmin \leftarrow \min_{N \in OPEN} cost(N.\Pi)$

7 $\quad FOCAL \leftarrow \{N | N \in OPEN, cost(N) \leqslant w \cdot cmin\}$

8 $\quad N_{best} \leftarrow arg\min_{N \in FOCAL} h_{FOCAL}(N)$

$\quad$ /* Стандартные шаги конфликтно-ориентированного планирования */

9

Рисунок 3.10 — Алгоритм конфликтно-ориентированного планирования для решения задачи AA-MAPF, использующий фокусировку поиска верхнего уровня

Алгоритм Focal-AA-CCBS во многом повторяет алгоритм AA-CCBS. Разница состоит в том, какой узел дерева ограничений выбирается на очередной итерации поиска. Напомним, что узел дерева ограничений N содержит в себе набор путей, $N.\Pi$, и ограничений, которым эти пути удовлетворяют, $N.\Psi$. Стоимость узла равна суммарной стоимости путей $cost(N) = cost(N.\Pi)$. В стандартном конфликтно-ориентированном планировании на очередной итерации из множества листов дерева ограничений, т.е. из списка OPEN, выбирает

лист с наименьшей стоимостью. Такой подход позволяет гарантировать оптимальность отыскиваемых решений. Алгоритм с фокусировкой – `Focal-AA-CCBS` – формирует (Строки 6–7) дополнительный список листов дерева, таких что их стоимость не превосходит минимальную стоимость листа, $cmin$, не более чем в w раз, где $w \geqslant 1$ – входной параметр алгоритма, т.н. фактор субоптимальности. Затем из этого списка, обычно именуемого FOCAL, происходит выбор узла для дальнейшей обработки, которая ни имеет отличий отличий от стандартного алгоритма. Этот выбор осуществляется с помощью эвристической функции h_{FOCAL}, которая передается на вход алгоритму. По аналогии с [185; 228] в данной работе предлагается использоваться следующую эвристическую функцию для выбора из FOCAL:

$$h_{FOCAL} = -\sum_{i=1}^{n} |N.\Psi_i| \tag{3.27}$$

Из FOCAL выбирается узел N с наибольшим числом ограничений, то есть – узел, для которого уже разрешено наибольшее число конфликтов (т.к. разрешение каждого конфликта влечет наложение ограничения). Потенциально такой узел ближе к искомому (бесконфликтному) решению по сравнению с остальными.

В [229] было доказано, что алгоритм `CCBS`, использующий фокусировку поиска, гарантирует что возвращаемое им решение задачи MAPF является ограниченно субоптимальным, т.е. его стоимость не превосходит стоимостью оптимального решения, более чем в w раз. Доказательство этого утверждения основано на том, что при использовании фокусировки в конфликтно-ориентированном планировании меняется лишь порядок рассмотрения узлов в дереве ограничений, но при этом нижний уровень планирования, т.е. поиск индивидуальных путей, удовлетворяющих ограничениям, остается оптимальным. Поскольку в рассматриваемом случае на нижнем уровне планирования используется алгоритм, гарантирующий отыскание оптимального решения, а именно – ранее предложенный в работе алгоритм `TO-AA-SIPP`, то справедливо следующее утверждение.

Утверждение 5. *Стоимость решения, отыскиваемого алгоритмом* `Focal-AA-SIPP` *для произвольной (имеющей решение) задачи* AA-MAPF*, не превосходит стоимости оптимального решения этой задачи, более чем в w раз, где $w > 1$ – параметр задаваемый пользователем (и передаваемый на*

вход алгоритму). Формально: $cost(\Pi_{\texttt{Focal-AA-SIPP}(w)}(p) \leqslant w \cdot cost(\Pi^*(p)))$, *где* Π_{alg} *– это решение* AA-MAPF *задачи* $p \in \mathbf{P}^+$[5], *а* $\Pi^*(p)$ *– это оптимальное решение этой же задачи.*

Имеет смысл сделать следующие два замечания, касательно описанного выше алгоритма `Focal-AA-CCBS`. Во-первых, ожидается, что на практике из-за фокусировки поиска алгоритм будет отличаться повышенной вычислительной эффективностью по сравнению с `AA-CCBS` (это подтверждается результатами экспериментов – см. дальнейший текст работы). Во-вторых, фокусировка поиска может быть скомбинирована с предложенными выше техниками наложениям мульти-ограничения и взаимоисключающего разделения, что, как ожидается, ещё более повысит вычислительную эффективность алгоритма (это предположение подтверждается результатами многочисленных экспериментов, которые будут описаны в последующих разделах работы).

Приоритизированное планирование для эффективного решения задачи AA-MAPF

Одним из широко известных в литературе и зарекомендовавших себя на практике подходом к решению задачи поиска совокупности неконфликтных путей на графе, в том числе на ГРД, является т.н. приоритизированное планирование [193; 230—232]. Идея этого подхода состоит в сведении сложной задачи поиска к серии простых подзадач, которые решаются последовательно. На первом этапе приоритизированного планирования каждому агенту (каждой паре (s^i, g^i)) назначается уникальный приоритет. Далее в соответствие с назначенными приоритетами происходит последовательное построение путей, при этом все ранее построенные пути считаются фиксированными и при построении текущего пути конфликтов со всем множеством этих путей необходимо избегать. Так происходит либо до тех пор пока не будет успешно построен последний путь (в этом случае множество построенных путей возвращается в качестве решения задачи), либо при построении очередного пути не выяснится, что его построить нельзя (в этом случае алгоритм возвращает специальный символ *failure*).

[5] Через $\mathbf{P}^+$ в данной работе обозначается множество задач, имеющих решение.

Важно отметить, что приоритизированное планирование, очевидно, не гарантирует отыскание оптимальных или даже ограниченно субоптимальных решений. В общем случае при приоритизированном планировании стоимость отыскиваемого решения ничем не ограничена. Однако, стоит заметить, что для большинства практических задач качество отыскиваемых решений весьма велико (т.е. их стоимость весьма умеренно превышает стоимость оптимальных решений). Помимо этого, общем случае планирование с приоритетами не гарантирует отыскание решения задачи, даже если оно (такое решение) существует. Пример задачи, которая не может быть успешно решена с помощью приоритизированного планирования, изображен на Рис. 3.11.

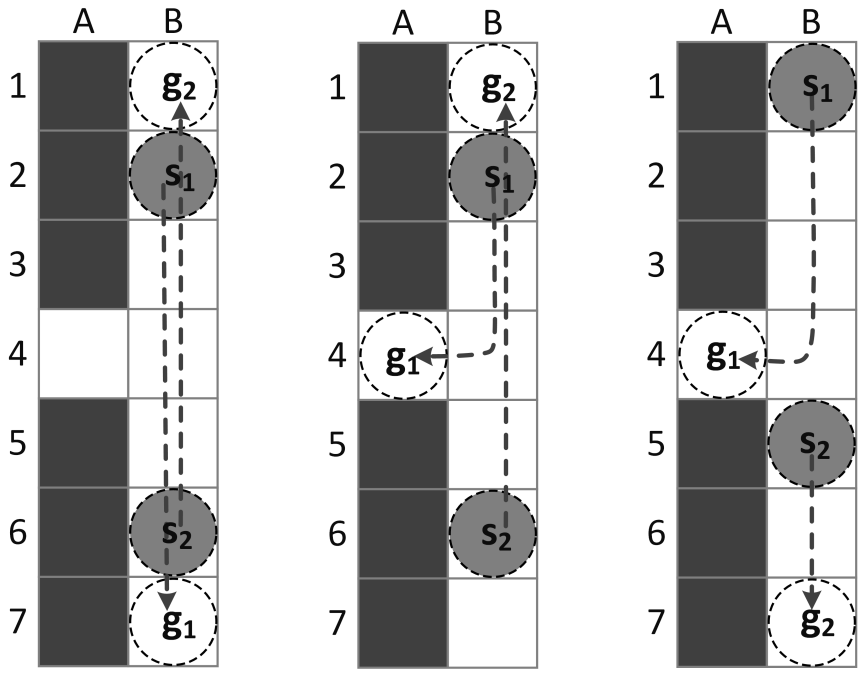


Рисунок 3.11 — Примеры нерешаемых (слева) и решаемых (по центру и справа) задач для приоритизированного планирования.

В изображенном примере необходимо построить лишь два неконфликтных пути (для агента 1 и для агента 2). Однако, если их начальные и целевые позиции расположены так, как показано на рисунке слева, то вне зависимости от того как будут назначены приоритеты (сначала строится индивидуальный путь для агента 1, а затем для агента 2, или – наоборот), решения задачи не будет найдено. Заметим, что при ином расположении начальных и целевых вершин, а именно так, как показано на центральной части рисунка, решение задачи будет найдено если путь для агента 1 будет спланирован раньше, чем путь для агента 2, но не наоборот. Если же начальные и целевые вершины рас-

положены так, как показано на рисунке справа, то решение будет найдено при любом варианте приоритизации.

Итак, задачи много-агентного планирования с точки зрения приоритизированного подхода к их решению можно разбить на три класса: нерешаемые задачи (Рис. 3.11 слева), условно-решаемые задачи (Рис. 3.11 по центру), безусловно-решаемые задачи (Рис. 3.11 справа). В связи с этим целесообразным представляется разработка и исследование подходов, направленных на повышение эффективности приоритизированного планирования для задач второго класса, т.е. тех задач, которые могут быть решены с помощью подходящей приоритизации и/или других техник. Именно такие подходы и будут описаны далее.

Безопасные интервалы в начальных вершинах По умолчанию в приоритизированном планировании при поиске i-го пути, π_i, все пути, построенные до этого, т.е. $\pi_1, \pi_2, ..., \pi_{i-1}$, входят в множество ограничений, т.е. конфликтов с этим путями нужно избегать. Более никаких ограничений на π_i не накладывается. В частности π_i может проходить через начальные вершины путей, которые будут планироваться впоследствии (на следующих итерациях приоритизированного планирования), т.е. через вершины s^j, $i < j \leqslant n$. Это может привести к тому, что последующие построение неконфликтных путей π^j станет невозможным.

Рассмотрим пример, изображенный на Рис. 3.12.

В левом верхнем углу рисунка показаны начальные и целевые позиции для трех агентов и отмечены их приоритеты. На первой итерации приоритизированного планирования будет построен и зафиксирован путь π^1, который показан на рисунке в правом верхнем углу. Затем будет найден и зафиксирован путь π^2, который изображен в левом нижнем углу. Теперь при поиске пути π^3 из $D2$ в $A3$ планировщик вернет *failure*. Действительно, т.к. агент 2 сразу же проходит через начальную вершину агента $s^3 = D2$, то агенту 3 остается лишь переместиться вниз в клетку $D3$, однако при таком перемещении возникнет конфликт с агентом 1. Следовательно, алгоритм приоритизированного планирования завершится, не отыскав решения задачи.

Одна из причин, по которой решение в указанном примере не будет найдено, состоит в том, что пространство начальных действий низкоприоритизированных агентов может существенно сократиться из-за того, что пути высокоприоритизированных агентов пролегают через их начальные вершины.

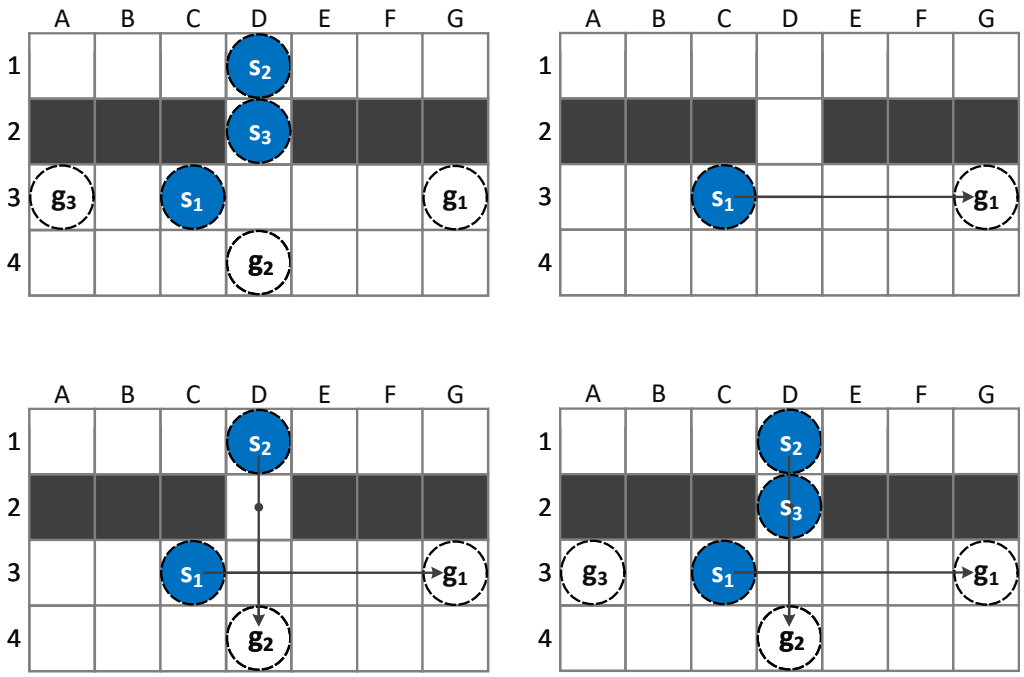


Рисунок 3.12 — Пример задачи, решение которой невозможно с помощью приоритизированного планирования (при заданных приоритетах) из-за конфликта возникающего в начальной вершине одного из низко-приоритизированных агентов.

Для решения этой проблемы предлагается ввести понятие гарантированно-безопасного начального интервала начальных вершин. То есть – добавить в множество безопасных интервалов *всех* начальных вершин, $s^1, s^2, ..., s^n$, интервал $[0, t_{ssi}]$[6], где t_{ssi} – настраиваемый параметр. Интуитивно, параметр t_{ssi} определяет сколько времени любой из агентов, вне зависимости от приоритета, может безопасно ждать в начальной вершине.

Легко видеть, что при $t_{ssi} > 2$ описанный выше пример имеет решение, т.к. путь агента 2 будет содержать действие ожидания в $D1$ перед переходом в $D2$ (т.к. с $D2$ теперь ассоциирован безопасный интервал $[0, t_{ssi}]$). При этом становится возможным построить безопасный путь и для агента 3 следующим образом. Сначала агент 3 ждёт в $D2$, затем – перемещается вниз в $D3$ и далее в $A3$. Следовательно, даже без изменения приоритетов проблема становится решаемой.

Стоит заметить, что при решении практических задач значение параметра t_{ssi} подбирается экспериментально. С одной стороны, чем больше это значение, тем больше шансов у низкроприоритизированных агентов найти решение (т.к. они могут продолжительное время безопасно ждать в своих начальных верши-

[6]Нижний индекс *ssi* образован от англ. start safe interval, т.е. начальный безопасный интервал.

нах). С другой стороны увеличение параметра t_{ssi} может негативно сказаться на стоимости отыскиваемых решений. Более того, если $t_{ssi} = \infty$ (то есть высокоприоритизированным агентам полностью запрещено использовать начальные вершины низкоприоритизированных агентов при построении своих путей), то некоторые задания могут оказаться нерешаемыми. В качестве примера рассмотрим опять задание, изображенное на Рис. 3.12. При $t_{ssi} = \infty$ построение пути для агента 2 невозможно, т.к. для того, чтобы попасть из его начальной вершины $D1$ в его целевую вершину $D4$ ему необходимо совершить переход в вершину $D2$, однако этого невозможно, если безопасный интервал этой вершины не ограничен.

Переназначение приоритетов Как было описано выше, результат приоритизированного планирования может быть весьма чувствителен к способу начальной приоритизации агентов. Решение может существовать при одном назначении приоритетов и не существовать при другом. Очевидно, что при необходимости построения n путей всего существует $n!$ различных приоритизаций. Это означает, что на практике перебрать их все поисках наиболее подходящей приоритизации для конкретной решаемой задачи весьма затруднительно.

Одним из вариантов решения проблемы, известным в литературе, см., например, [194; 195], является случайное переназначение приоритетов для $k \leqslant n$ агентов. То есть, если с текущими приоритетами решение найти не удалось (и при этом не истекло время, отведенное на планирование), происходит случайное переназначение приоритетов для $k \leqslant n$ агентов и планирование осуществляется заново. Основное преимущество этого подхода заключается в его простоте. Недостатками являются, во-первых, необходимость выбора параметра k, во-вторых, недетерминированность алгоритма, т.е. решение, получаемое на одних и тех же входных данных может отличаться от запуска к запуску, что может быть нежелательным на практике.

Для устранения указанных недостатков, в данной работе предлагается использовать детерминированную технику переназначения приоритетов, не требующую параметризации. Эта техника основана на простом наблюдении – при приоритизированном планировании всегда известна причина, по которой не удалось построить решение, а именно – известен агент, для которого не удалось найти путь (избегающий конфликтов с путями высокоприоритизированных агентов). Поэтому логичным способом изменения приоритетов является

повышение приоритета этого агента до наивысшего, и понижение приоритетов других агентов на единицу.

Псевдокод Приведем теперь псевдокод предлагаемого алгоритма приоритизированного планирования для решения задачи AA-MAPF с использованием техник и улучшений, описанных выше. В качестве алгоритма планирования индивидуальных путей предлагается использовать ранее предложенный в работе алгоритм `AA-SIPP` – высокоэффективный (в вычислительном смысле) алгоритм построения пути на динамическом ГРД (т.е. на графе часть вершин и ребер которого недоступна в определенные интервалы времени). Этот алгоритм не гарантирует отыскание оптимальных решений, однако, как было показано выше, сам по себе подход приоритизированного планирования является субоптимальным, поэтому использование `AA-SIPP` вполне целесообразно и уместно.

В Строках 1–4 происходит инициализация, включающая в себя назначение начальных приоритетов (с помощью внешней по отношению к алгоритму процедуры `SetInitialPriorities`); добавление безопасного интервала $[0, t_{ssi}]$ в множества безопасных интервалов всех вершин графа; формирование множества (пока пустого) рассмотренных приоритизаций PS.

Далее идет основной цикл работы алгоритма (Строки 5–20). Каждая итерация начинается с добавления текущего набора приоритетов ps в множество рассмотренных приоритизаций PS (Строка 6) и инициализации решения – набора некофликтных путей Π, – пустым множеством (Строка 7). Далее осуществляется индивидуальное планирование в порядке приоритетов (Строки 8–20). Для поиска индивидуального пути с приоритетом i_j (Строка 9) используется алгоритм `AA-SIPP`, ранее предложенный в работе[7].

Если путь найден (Строка 10), то он добавляется с множество построенных путей Π (Строка 11) и используется для обновления безопасных интервалов вершин и ребер графа (Строка 12), с тем чтобы пути, построенные на последующих итерациях не приводили к конфликтам между агентами. Если $j = n$, то есть успешно найден путь для последнего агента, то алгоритм завершает свою работу и в качестве решения задачи возвращает Π (Строка 14).

Если путь не найден (Строка 15), то происходит детерминированное переназначение приоритетов, а именно приоритет текущего агента становится

[7]Заметим, что сигнатура вызова `AA-SIPP` несколько упрощена, по сравнению с сигнатурой, ранее использованной в работе в разделе 2.2.4. Это сделано для упрощения восприятия. По факту алгоритму `AA-SIPP` передаются все необходимые аргументы – см. раздел 2.2.4.

Алгоритм `AA-SIPP(m)`($\mathcal{G}$, $s^1, ..., s^n$, $g^1, ..., g^n$, t_{ssi})**:**

Входные данные: Граф $\mathcal{G} = (V,E)$, множество начальных ($s^1, ..., s^n$), и целевых ($g^1, ..., g^n$) вершин, продолжительность начального безопасного интервала t_{ssi}

Выходные данные: Множество неконфликтных путей $\Pi = \{\pi^1, ..., \pi^n\}$

```
1  ps = {i_1, i_2, ..., i_n} ← SetInitialPriorities()
2  foreach v ∈ V do
3      T_safe(v) ← {[0, t_ssi]}
4  PS ← ∅
5  while true do
6      PS ← PS ∪ {ps}
7      Π ← ∅
8      for j from 1 to n do
9          π_j ← AA-SIPP(G, s^{i_j}, g^{i_j}, {T_safe(v)})
10         if π_j is found then
11             Π ← Π ∪ {π_j}
12             UpdateSafeIntervals(G, π_j)
13             if j = n then
14                 return Π
15         else
16             ps ← {i_1 = i_j, i_2 = i_1, ..., i_n = i_{n-1}}
17             if ps ∈ PS then
18                 return failure
19             else
20                 break
```

Рисунок 3.13 — Алгоритм приоритизированного планирования для решения задачи AA-MAPF, использующий предлагаемые в работе техники динамического переназначения приоритетов и безопасные интервалы в начальных вершинах.

максимальным, а всех остальных понижается (Строка 16). Если такой набор приоритетов уже рассматривался ранее, то алгоритм завершает свою работу и

возвращает специальный символ *failure* – решения найти не удалось (Строка 18). В противном случае, осуществляется переход к очередному циклу приоритизированного планирования с обновленными приоритетами (Строка 20).

3.3 Экспериментальные исследования

Для проведения экспериментальных исследований предложенных в данной главе методов они были программно реализованы на языке C++ и с использованием этих реализаций был проведен ряд численных (модельных) экспериментов. Отдельный эксперимент состоял в запуске одного из реализованных алгоритмов и их аналогов (при наличии), на отдельном задании, которое представляло собой набор начальных и целевых вершин на фиксированном ГРД. Время, отводимое алгоритму на решение задачи ограничивалось разумной величиной (т.к. зачастую решение заданий с большим числом агентов требовало чрезмерно долгого времени, что затрудняло массовое тестирование). В ходе проведения эксперимента отслеживались и фиксировались различные индикаторы, характеризующие эффективность алгоритма, такие как время отыскания решения и стоимость построенного решения. Далее результаты работы алгоритмов агрегировались и сравнивались в таком виде между собой, на основании чего делались определенные выводы о качестве работы алгоритмов. При проведении экспериментов активно использовалась распространенная в этой предметной области коллекция заданий MovingAI [219], содержащая множестве различных ГРД и начальных/целевых вершин для этих ГРД.

Помимо численных экспериментов в сотрудничестве с коллегами из НИЦ “Курчатовский институт” были проведены экспериментальные исследования одного из наиболее эффективных в вычислительном смысле алгоритмов, а именно алгоритма `AA-SIPP(m)`, на реальных колесных роботах.

Опишем теперь ход и результаты проведенных экспериментальных исследований более подробно.

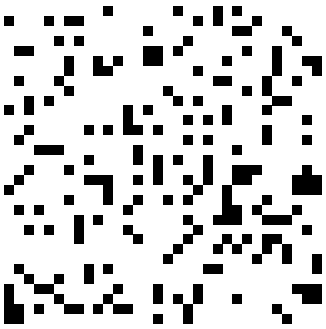
3.3.1 Экспериментальные исследования методов решения задачи AA-MAPF на основе конфликтно-ориентированного планирования

В рамках этой серии экспериментов проводились исследования алгоритма `AA-CCBS` и его различных модификаций, предложенных в работе. Сначала было проведено исследование непосредственно алгоритма `AA-CCBS` и его модификаций, гарантирующих отыскание решений минимальной стоимости, затем – модификаций, гарантирующих отыскание ограниченно-субоптимальных решений. Поскольку на момент проведения работы прямые аналоги этих алгоритмов в мире отсутствовали (т.е. не были известны альтернативные алгоритмы, гарантирующие отыскание оптимальных или ограниченно-субоптимальных решений задачи AA-MAPF), то другие алгоритмы в ходе экспериментального исследования не рассматривались.

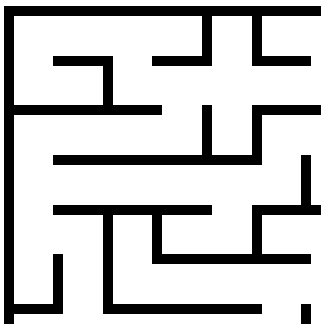
Для проведения экспериментов использовались 5 ГРД различного размера и топологии, являющихся частью широко-используемой в сообществе коллекции MovingAI [219]: empty-16-16, random-32-32-20, maze-32-32-4, den321d, warehouse-10-20-10-2-2. Будем их именовать в дальнейшем просто как empty, random, maze, den321d, warehouse и ссылаться на эти ГРД так же как на карты.

Первая карта, empty, имеет размер 16×16 клеток и не содержит препятствий. Вторая карта, random, имеет размер 32×32 клетки, из которых выбранные случайным образом 20% клеток являются заблокированными. Размер карты maze так же составляется 32×32, и она представляет из себя лабиринт. Следующая карта, den312, имеет размер 65×81 и моделирует несколько помещений (комнат), соединенных проходами. Наконец, карта warehouse имеет размер 170×84 и моделирует склад, состоящий из 200 продольных стеллажей с узкими коридорами между этими стеллажами и свободными зонами по краям карты. Графическое представление используемых в экспериментах карт изображено на Рис. 3.14.

С каждой картой в коллекции MovingAI ассоциирован набор из 25 так называемых сценарных файлов (сценариев). Отдельный сценарий содержит конечное (достаточно большое) число пар “начальное - целевое положение”. Согласно общепринятой методологии (см. [26]) проведение экспериментов с использованием сценариев организовано следующим образом. Сначала из файла-сценария извлекаются первые две пары начальных и целевых вершин.

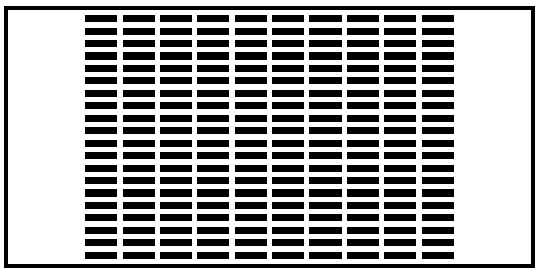


а) Карта random (32 × 32)



б) Карта maze (32 × 32)

в) Карта den312 (65 × 81)



г) Карта warehouse (170 × 84)

Рисунок 3.14 — Карты, используемые в экспериментальном исследовании алгоритма AA-CCBS и его модификаций.

На получившемся задании запускается алгоритм планирования с ограничением по времени работы. Если алгоритм отыскивает решение, то к множеству начальных и целевых вершин добавляется следующая пара из файла и алгоритм опять запускается (с тем же ограничением по времени работы). Так происходит до тех пор пока алгоритм не уложится в отведенное ему время. Когда это происходит, эксперименты с данным сценарием прекращаются и берется следующий сценарий. В данной работе ограничение на время решения отдельного задания составляло 300 секунд.

Эксперименты проводились с базовым алгоритмом AA-CCBS и его 5 различными модификациями, а именно: AA-CCBS+MC1, AA-CCBS+MC2, AA-CCBS+MC3, AA-CCBS+DS, AA-CCBS+DS+MC3. Далее на графиках базовая версия алгоритма будет обозначаться как Vanilla, а модифицированные версии будут обозначаться сокращенно, например, алгоритм AA-CCBS+DS будет обозначаться просто как +DS (другие алгоритмы – аналогично).

На Рис. 3.15 изображена диаграмма, показывающая сколько заданий в отведенное время решил тот или иной алгоритм (чем больше, тем лучше). Разные

столбцы соответствуют различным версиям алгоритма. Ось Y – это число заданий, решение для которых удалось найти алгоритму в отведенный временной лимит. Разными цветами отмечены задания на разных картах.

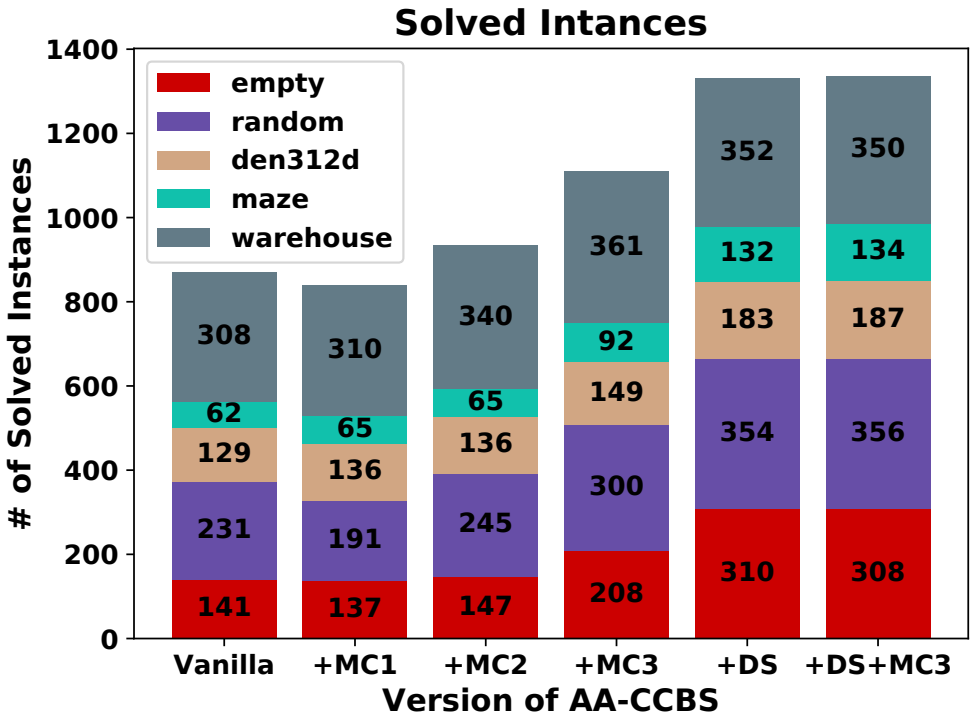


Рисунок 3.15 — Число успешно решенных заданий в отведенный временной лимит для различных версий алгоритма `AA-CCBS`.

Анализируя этого график можно сделать следующие выводы. Во-первых, алгоритм, использующий известный ранее в литературе способ формирования мульти-ограничений (MC1), по производительности уступает даже базовому алгоритму AA-CCBS без использования мульти-ограничений: число заданий, решенных `AA-CCBS+MC1` ниже, чем у `AA-CCBS`). При этом ранее, в работе [221], приводились данные, свидетельствующие о том, что MC1 положительно сказывается на производительности конфликтно-оринетированного планирования. Это противоречие легко объясняется тем, что в данной работе рассматривается задача AA-MAPF, в которой допускается перемещение между произвольным (в т.ч. удаленными друг от друга) вершинами ГРД, в отличие от рассматриваемой в работе [221] задачи MAPF. Это подтверждает тот факт, что задача AA-MAPF существенно отличается от стандартной задачи MAPF, и решения и техники, которые хорошо зарекомендовали себя на практике при решении задачи MAPF могут не давать нужного эффекта при решении задачи AA-MAPF.

Далее, очевидно, что способы формирования мульти-ограничений, предложенные в работе (MC2 и MC3), повышают производительность базового

алгоритма – число успешно решенных заданий заметно возрастает. Так AA-CCBS решает 871 против 1110 у AA-CCBS-MC3, т.е. прирост составляет 27%.

При этом наибольшее повышение эффективности алгоритма достигается при использовании техники взаимоисключающего разделения (DS). Число успешно решенных заданий для алгоритма AA-CCBS+DS составляет 1331 (прирост 53%). Комбинирование же техники взаимоисключающего разделения с наложением мульти-ограничений повышает число успешно решенных заданий, но весьма незначительно: 1335 решенных заданий для алгоритма AA-CCBS+DS+MC3[8].

Проведем более детальный анализ вычислительной эффективности всех исследуемых алгоритмов. Для этого построим график, который бы демонстрировал, сколько заданий решил тот или иной алгоритм за время $t \in [0, 300]$ c. Такой график представлен на Рис. 3.16.

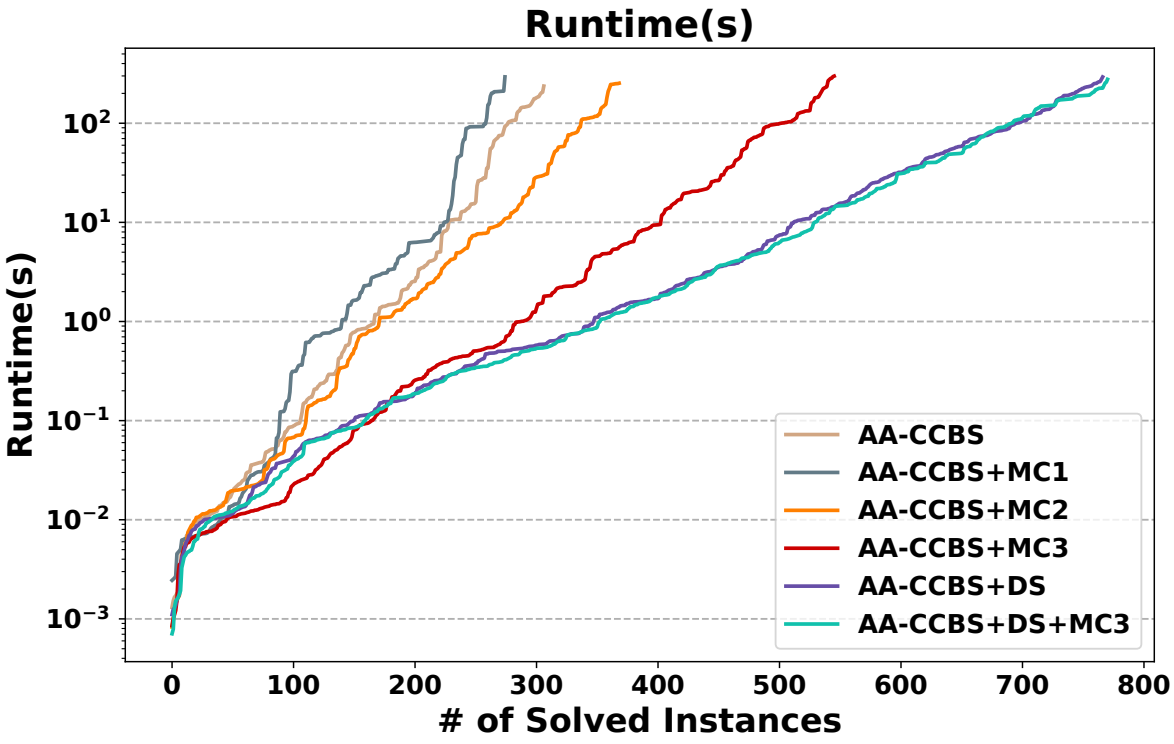


Рисунок 3.16 — Число успешно решенных заданий за t секунд для различных версий алгоритма AA-CCBS.

На графике по вертикальной оси отложено время (шкала логарифмическая), горизонтальная соответствует числу успешно решенных заданий. То есть точку (x, t) на этом графике следует интерпретировать так: алгоритм решает x

[8]В ходе экспериментов проводилось комбинирование техники DS и с другими способами наложения мульти-ограничений, однако результаты не улучшались.

заданий при временном лимите t секунд. Разные линии соответствуют различным версиям алгоритма AA-CCBS: чем правее и ниже расположена линия, тем лучше. При этом при построении этого графика не учитывались задания, для решения которых не было необходимости накладывать ограничения на верхнем уровне конфликтно-ориентированного планирования (т.е. такие задания, которые достаточно решить просто независимым построением путей). Этим объясняется тот факт, что ось X заканчивается на значении чуть меньшем 800 (в то время, как число успешно решенных заданий, показанное на Рис. 3.15 больше).

Анализируя этот график можно сделать следующее заключение. Несмотря на то, что техника взаимоисключающих ограничений превосходит по эффективности технику мульти-ограничений при больших значениях параметра t (времени отводимого алгоритму на решение задачи), ситуация меняется на противоположную, когда $t \in [0.01, 0.1]$ $c.$ – красная линия, соответствующая алгоритму AA-CCBS+MC3 находится заметно правее линии, соответствующей алгоритмам AA-CCBS+DS и AA-CCBS+MC3, когда $t \in [0.01, 0.1]$. То есть, если отводимое время на решение задачи AA-MAPF мало (например, меньше 0.1 $c.$), то предпочтение следует отдавать алгоритму конфликтно-ориентированного планирования, использующему предложенный в работе способ формирования мульти-ограничений MC3.

Во второй серии экспериментов исследовались модификации алгоритма AA-CCBS, использующую фокусировку поиска для построения ограниченно-субоптимальных решений, т.е. таких решений, стоимость которых не превосходит стоимость оптимального решения более чем в w раз, где w – это входной параметр алгоритма. В рамках этих экспериментов полагалось $w \leftarrow \{1.0, 1.01, 1.1, 1.25\}$. Т.е. целью было как отыскание оптимальных решений ($w = 1$), так и решений, превосходящих оптимальные по стоимости не более, чем на 1% ($w = 1.01$), на 10% ($w = 1.1$), на 25% ($w = 1.25$). Исследовались алгоритмы AA-CCBS, AA-CCBS+MC3, AA-CCBS+DS, AA-CCBS+DS+MC3 с фокусировкой поиска.

Использовалась та же методология проведения эксперимента, что и ранее (те же карты и задания, тот же лимит на время выполнения алгоритма 300 $c.$). Результаты приведены в Табл. 3.1 и Рис. 3.17.

Как явно следует из Табл. 3.1 фокусировка поиска позволяет существенно повысить эффективность базового алгоритма. Так, например, при уже при $w = 1.01$ (т.е. когда допускается превышение стоимости оптимального решения

Таблица 3.1 — Число успешно решенных заданий (за отведенный лимит времени) различными модификациями алгоритма AA-CCBS, использующими фокусировку поиска.

	$w = 1.0$	$w = 1.01$	$w = 1.1$	$w = 1.25$
AA-CCBS	871	1653	1929	1950
AA-CCBS+MC3	1110	**2002**	**2437**	**2440**
AA-CCBS+DS	1331	1661	1583	1564
AA-CCBS+DS+MC3	1335	1626	1561	1543

лишь на 1%), алгоритм AA-CCBS решает почти в два раза больше заданий: 1653 против 871 (для $w = 1$). Схожий прирост в производительности демонстрирует и алгоритм, использующий мульти-ограничения типа 3 – AA-CCBS+MC3: 2002 задания при $w = 1.01$ против 1110 при $w = 1$. Стоит дополнительно отметить тот факт, что для алгоритмов, использующих технику взаимоисключающего разделения, AA-CCBS+DS и AA-CCBS+DS+MC3, такого эффекта не наблюдается. Число решаемых этими алгоритмами заданий не повышается существенно с ростом w, что говорит о том, что взаимоисключающее разделение гораздо хуже подходит для сфокусированного поиска по сравнению с наложением мульти-ограничений. В целом, можно сделать вывод о том, что предлагаемый в работе способ наложения мульти-ограничений существенно повышает вычислительную эффективность фокусированного конфликтно-ориентированого планирования при поиске ограниченно-субоптимальных решений.

Отдельный интерес представляет замер фактического превышения стоимости (по сравнению с оптимальным решением) при использовании фокусировки поиска – см. Рис. 3.17.

На этом рисунке изображены диаграммы размаха: небольшими кружками показаны выбросы, заполненная цветом область – область от 1-го до 3-го квартиля (т.е. большинство заданий имеют такую стоимость решения), засечками – минимум и максимум (за исключением выбросов). По горизонтальной оси отложены различные значения параметра w (фактора субоптимальности). Различными цветами отмечены различные модификации алгоритма AA-CCBS.

Анализируя этот график можно сделать следующие выводы. Во-первых, очевидно, что даже при $w = 1.25$, то есть когда допускается превышение стоимости оптимального решения на 25%, фактическое превышение стоимости во многих случаях укладывается в 5%. Во-вторых, использование мульти-огра-

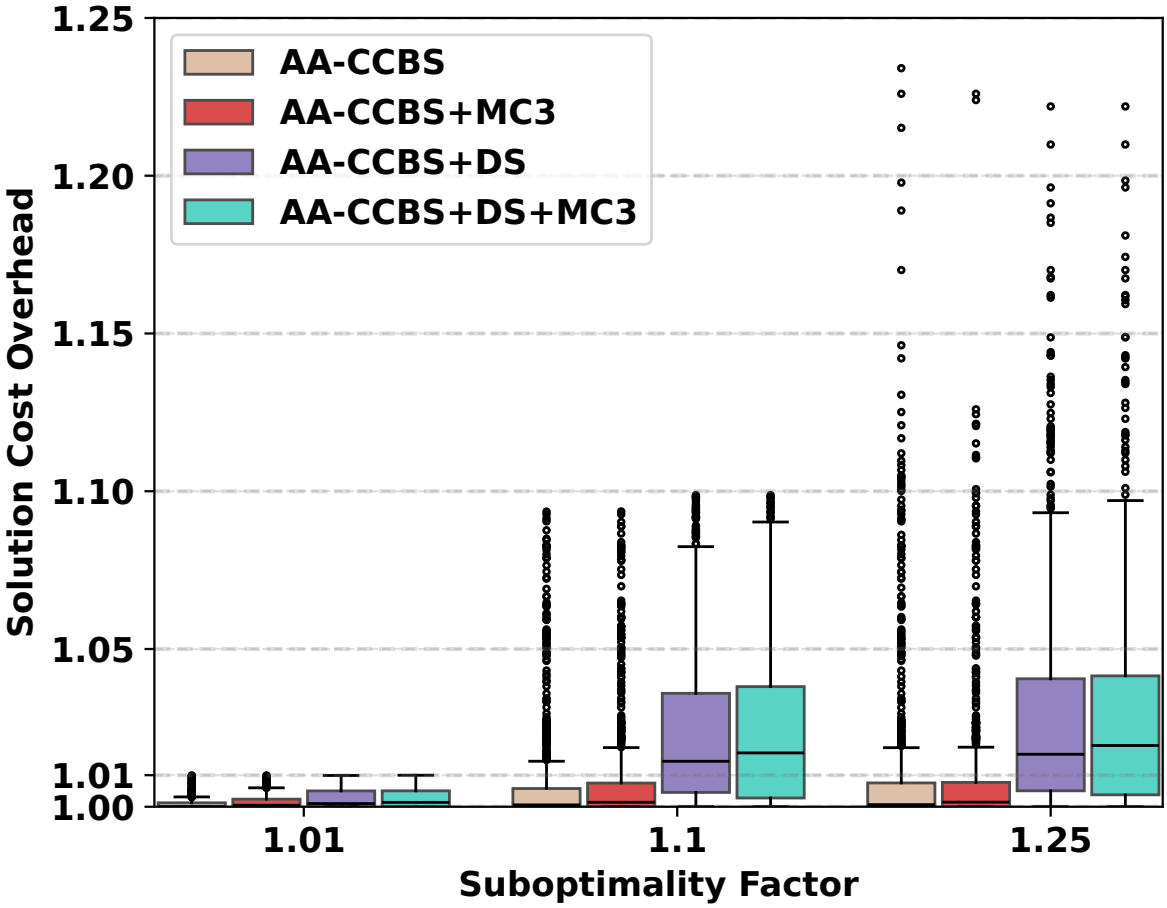


Рисунок 3.17 — Стоимость решений, отыскиваемых различными модификациями алгоритма `AA-CCBS` при фокусировке поиска.

ничений предпочтительней с точки зрения стоимости отыскиваемых решений: фактическая стоимость решений, отыскиваемых `AA-CCBS+MC3` ниже, чем у `AA-CCBS+DS`, `AA-CCBS+DS+MC3`.

Обобщая вышесказанное, можно утверждать, что предложенный в работе новый способ формирования мульти-ограничений существенно повышает эффективность конфликтно-ориентированного планирования на практике при поиске как оптимальных решений задачи AA-MAPF, так и ограниченно-субоптимальных.

3.3.2 Экспериментальные исследования методов решения задачи AA-MAPF на основе приоритизированного планирования

Предлагаемый в работе алгоритм приоритизированного планирования, `AA-SIPP(m)`, был реализован на языке C++ и проведено его поэтапное экспериментальное исследование. Сначала исследовались, какое влияние оказывают предлагаемые в работе техники переназначения приоритетов и фиксации безопасных интервалов начальных вершин на эффективность `AA-SIPP(m)`. Для этого были созданы авторские задания, характеризующиеся повышенной плотностью агентов. Затем с помощью алгоритма `AA-SIPP(m)` решались задания из раздела 3.3.1 для его сравнения с алгоритмами конфиктно-ориентированного поиска, гарантирующими отыскание оптимальных и ограниченно-субоптимальных решений. Наконец в завершающей серии экспериментов алгоритм `AA-SIPP(m)` сравнивался с алгоритмами решения классической задачи MAPF, т.е. задачи в которой не допускается перемещение агентов вне исходной топологии ГРД. Целью этих экспериментов было установить, насколько сокращается стоимость решения при переходе от задачи MAPF к задаче AA-MAPF, в которой агенты могут перемещаться между произвольными вершинами ГРД.

Первая серия экспериментов Целью этих экспериментов было установить, насколько предлагаемые в работе техники повышают эффективность приоритизированного планирования при решении различных задач. Для проведения этих экспериментов использовались два ГРД: empty-32-32 и warehouse-35-21, именуемые в дальнейшем просто как empty и warehouse. Будем ссылаться на эти ГРД как на карты.

Карта empty имеет размер 32×32 и не содержит непроходимых вершин (клеток). Число агентов для этой карты варьировалось от 8 до 320. Карта warehouse имеет размер 21×35. Она заимствована из работы [164], посвященной исследованию алгоритмов планирования траекторий в складских помещениях. Она содержит 10 продольных препятствий (размером 10×1) с широкими проходами между ними. Число агентов для этой карты варьировалось от 16 до 160. Пример задания с 160 агентами, приведен на Рис. 3.18. Как на карте empty, так на карте warehouse для каждого числа агентов было сгенерировано случай-

ным образом по 100 различных заданий (т.е. 100 различных пар начальных и конечных вершин).

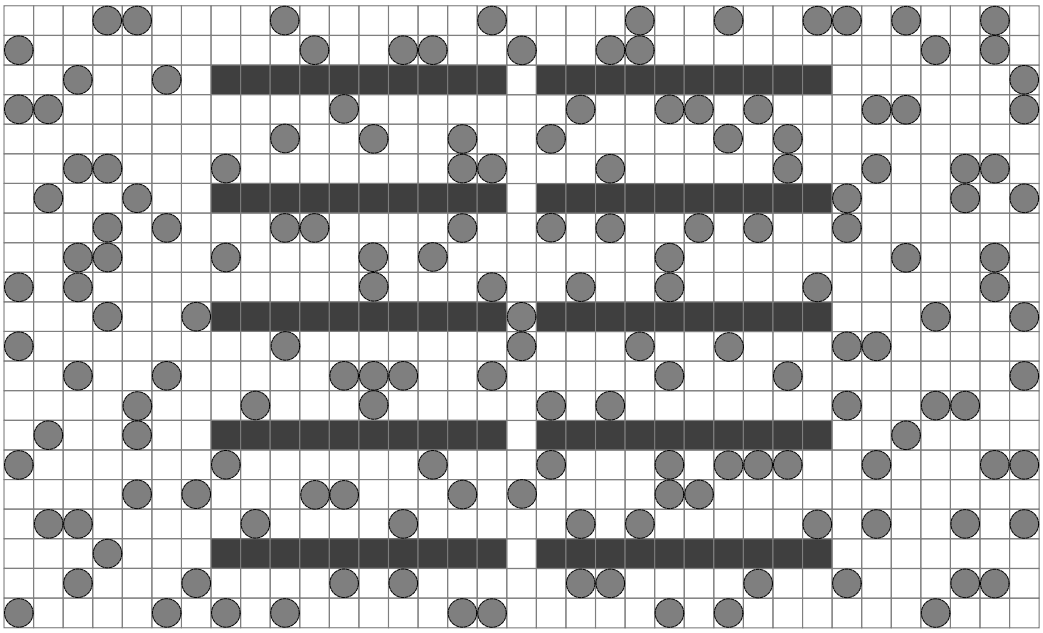


Рисунок 3.18 — Пример карты warehouse и задания, содержащего 160 агентов.

Сначала исследовалось влияние предложенной техники назначения безопасных интервалов в начальных вершинах (SSI). Для этого на карте empty для 8-320 проводились тесты с различной продолжительностью безопасных интервалов $[0,t_{ssi}]$, $t_{ssi} \in \{0, 1, 3, 5, 7, 9, \infty\}$.

На графике, изображенном на Рис. 3.19, показан процент успешно решенных заданий – SR (от англ. success rate), – для различного числа агентов при различной продолжительности безопасных интервалов.

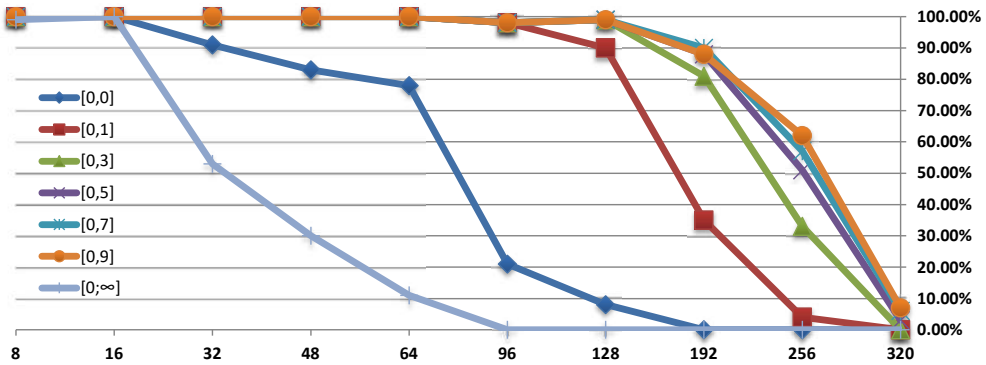


Рисунок 3.19 — Процент успешно решенных заданий для различной продолжительности безопасного интервала в начальных вершинах.

Как видно из вышеприведенного графика хуже всего себя показывает приоритизированное планирование с $t_{ssi} = \infty$, т.е. когда через начальные вершины агентов вообще не могут проходить пути других агентов. Процент успешно

решенных заданий заметно увеличивается, если разрешить высокоприортизированным агентам использовать начальные вершины низкоприоритизированных агентов при планировании. Однако, если не использовать безопасные интервалы в начальных вершинах (формально – использовать интервал [0,0]), то эффективность планирования заметно ниже, чем при их использовании. Согласно полученным результатам, процент успешно решенных заданий увеличивается (график сдвигается выше и правее) при увеличении t_{ssi} от 1 до 5. Однако дальнейшего повышения эффективности не наблюдается. То есть установление продолжительности безопасных интервалов в начальных вершинах больше определенного порога (в данном случае – больше 5) не имеет практического смысла. Это подтверждается и результатами, представленными в Табл. 3.2, в которой показано среднее время работы алгоритма – t, средняя стоимость отыскиваемых решений – $cost$ и процент успешно решенных заданий – SR для 192 агентов при различной продолжительности безопасных интервалов в начальных вершинах.

Таблица 3.2 — Результаты работы алгоритма `AA-SIPP(m)` при различной продолжительности безопасного интервала в начальных вершинах на карте emty (32×32) для 192 агентов.

	SR	t	$cost$
[0,1]	35%	1,15	3 741,77
[0,3]	81%	1,17	3 799,47
[0,5]	88%	1,07	3 835,09
[0,7]	90%	1,20	3 872,36
[0,9]	88%	1,22	3 907,59
[0,11]	87%	1,37	3 935,78
[0,13]	87%	1,40	3 975,10
[0,15]	87%	1,44	4 014,03
[0,17]	86%	1,48	4 054,71
[0,19]	88%	1,24	4 098,57
[0,21]	86%	1,27	4 140,51
[0,23]	87%	1,31	4 187,28

Как следует из таблицы, увеличение значения t_{ssi} больше определенного порога (больше 5 в данном случае) не оказывает значимого положительного влияния на процент успешно решенных задач. При этом, такое повышение

негативно сказывается на стоимости отыскиваемых решений. На время работы алгоритма значение t_{ssi} не оказывает значительного влияния.

В следующем эксперименте исследовалась предложенная в работе техника переназначения приоритетов (в случае когда решения с начальной приоритизацией не найдено). Для этого эксперимента использовалась карта warehouse, а значение t_{ssi} равнялось 5.

Алгоритм, использующий предлагаемую технику, `AA-SIPP(m)`, сравнивался с аналогичным алгоритмом, в котором процедура детерминированного переназначения приоритетов заменена на случайную ре-приоритизацию. Для обозначения этой модификации алгоритма используется запись `AA-SIPP(m)-RR` (от англ. random re-start). Поскольку время работы `AA-SIPP(m)-RR` может быть чрезмерно велико (из-за того, что число возможных последовательностей приоритетов составляет $n!$, где n – число агентов), время работа этого алгоритма было ограничено 5 минутами. Если за это время алгоритм не находил решение (но продолжал работать), то он прерывался, и этот запуск считался неудачным. Для того, чтобы результаты `AA-SIPP(m)-RR` были более достоверны (т.к. алгоритм опирается на случайный выбор), проводилось 100 запусков алгоритма на одном и том же задании (и его результирующие показатели усреднялись).

Результаты представлены в Табл. 3.3, в которой, как и ранее, показаны процент успешно решенных заданий – SR, среднее время работы алгоритма – t, средняя стоимость отыскиваемых решений – $cost$ для различного числа агентов. В столбцах с пометкой $tries$ показано среднее число переназначений приоритетов.

Таблица 3.3 — Результаты работы алгоритмов `AA-SIPP(m)`, `AA-SIPP(m)-RR` для различного числа агентов на карте warehouse (21×35).

	`AA-SIPP(m)-RR`				`AA-SIPP(m)`			
	tries	*SR*	*t*	*cost*	*tries*	*SR*	*t*	*cost*
16	1.00	1.00	0.01	285.15	1.00	1.00	0.01	285.15
32	1.00	1.00	0.04	591.30	1.00	1.00	0.04	591.30
64	1.22	1.00	0.22	1 274.94	1.16	1.00	0.17	1 252.82
96	1.44	1.00	0.68	2 212.14	1.35	1.00	0.49	2 149.31
128	7.34	1.00	5.73	3 543.92	2.51	1.00	1.40	3 322.52
160	140.96	0.42	117.82	5 220.90	28.21	0.99	28.28	5 298.85

Как следует из представленных результатов, предлагаемая в работе техника детерминированного переназначения приоритетов существенно превосходит случайное переназначение – время работы алгоритма снижается и процент успешно решенных заданий становится выше. Так для наибольшего числа агентов, 160, `AA-SIPP(m)` решает 99% заданий, а среднее время работы составляет 28 секунд (при этом, если агентов меньше 100, то время работы меньше 0.5 секунд), в то время как аналог способен решить лишь 42% заданий со средним времени работы в 4.5 раза выше.

Итак, можно сделать однозначный вывод о том, что предлагаемые в работе техники существенно повышают эффективность приоритизированного планирования. При этом алгоритм `AA-SIPP(m)`, их использующий, способен решать сложные задания с высоким числом агентов (порядка сотен агентов) менее, чем за 1 секунду.

Вторая серия экспериментов Целью этих экспериментов было сравнение предлагаемого в работе алгоритма `AA-SIPP(m)`, предназначенного для решения задачи AA-MAPF, с известными алгоритмами решения задачи MAPF (т.е. поиска совокупности путей на 4-связном ГРД, когда не допускается перемещение между произвольными вершинами ГРД). В частности исследовались алгоритмы:

- `ICBS` [180] – модификация известного алгоритма `CBS` [48], включающая в себя ряд улучшений, наплавленных на повышение эффективности поиска.
- `ECBS` [185] – модификация `CBS`, использующая фокусировку на нижнем и верхнем уровне конфликтно-ориентироавнного поиска.
- `SIPP(m)` – модификация предлагаемого в работе алгоритма `AA-SIPP(m)`, которая не допускает перемещение между произвольными вершинами ГРД.

Первые два алгоритма опираются на принцип конфликтно-ориентированного планирования и предназначены для поиска оптимальных (`ICBS`) и ограниченно субоптимальных решений (`ECBS`) на 4-связном ГРД. Для `ECBS` фактор субоптимальности был выбран как $w = 1.1$, т.е. фокусировка поиска производилась так, чтобы гарантировать, что стоимость отыскиваемого решения не превысит стоимости оптимального решения, более чем на 10%. Алгоритм

SIPP(m) является вариантом предлагаемого в работе приоритизированного алгоритма AA-SIPP(m), в котором для поиска индивидуального пути применяется базовый метод SIPP, а не предлагаемый в работе AA-SIPP.

Для проведения экспериментов использовались 4 карты различного размера и топологии:

- empty (64×64) – пустая карта.
- brc202 (530×481) – карта из известной в сообществе коллекции MovingAI [219], содержащая множество коридоров, которые соединяют различные помещения, ограниченного размера.
- den520d (256×257) – карта из известной в сообществе коллекции MovingAI [219], содержащая множество открытых пространств и почти не содержащая коридоров.
- ost003d (194×194) – карта из известной в сообществе коллекции MovingAI [219], содержащая как открытые пространства, так и коридоры между ними.

Число агентов для карты empty варьировалось от 50 до 250 (с шагом 50). Для каждого числа агентов было создано 100 различных заданий, различающихся расстановкой начальных и конечных вершин. Задания генерировались случайно, но так, чтобы расстояние между любыми вершинами старта-финиша было больше либо равно $4 \cdot r$, где r – это радиус безопасности каждого из агентов ($r = 0.5$ в данных экспериментах). Такой способ генерации гарантирует, что любое из заданий является решаемым для приоритизированных планировщиков в случае, когда $t_{ssi} = \infty$, т.к. любой низко-приоритизированный агент может сколь угодно долго ждать в своей начальной вершине, а любой высоко-приоритизированный агент может найти путь до своей целевой вершины даже если все остальные агенты зафиксированы в своих начальных (целевых) вершинах. Такой тип расстановки начальных и целевых вершин известен в литературе как правильно-сформированная инфраструктура (англ. well-formed infrastructure) см. [193] и часто используется при экспериментальном исследовании приоритизированных алгоритмов планирования, поэтому он и был включен в этот эксперимент.

Для карт brc202, den520d, ost003d число агентов варьировалось от 25 до 100 (с шагом 25). Для каждого числа агентов было сгенерировано 100 различных заданий, различающихся расстановкой начальных и конечных вершин. Задания генерировались случайно следующим образом. Сначала для очеред-

Таблица 3.4 — Результаты работы алгоритмов ICBS, SIPP(m), AA-SIPP(m), ECBS на карте empty (64 × 64) для 50-250 агентов.

n		ICBS	SIPP(m)	AA-SIPP(m)	ECBS
50	SR	97%	100%	100%	100%
	t(s)	3.42	0.01	0.13	0.14
	cost	2 240.56	2 249.31 (+0.39%)	1 758.29 (-21.52%)	2 241.76 (+0.05%)
100	SR	62%	100%	100%	100%
	t(s)	26.98	0.06	0.42	0.59
	Cost	4446.65	4 483.66 (+0.83%)	3 575.87 (-19.58%)	4 452.28 (+0.13%)
150	SR	14%	100%	100%	100%
	t(s)	–	0.12	0.89	1.70
	Cost	–	6 749.81	5 485.56	6 668.04
200	SR	0%	100%	100%	100%
	t(s)	–	0.21	1.68	4.12
	cost	–	9 106.20	7 574.80	8 947.80
250	SR	0%	100%	100%	100%
	t(s)	–	0.33	2.88	9.01
	cost	–	11 532.85	9 812.70	10 778.60

ного агента случайно выбиралась начальная вершина. Затем выполнялось случайное блуждание из этой вершины длиной в 100 000 шагов для фиксации целевой вершины.

Время, отводимое каждому алгоритму на решение задания, было ограничено, т.к. в противном случае время работы алгоритмов, опирающихся на конфликтно-ориентированное планирование (в частности – ICBS), было очень долгим. Временной лимит в этих экспериментах составлял 5 минут (схожий временной лимит обычно используется в литературе).

Результаты экспериментов на карте empty (пустой карте) представлены в Таблице 3.4. Как и ранее в таблице указан процент успешно решенных заданий – SR, среднее время, затраченное алгоритмом на отыскание решения – t (в секундах) и средняя стоимость полученных решений – *cost*.

Как видно из таблицы алгоритм ICBS является наименее эффективным в вычислительном смысле. Из-за высокого времени работы он не способен ре-

шать (за отведенный временной лимит – 5 минут) задания с числом агентов, превышающим 150. В то время как остальне алгоритмы, в т.ч. предлагаемый в работе алгоритм AA-SIPP(m) нашли решения всех заданий. По времени работы AA-SIPP(m) уступает своей упрощенной версии SIPP(m), это не удивительно, т.к. планирование с учетом возможности перемещения между произвольными вершинами более трудозатратно. Вместе с этим AA-SIPP(m) превосходит по времени работы алгоритм ECBS. Здесь уместно отметить, что прямое сравнение времен работы этих алгоритмов не вполне некорректно из-за различий в реализации. Тем не менее, вполне корректно сравнить темпы роста времени работы алгоритмов в зависимости от размерности входа (числа агентов n). Здесь следует отметить, что с ростом числа агентов время работы AA-SIPP(m) увеличивается не так значительно, как время работы ECBS. Так при увеличении числа агентов с $n = 100$ до $n = 100$ время работы ECBS увеличивается в 7 раз (с 0.59 до 4.12 секунд), а AA-SIPP(m) – в 4 раза (с 0.42 до 1.68 секунд).

Одно из самых важных преимуществ AA-SIPP(m) – ожидаемое снижение стоимости решения. Как видно из таблицы стоимость решений, отыскиваемых предлагаемым в работе алгоритмом существенно ниже, чем у аналогов. Как видно из таблицы для 50-100 агентов решения, отыскиваемые AA-SIPP(m) на 20% ниже по стоимости, чем решения ICBS. При этом ICBS – это алгоритм, гарантирующий отыскание оптимальных решений на 4-связном ГРД. То есть даже при субоптимальной приоритизированной схеме решения задачи AA-MAPF, получаемые решения на 20% лучше (с точки зрения стоимости), чем оптимальные решения задачи MAPF.

Рассмотрим далее результаты, полученные на картах brc202, den520d, ost003d. В частности, на Рис. 3.20 показан график зависимости процента успешно решенных заданий, SR, от числа агентов. Все алгоритмы, кроме ICBS решили 100% предъявленных заданий.

Время работы алгоритмов показано в Табл. 3.5.

Таблица 3.5 — Среднее время работы алгоритмов ICBS, ECBS, SIPP(m), AA-SIPP(m) в зависимости от числа агентов на картах brc202, den520d, ost003d.

	brc				den				ost			
	$n = 25$	$n = 50$	$n = 75$	$n = 100$	$n = 25$	$n = 50$	$n = 75$	$n = 100$	$n = 25$	$n = 50$	$n = 75$	$n = 100$
ICBS	0.11	2.06	5.29	13.11	0.35	4.93	15.05	32.64	0.64	10.78	–	–
ECBS	0.21	0.48	0.92	1.61	0.29	0.83	1.53	1.90	0.18	0.51	1.40	2.57
SIPP(m)	0.09	0.19	0.31	0.47	0.11	0.23	0.39	0.57	0.10	0.22	0.42	0.59
AA-SIPP(m)	0.47	1.14	2.01	3.20	0.69	1.71	3.23	4.91	0.48	1.30	2.79	4.23

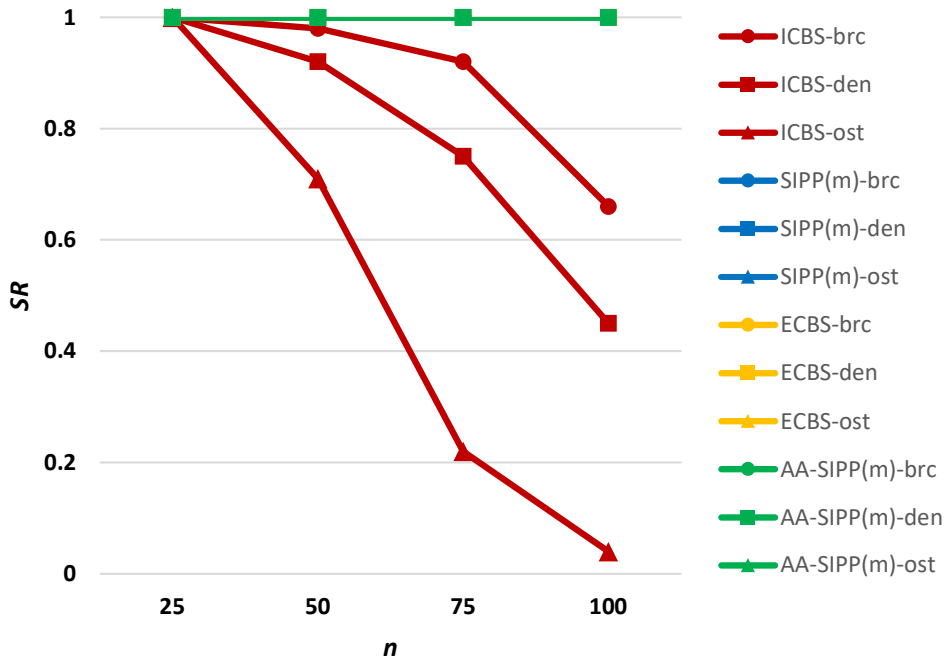


Рисунок 3.20 — Процент успешно решенных заданий для алгоритмов `ICBS`, `ECBS`, `SIPP(m)`, `AA-SIPP(m)` в зависимости от числа агентов на картах brc202, den520d, ost003d.

Результаты, представленные в таблице для карт brc202, den520d, ost003d схожи с ранее полученными результатами для карты empty: алгоритм `ICBS` демонстрирует наихудшее время работы, а `SIPP(m)` – наилучшее. Предлагаемый в работе алгоритм `AA-SIPP(m)`, как и ранее, уступает по времени работы `SIPP(m)` и незначительно уступает `ECBS`, хотя ранее `ECBS` проигрывал `AA-SIPP(m)` по этому показателю. Это можно объяснить тем фактом, что на картах большого размера, содержащих препятствия, накладные расходы на поиск путей между произвольными вершинами графа заметно возрастают по сравнению с аналогичным поиском на пустой карте небольшого размера. В целом, эффективность работы предлагаемого алгоритма `AA-SIPP(m)` можно считать достаточно высокой – он способен отыскивать неконфликтные пути для многих десятков (и даже сотен) агентов на карте большого размера за приемлемое для практических применений время.

Как и ранее, основное преимущество алгоритма `AA-SIPP(m)` – снижение стоимости отыскиваемых решений. На Рис. 3.21 приведена диаграмма нормализованной средней стоимости решения. Эта диаграмма строилась следующим образом: для каждого задания, которое было решено всеми 4 исследуемыми алгоритмами, стоимость решения делилась на стоимость решения `ICBS` (т.к. это наименьшая стоимость решения задачи MAPF на 4-х связном ГРД).

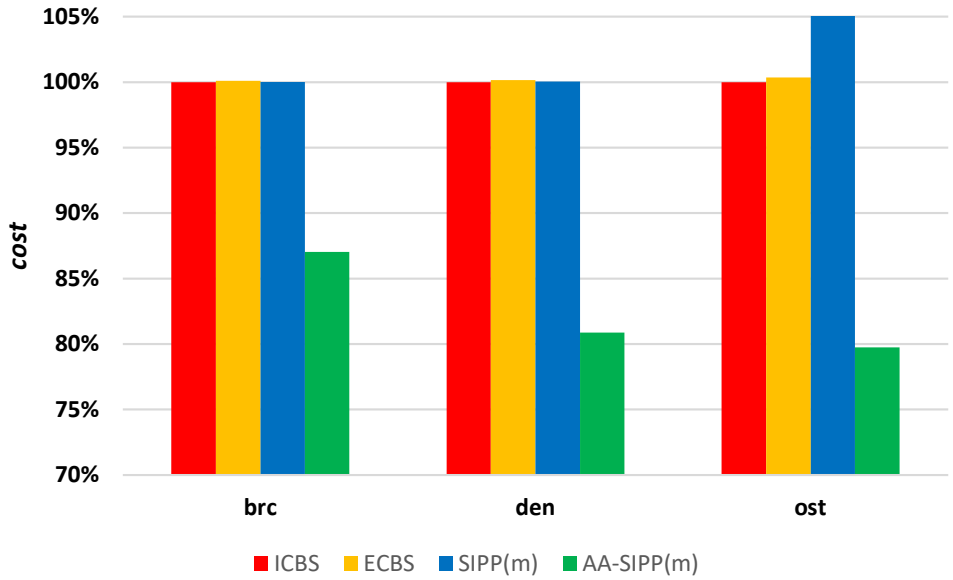


Рисунок 3.21 — Нормализованная стоимость решений задачи MAPF/AA-MAPF, отыскиваемых алгоритмами `ICBS`, `ECBS`, `SIPP(m)`, `AA-SIPP(m)` на картах brc202, den520d, ost003d.

Как видно на диаграмме, стоимость решений, отыскиваемых алгоритмами `ICBS`, `ECBS` и `SIPP(m)` находится на одном уровне, в то время как аналогичный показатель алгоритма `AA-SIPP(m)` ниже на 13-20%. То есть, как и ранее, стоимость отыскиваемых им решений заметно ниже аналогов, что подтверждает преимущество разработанного алгоритма в решении практических задач.

Дополнительно были проведены эксперименты, направленные на сравнение алгоритма `AA-SIPP(m)` с алгоритмом `AA-CCBS`, гарантирующим отыскание оптимальных решений задачи AA-MAPF и его модификацией `Focal-AA-CCBS`, гарантирующей отыскание ограниченно-субоптимальных решений. Исследовались наиболее эффективные модификации последних двух алгоритмов, в частности в `AA-CCBS` использовались мульти-ограничения типа 3 и техника взаимоисключающего разделения, алгоритм `Focal-AA-CCBS` использовал мульти-ограничения типа 3, при этом фактор субоптимальности был задан как $w = 1.25$.

Методология эксперимента совпадает с методологий описанной в разделе 3.3.1: использовались карты и задания из коллекции MovingAI [219], в частности карты empty, random, maze, den312 (внешний вид карт показан на Рис. 3.14). Ограничение на время работы алгоритма составляло 5 минут. Результаты эксперимента представлены в Табл. 3.6, 3.7.

Таблица 3.6 — Число успешно решенных заданий алгоритмами AA-CCBS, Focal-AA-CCBS, AA-SIPP(m) на картах empty, random, maze, den312

	AA-CCBS	Focal-AA-CCBS	AA-SIPP(m)
empty	267	455	3 640
random	340	623	1 922
maze	120	199	1 968
den312	189	366	7 512
Total	916	1 643	15 042

Таблица 3.7 — Нормализованная стоимость решений, отыскиваемых алгоритмами AA-CCBS, Focal-AA-CCBS, AA-SIPP(m) на картах empty, random, maze, den312.

	AA-CCBS	Focal-AA-CCBS	AA-SIPP(m)
empty	100.0%	101.4%	104.3%
random	100.0%	101.1%	104.4%
maze	100.0%	100.4%	104.2%
den312	100.0%	100.2%	102.3%
AVG	100.0%	100.8%	103.8%

В Табл. 3.6 показано сколько заданий успел решить тот или иной алгоритм с учетом отведенного временного лимита (5 минут на решение одного задания). Очевидно, что алгоритм AA-SIPP(m), опирающийся на приоритизированное планирование, существенно превосходит алгоритмы AA-CCBS и Focal-AA-CCBS. Число успешно решенных им заданий примерно в 10 раз превосходит аналогичный показатель Focal-AA-CCBS (и в 16 – AA-CCBS). Это является наглядным подтверждением достаточно очевидного факта, что приоритизированное планирование существенно эффективней, чем конфликтно-ориентированное планирование при решении практических задач. При этом, как следует из Табл. 3.7 разница в стоимости отыскиваемых решений не является существенной. Так, решения, отыскиваемые алгоритмом AA-SIPP(m) в среднем лишь на 4% превышают по стоимости оптимальные решения.

Третья серия экспериментов Целью этой серии экспериментов была апробация предложенных в работе методов решения задачи AA-MAPF на реальных робототехнических системах. Экспериментальное исследование проводилось на базе Национального исследовательского центра “Курчатовский институт”

(лаборатория робототехники Курчатовского центра НБИКС-технологий). Для эксперимента использовались малые колесные роботы с дифференциальным приводом – YARP-2. Каждый робот имеет размер 21 × 21 см. и способен перемещаться со скоростью 0.1 м/с. На каждом роботе сверху установлен цветной маркер, с помощью которого положение робота на полигоне отслеживается в реальном времени видео-камерами, закрепленными под потолком по периметру полигона. Полигон имеет размер 4.8 × 6 м., и для планирования он моделировался с помощью ГРД размера 120 × 96 (т.е. расстояние между ортогонально-смежными вершинами составляло 5 см.). Внешний вид роботов и полигона представлен на Рис. 3.22.

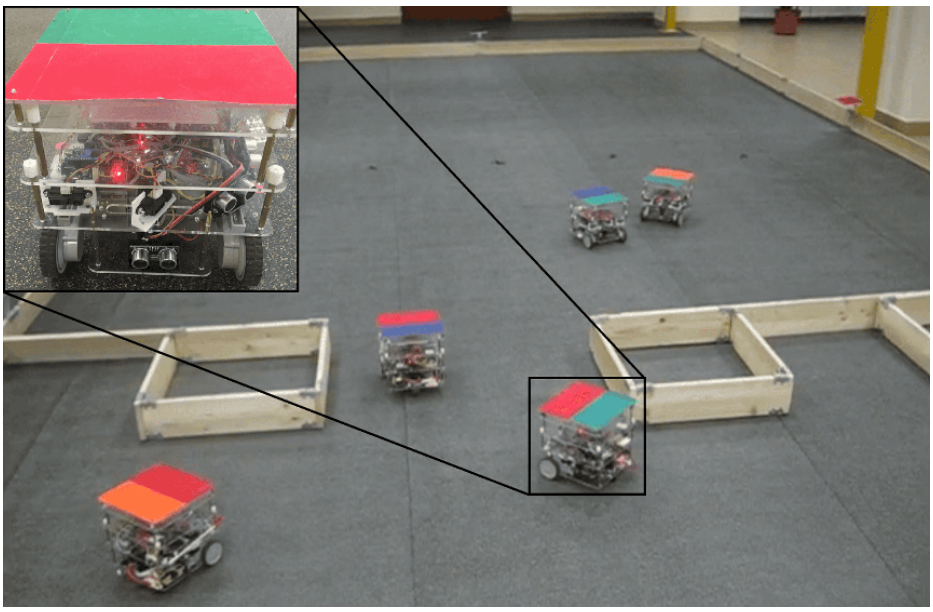


Рисунок 3.22 — Роботы и полигон, используемые для проведения экспериментальных исследований разрабатываемых методов решения задачи AA-MAPF.

Для проведения экспериментов использовалась распределенная система управления. Во-первых, использовался центральный управляющий сервер (персональный компьютер), под управлением операционной системы Ubuntu, с установленной библиотекой ROS (Robotic Operating System). Специалистами лаборатории робототехники НИЦ “Курчатовский институт” были реализованы следующие управляющие модули этого сервера:

– модуль получения и обработки видео-потока с камер, отслеживающих состояние роботов;
– модуль локализации (вычисления положения) роботов на полигоне;
– модуль коммуникации с роботами.

С помощью последнего модуля осуществлялась передача роботу очередного действия для исполнения согласно построенному (с помощью предлагаемых в работе алгоритмов решения задачи AA-MAPF) плану. Действие исполнялось с помощью локального контроллера робота, который был реализован на основе пропорционально-интегрально-дифференцирующего регулятора (ПИД-регулятора).

В качестве глобального планировщика, ответственного за построение совокупности неконфликтных путей использовался алгоритм `AA-SIPP(m)`, показавший свою эффективность по результатам предыдущих экспериментов. Алгоритм был модифицирован так, чтобы учитывать ориентацию робота и время, затрачиваемое на поворот на месте для смены направления движения[9].

Перед началом экспериментов с группой роботов был проведен эксперимент на одиночном роботе, который должен был проследовать по заранее построенной алгоритмом траектории. Целью этого эксперимента было установить насколько робот в процессе исполнения отклоняется от заданной траектории. Было проведено более 15 запусков робота по траектории. На Рис. 3.23 показано то насколько реальной положение робота отличается от запланированного в каждом из них.

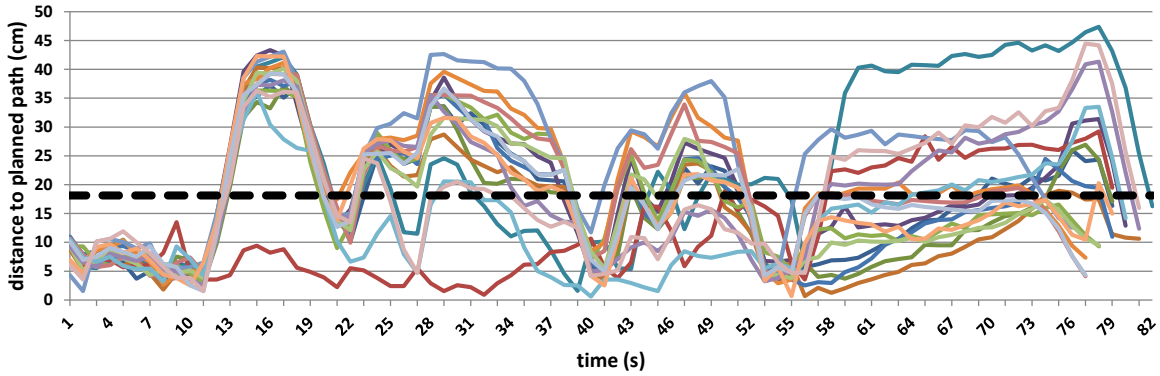


Рисунок 3.23 — Ошибка следования вдоль запланированного пути при проведении экспериментов на реальном роботе.

Как видно из вышеприведенного рисунка ошибка следования вдоль траектории может быть достаточно велика и достигать 45 см. Т.е. превышать линейные размеры робота (21 × 21 см.) чуть более, чем в 2 раза. При этом

[9]Модификация алгоритма носит технический характер, подробности изложены в [232]

средняя ошибка (показана жирной пунктирной линией на графике) составляет 18 см. (т.е. примерно равняется размеру робота)[10].

Для компенсации этой ошибки было предложено использовать ряд техник. Во-первых, при планировании перед каждым действием перемещения добавлялась искусственная задержка продолжительностью 5 секунд (значение подбиралось экспериментально). При следовании же по траектории этот временной интервал использовался для компенсации задержки достижения роботом промежуточных точек маршрута. Таким образом происходил своеобразный "сброс ошибки" после каждого действия перемещения и негативный эффект от накопления ошибки существенно снижался. Во-вторых, при расчете безопасных интервалов для осуществления поиска индивидуальных путей робот моделировался кругом расширенного радиуса. Так робот YARP-2 при повороте на месте описывает окружность радиусом примерно 15 см., но радиус безопасности, используемый, при планировании мог увеличиваться до 25 и 30 см. с тем, чтобы даже при отклонении от траектории реальный робот не покидал своего виртуального радиуса безопасности.

Далее проводились эксперименты на 6 роботах, движущихся по полигону одновременно. Препятствия на полигоне были сконструированы вручную так, как показано на Рис. 3.24 (на рисунке одна клетка изображенного ГРД для наглядности соответствует 9 клеткам ГРД, использовавшегося в экспериментах.)

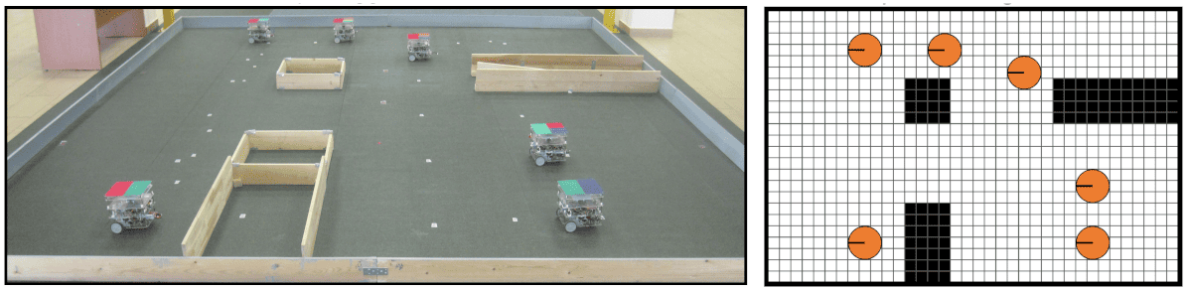


Рисунок 3.24 — Внешний вид полигона во время проведения экспериментов с группировкой из 6 роботов – слева. Справа – соответствующий ГРД.

Было сгенерировано вручную 10 различных заданий для группы из 6 роботов. Для каждого задания с помощью алгоритма `AA-SIPP(m)` строилось 6

[10]Здесь следует отметить, что на графике показано отклонение между запланированными позициями робота и позициями, вычисленными с помощью имеющейся системы технического зрения. Эти вычисления, разумеется, тоже подвержены ошибкам, которые в данной работе не оценивались. Именно наличие этих неточностей объясняет тот факт, что в самом начале траектории уже наблюдается отклонение между запланированной и рассчитанной позицией робота в 5-10 см.

различных решений: с дополнительной задержкой между действиями перемещения и без; с разными значениями радиуса безопасности, учитываемого при планировании (15, 25 и 30 см.). Всего было осуществлено 60 запусков эксперимента: $10 \times 2 \times 3$. Процент успешно завершившихся запусков, т.е. таких запусков, когда роботы достигли своих целевых позиций без столкновений, показан в Табл. 3.8.

Таблица 3.8 — Процент успешно затершихся запусков в зависимости от радиуса безопасности (строки) и использования техники дополнительной задержки (столбцы).

	без задержки	**с задержкой**
15 см.	0%	10%
25 см.	40%	70%
35 см.	50%	100%

Как видно из таблицы, если планирование осуществлялось без использования расширенного радиуса безопасности и без дополнительной задержки, то при следовании по построенным траекториям всегда происходили столкновения (процент завершенных заданий равен 0). Это прямое следствие того, что в силу естественных ограничений роботы не могут следовать строго по спланированным траекториям. При этом процент успешно выполненных заданий заметно повышается при увеличении радиуса безопасности при построении и путей и использовании дополнительной задержки между действиями перемещения. Так при расширенном радиусе безопасности равном 30 см. и планировании с задержкой процент успешно завершившихся запусков равен 100%.

На Рис. 3.25 приведена диаграмма, показывающая насколько исполненные роботами траектории отличаются от спланированных по стоимости (суммарному времени следования).

Стоимость указана в нормализованном виде, за 100% взята стоимость решения задачи AA-MAPF, отыскиваемого алгоритмов `AA-SIPP(m)` без использования расширенного радиуса безопасности и добавления дополнительных задержек между действиями перемещения (т.е. это стоимость “идеальной” совокупности траекторий, проследовать по которым роботы без столкновений не могут, как было показано выше). Из диагаммы следует, что увеличение радиуса до 35 см. приводит к повышению стоимости решения примерно на 30%, а добавление задержек добавляет ещё около 45%, т.е. суммарное превышение

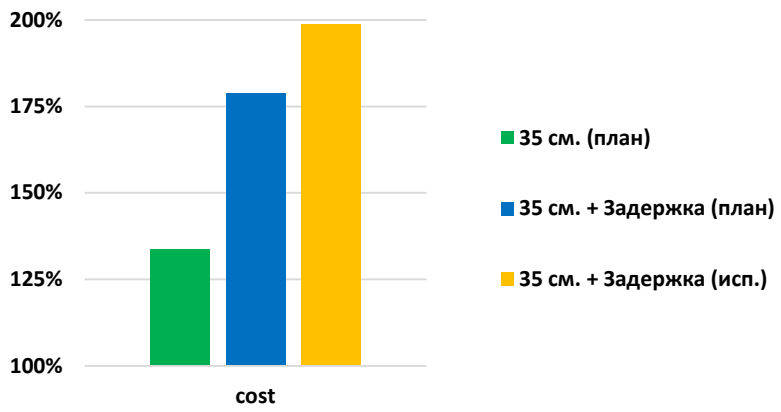


Рисунок 3.25 — Нормализованная стоимость решений задачи AA-MAPF в экспериментах на реальных роботах.

составляет чуть более 75%. Из-за неточностей же исполнения реальное суммарное время, затрачиваемое роботами на исполнение траекторий, увеличивается на 100%.

Обобщая результаты экспериментальных исследований на реальных роботах, можно сделать однозначный вывод о том, что предлагаемые в работе методы и алгоритмы решения задачи AA-MAPF, в частности алгоритм `AA-SIPP(m)`, могут успешно применяться для решения задач навигации коалиции колесных роботов с дифференциальным приводом и позволяют, в т.ч. за счет использования принципа безопасно-интервального планирования, компенсировать неточности, неизбежно возникающие при следовании по построенным траекториям на практике.

3.4 Выводы по главе

В данной главе работы рассматривалась задачи поиска совокупности неконфликтных путей на графе регулярной декомпозиции. При этом в отличие от других работ в данной области допускались перемещения между произвольными вершинами графа. Был предложен ряд новых, оптимальных и суб-оптимальных, алгоритмов решения указанной задачи. Так был предложен алгоритм `AA-CCBS` для поиска оптимальных решений рассматриваемой задачи. Этот алгоритм опирается на принцип конфликтно-ориентированного планирования и является двух-уровненвым. На верхнем уровне осуществляет-

ся управление конфликтами, в частности – поиск конфликта на множестве путей-кандидатов и наложение ограничений для его устранения, на нижем осуществляется поиск индивидуальных путей с учетом наложенным ограничений. В качестве алгоритма индивидуального поиска используется предложенный в работе ранее метод `TO-AA-SIPP`. Для повышения эффективности поиска на верхнем уровне предлагается ряд оригинальных подходов к наложению ограничений, направленных на агрегацию нескольких отдельных ограничений в одно мульти-ограничение, которое может быть устранено путем единственного вызова планировщика нижнего уровня, что, в свою очередь, положительно сказывается на общей производительности алгоритма. Для поиска суб-оптимальных решений рассматриваемой задачи предлагается двух-уровненвый планировщик на основе приватизированного планирования, названный `AA-SIPP(m)`, т.к. в качестве алгоритма поиска пути на нижнем уровне используется ранее описанный в работе алгоритм `AA-SIPP`. Для повышения вычислительной эффективности `AA-SIPP(m)` разработан ряд оригинальных техник (безопасные интервалы в начальных вершинах, динамическое изменение приоритизации). Предложенные алгоритмы экспериментально исследованы с использованием публично-доступных задач из известной в сообществе коллекции MovingAI. Результаты экспериментов демонстрируют существенное превосходство разработанных алгоритмов над аналогами по скорости работы и, как следствие, способности решать предъявляемые задания за ограниченный временной бюджет (что весьма важно с практической точки зрения). Дополнительно проведены экспериментальные исследования на реальных колесных роботах, показывающие применимость и эффективность разработанных методов на практике.

Глава 4. Поиск путей с геометрическими ограничениями на графах регулярной декомпозиции

В главе 1 описывалась связь задачи поиска пути на графе регулярной декомпозиции, вложенном в метрическое пространство, с задачей планирования траектории для мобильного робота. На практике многие мобильные роботы обладают так называемыми кинематическими ограничениями, то есть не могу совершать определенные типы движений из-за особенностей конструкции. Например, роботы автомобильного типа, обладающие так называемой велосипедной моделью движения, не могут поворачиваться на месте и совершать резкие повороты в движении. Если не учитывать этот факт при построении путей на графе регулярной декомпозиции, который моделируют окружающую робота среду, то эти пути окажутся не исполнимыми для робота, т.е. его система управления не сможет обеспечить достаточно точное следование вдоль траектории, что может привести к столкновению – см. Рис. 4.1.

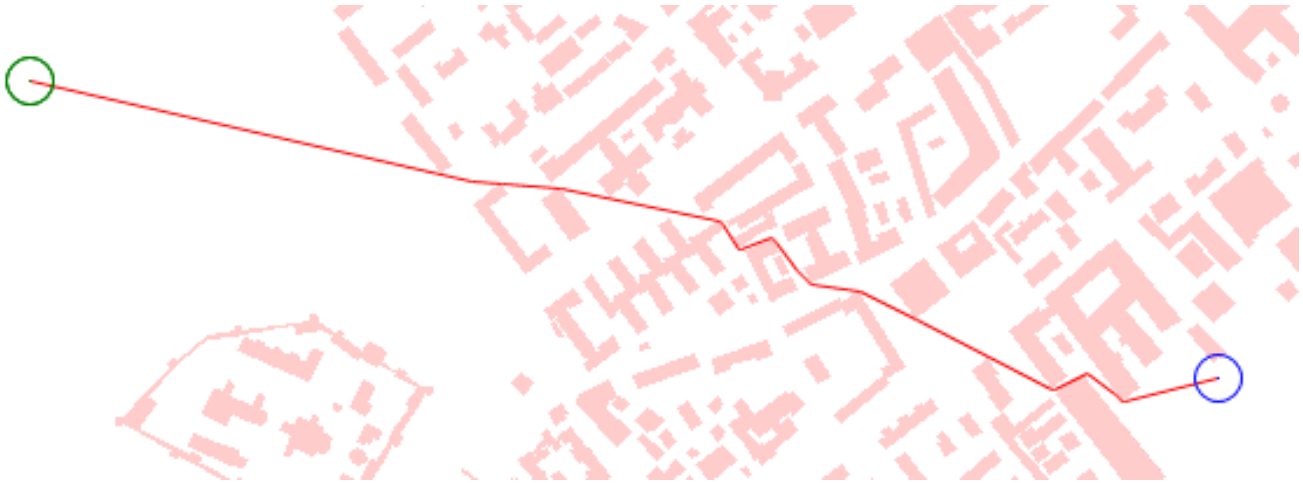


Рисунок 4.1 — Пример траектории, следование по которой опасно для объекта управления из-за наличия резких поворотов вблизи препятствий.

В связи с этим возникает задача построения таких путей на ГРД, что следование по ним может быть обеспечено системой автоматического управления робота с допустимой точностью. Одним из способов решения данной задачи является поиск путей на ГРД, обладающих геометрическими ограничениями, в частности ограничениями на то, как резко изменяется направление движения между смежными сегментами пути, образованными парами (удаленных друг

от друга) вершин ГРД. Именно этой задаче и эффективным методам её решения и посвящена данная глава. В ней будет формально представлена задача поиска пути с ограничением на угол отклонения между секциями, предложены эффективные методы её решения, проведено их теоретическое и эмпирическое исследование.

4.1 Постановка задачи (задача AC-PF)

Как и ранее в работе, рассмотрим конечный 4(8)-связный взвешенный ГРД $\mathcal{G} = (V, E, w)$, вложенный в метрическое пространство $\mathcal{M} = \mathbb{R}^2$ так, как это описано в Главе 1. То есть каждой вершине ГРД соответствует точка на плоскости. С каждым ребром графа $e = (u, v)$ будем ассоциировать отрезок AB, т.ч. $coord(u) = A$, $coord(v) = B$, где $coord : V \to \mathbb{R}^2$ – функция, сопоставляющая вершины графа точкам в $\mathbb{R}^2$ (функция вложения графа в пространство). Определим вес ребра как $w(e) = \|AB\|$.

Без ограничения общности будем считать, что горизонтально-смежные и вертикально-смежные вершины ГРД отстоят друг от друга на расстоянии 1. Следовательно, вес ортогональных ребер ГРД равен 1, а диагональных ребер – $\sqrt{2}$.

Будем полагать, что на всех парах вершин ГРД определена функция:

$$los: \ V \times V \to \{true, false\},$$

которая определяет возможность перехода не только между смежными, но и между *произвольными* вершинами графа. По умолчанию будет считать, что если $(u, v) \in E$, то $los(u,v) = true$.

С точки зрения задачи планирования траектории, функция $los(u, v)$ определяет возможность перехода между точками рабочего пространства с учетом особенностей агента (например, его размера) и расположения статических препятствий. На практике функция $los(u, v)$ (от. англ. “line-of-sight” – линия видимости) возвращает $true$ тогда и только тогда, когда агент может совершить перемещение по прямой от вершины u до вершины v без столкновения со статическими препятствиями, расположенными в рабочем пространстве.

Будем считать, что функция w, определяющая веса ребёр, определена и на произвольных парах вершин: $w : V \times V \to \mathbb{R}^+$ и соответствует расстоянию между вершинами.

Упорядоченную пару произвольных вершин ГРД будем называть *секцией*. Для обозначения секции будем использовать (с некоторым нарушением нотации) такую же запись, как и для обозначения ребра ГРД: $e = (u,v)$. Вершину u будем называть начальной вершиной секции и обозначать как $source(e)$. Аналогично, вершину v будем называть конечной и обозначать как $target(e)$. Длиной секции будем называть её вес и обозначать как $len(e)$.

Определение 36. Секция $e = (u,v)$ является *допустимой*, если и только если $los(u,v) = true$.

Определение 37. Секции $e = (u, v)$ и $e' = (u',v')$ являются *смежными*, если $target(e) = source(e')$, т.е. $v = u'$.

Определение 38. Пусть имеются две последовательные смежные секции: $e = (u, v)$ и $e' = (v,v')$. *Углом отклонения* секции e' от e будем называть модуль угла между векторами, координаты которых равны $(v_x - u_x, v_y - u_y)$ и $(v'_x - v_x, v'_y - v_y)$.

Будем обозначать угол отклонения секции e от e' как $\alpha(e,e')$ – см. Рис. 4.2.

Если $\alpha(e,e') \neq 0$, то общую для секций $e = (u,v)$ и $e' = (v,v')$ вершину, т.е. v, будем называть вершиной поворота.

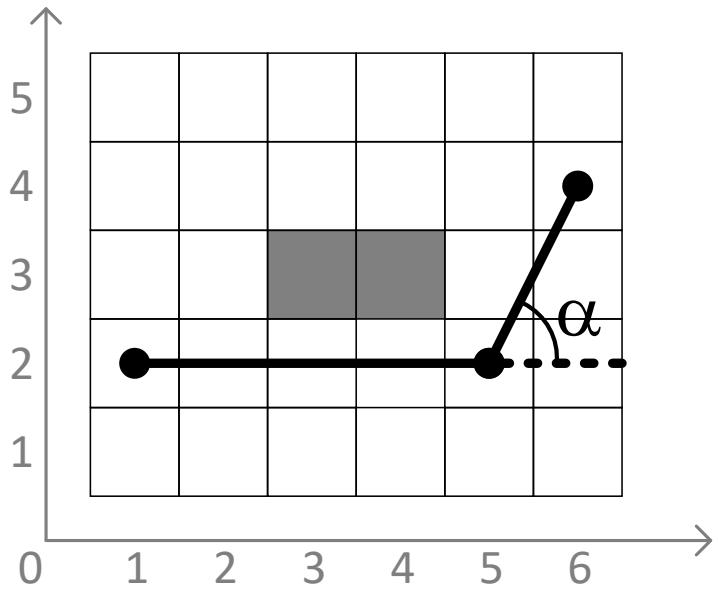


Рисунок 4.2 — Угол между смежными секциями на ГРД.

Определение 39. Путь на ГРД между вершинами v_{start} и v_{goal} – $\pi(v_{start},v_{goal})$, – это последовательность смежных допустимых секций, начинающаяся в v_{start} и заканчивающаяся в v_{goal}, т.е.:

$$\pi(v_{start}, v_{goal}) = (e_1, e_2, ..., e_n),$$
т.ч.:
$$source(e_1) = v_{start},$$
$$target(e_n) = v_{goal},$$
$$\forall i \in [1,n-1] : \ target(e_i) = source(e_{i+1})$$
$$\forall i \in [1,n] : \ los(source(e_i), target(e_i)) = true.$$

Определение 40. Стоимость (длина) пути $\pi = (e_1, ..., e_n)$ – это сумма длин секций его образующих:

$$cost(\pi) = \sum_{i=1}^{n} len(e_i)$$
$$= \sum_{i=1}^{n} w(source(e_i),target(e_i))$$

Введем в рассмотрение следующую характеристику пути, на которую будем ссылаться как на максимальный угол поворота (для этого пути).

Определение 41. Пусть имеется путь $\pi = (e_1, ..., e_n)$. Максимальный угол поворота для этого пути – есть величина определяемая следующим образом:

$$\alpha_m(\pi) = \max_{i=1..n}\{\alpha(e_1,e_2),...,\alpha(e_{n-1},e_n)\}$$

Интересующая нас задача построения пути с геометрическими ограничениями, а именно – ограничением на максимальный угол поворота, определяется следующим образом.

Определение 42. Задача поиска пути с ограничением на максимальный угол поворота – это набор:

$$\text{AC-PF} = (\mathcal{G},v_{start},v_{goal}, \alpha_{max},los), \tag{4.1}$$

где $\mathcal{G} = (V,E)$ – заданный ГРД, $v_{start} \in V$, $v_{goal} \in V$ – зафиксированные начальная и конечная вершины соответственно, $\alpha_{max} \in [0°, 180°]$ – заданное

ограничение на максимальный угол поворота, los – функция, определяющая возможность перехода между произвольными вершинами ГРД.

Аббревиатура AC-PF для обозначения введенной в рассмотрение задачи используется как сокращение следующего словосочетания на английском языке: **A**ngle **C**onstrained **P**ath **F**inding (дословно – поиск пути с ограничением на угол).

Определение 43. Решение задачи AC-PF – это набор смежных допустимых секций, образующих путь π (в соответствии с Определением 39) из v_{start} в v_{goal}, т.ч. $\alpha_m(\pi) \leqslant \alpha_{max}$.

На Рис. 4.3 изображены несколько решений одной и той же задачи AC-PF (начальная вершина помечена как s, а целевая как g). При этом, все три пути являются допустимыми решениями при $\alpha_{max} = 90°$. При $\alpha_{max} = 60°$, только пути, изображенные по центру и справа, являются решения задачи AC-PF. Наконец, если $\alpha_{max} = 45°$, то только правый путь является решением задачи AC-PF.

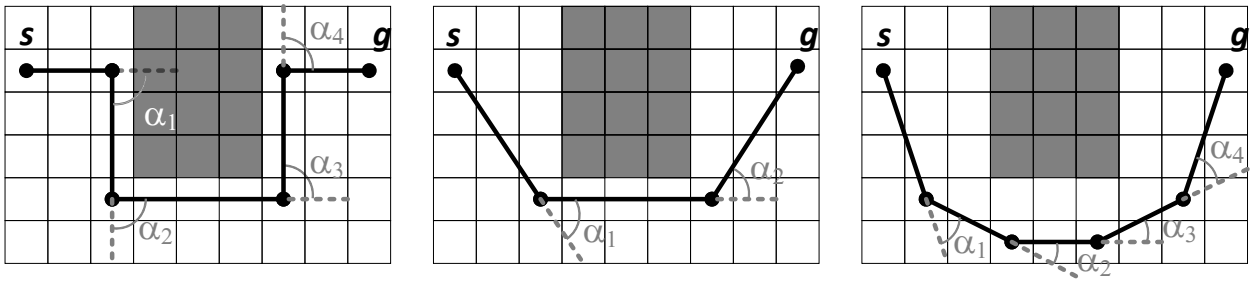


Рисунок 4.3 — Различные пути на ГРД, отличающиеся максимальным углом отклонения.

Интуитивно, чем меньше ограничение на максимальный угол поворота, тем плавнее должен быть путь. В практических задачах это ограничение задается пользователем исходя из особенностей мобильного агента (робота), для которого необходимо построить безопасный маршрут следования к целевой точке рабочей области. В работе [233], например, исследовалась задача построения пути и обеспечения следования вдоль него для малого беспилотного летательного аппарата мульти-роторного типа (коптера) и предлагалось использовать значения α_{max} в диапазоне [15°,30°]. В данной работе предполагается, что этот параметр предварительно зафиксирован и не может варьироваться в ходе поиска, т.е. любой путь π, не удовлетворяющий условию $\alpha_m(\pi) \leqslant \alpha_{max}$, не является допустимым (пусть даже разница между $\alpha_m(\pi)$ и α_{max} невелика).

4.2 Методы и алгоритмы

4.2.1 Известные методы и их ограничения

Как уже было упомянуто ранее в работе, в искусственном интеллекте известно большое число алгоритмов (эвристического) поиска пути на графах. Почти все из них являются модификациями известного эвристического алгоритма A* [11], который в свою очередь является модификацией алгоритма Дейкстры [8]. Некоторые из них предназначены для поиска исключительно на ГРД, такие как например JPS [15], Theta* [7], HPA* [16] и пр. Другие методы, например, R* [17], ARA* [43], являются более общими и могут быть адаптированы для поиска на графах, вложенных в метрическое пространство, в том числе на ГРД. Однако, при этом число алгоритмов, которые способны в том или ином виде учитывать форму пути при поиске на ГРД, весьма ограничено. Так, например, алгоритм A*-PS [16] после поиска пути производит его сглаживание, последовательно рассматривая последовательности вершин на предмет возможности исключения промежуточных. В результате такой обработки из построенного пути исключаются лишнее вершины поворота, сам путь при этом становится короче. Алгоритм Theta*, или более точно – Basic Theta* [7], осуществляет аналогичную обработку, нацеленную на устранение промежуточных элементов пути, но уже во время своей работы: проверка и исключение лишних вершин производится на каждой итерации алгоритма за счет так называемой техники сброса родителя (эта техника уже описывалась ранее в работе и использовалась для создания эффективного алгоритма поиска пути на динамическом ГРД, описанном в разделе 2.2.4). В [234] представлена модификация Basic Theta*, суть которой заключается в использовании эвристической функции специального вида для построения путей к цели, минимизирующих число вершин поворота. Ещё одна модификация Basic Theta* – алгоритм wARC-Theta*, – предоставлена в [235]. С некоторыми модификациями этот алгоритм уже может применяться для решения задачи AC-PF, однако, он не является полным, т.е. не гарантирует отыскание решения, даже если оно существует. Более того на практике число успешно решаемых задач этим методом невелико. В работе [236] была предложена модификация алгоритма wARC-Theta*, названная

wTheta*-LA, реализующая ряд техник, нацеленных на повышение вероятности отыскания решения задачи AC-PF. Было показано, что алгоритм wTheta*-LA существенно превосходит предшественника по числу успешно решенных заданий. Тем не менее, в абсолютных значениях этот показатель при решении практических задач остается низким. Например, согласно экспериментальным данным из [236], процент успешно решенных задач не превышает 14-73% (в зависимости от ограничения на угол отклонения и типа/размера ГРД). Для устранения описанных недостатков в данной работе предлагается ряд новых высоко-эффективных (как с точки зрения объема проводимых вычислений и времени работы, так и с точки зрения гарантии отыскания решений определенного класса) алгоритмов решения задачи AC-PF.

Стоит отдельно отметить, что описанные выше алгоритмы алгоритмы поиска пути на ГРД, учитывающие ограничения на геометрическую форму этого пути, в том числе алгоритмы wARC-Theta* и wTheta*-LA, предназначенные для решения поставленной задачи AC-PF, не гарантируют отыскание минимальных по стоимости путей. В целом, можно утверждать, что с практической точки зрения поиск оптимальных решений задачи AC-PF, т.е. путей минимальной длины (и при этом удовлетворяющих ограничениям на максимальный угол отклонения) не является целесообразным, из-за того, что пространство поиска в этой задаче чрезвычайно велико. Размерность этого пространства аналогична размерности пространства поиска в задаче AA-PFD, которая была рассмотрена в Главе 2, т.к. из каждой вершины ГРД потенциально может быть совершен переход в любую другую вершину ГРД, и он должен быть рассмотрен в ходе поиске и при необходимости отсечен. Более того в задаче AC-PF отсечение по наименьшей стоимости не является корректным, т.к. достижение какой-либо вершины по более короткому пути не гарантирует того, что более длинный путь теперь может быть отсечен, т.к. он может характеризоваться меньшим значением $\alpha_m(\pi)$ и его продолжение может приводить к построению итогового решения, а продолжение более короткого пути с б*о*льшим значением $\alpha_m(\pi)$ – нет. Следовательно, для поиска оптимальных решений нельзя использовать процедуры отсечения по стоимости для промежуточных состояний и поиск становится чрезмерно ресурсозатратным. Поэтому предлагается сосредоточится на разработке и исследовании субоптимальных методов решения задачи AC-PF, обладающим гарантиями относительно отдельных классов решений задачи AC-PF. Такие методы и будут представлены далее.

4.2.2 Базовый алгоритм решения задачи AC-PF

Базовый алгоритм решения задачи AC-PF предлагаемый в работе и носящий имя `LIAN` (от англ. **Li**mited **An**gle, т.е. буквально ограниченный угол) опирается на идею поиска решений определенного класса (с предоставлением гарантий полноты и оптимальности в этом классе). Опишем сначала этот класс, введя предварительно ряд определений и обозначений.

Пусть $v \in V$ – произвольная вершина ГРД, а $\Delta \in \mathbb{N}$ – произвольное натуральное число. Обозначим как $CIRCLE(v,\Delta)$ множество вершин ГРД, аппроксимирующих окружность радиуса Δ с центром в v – см. Рис. 4.4.

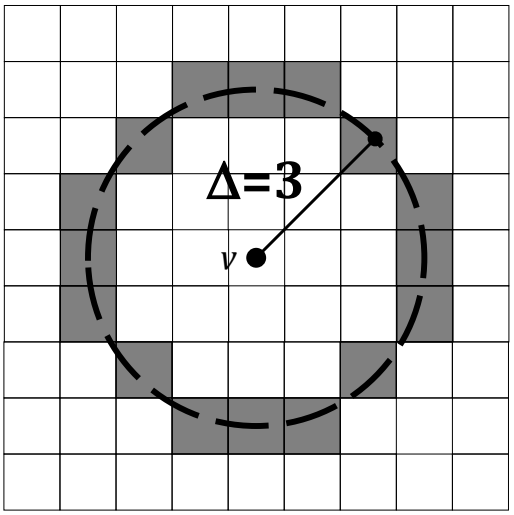


Рисунок 4.4 — Вершины и соответствующие клетки ГРД, образующие множество $CIRCLE(v,\Delta)$, $\Delta = 3$.

Более формально:

$$CIRCLE(v,\Delta) \subseteq \{u \in V |\ \rho(v,\Delta) \cap cell(u) \neq \varnothing\} \tag{4.2}$$

Здесь запись $\rho(v,\Delta)$ обозначает окружность радиуса Δ с центром в точке, соответствующей вершине v, а $cell(u)$ – клетка ГРД (см. Определение 10), ассоциированная с вершиной u.

Здесь и далее будем считать, что построение множества $CIRCLE(v,\Delta)$ может быть осуществлено с помощью некоторой фиксированной процедуры. Например, для построения этого множества может быть использован известный в компьютерной графике алгоритм средней точки (midpoint algorithm) [237], с помощью которого окружности изображаются на растровых дисплеях.

С использованием множества $CIRCLE(v,\Delta)$ введем следующее определение.

Определение 44. Секция $e = (u,v)$, т.ч. $v \in CIRCLE(u,\Delta)$ для некоторой $\Delta \in \mathbb{N}$ – есть Δ-секция.

Неформально Δ-секция – это секция ГРД, длина которой равна Δ либо незначительно отличается от Δ.

С использованием определения Δ-секции введем и определение Δ-пути.

Определение 45. Δ-путь на ГРД, $\pi = (e_1,...,e_n)$, – это путь (в соответствии с Определением 39), т.ч. каждая его секция за исключением может быть последней, является Δ-секцией. То есть:

$$\forall i \in [1,n-1] : \ target(e_i) \in CIRCLE(source(e_i),\Delta). \tag{4.3}$$

Алгоритм `LIAN`, предлагаемый в работе, предназначен именно для поиска Δ-путей. Он является алгоритмом систематического, эвристического поиска, базовые принципы функционирования которого были описаны в Главе 1. Алгоритмы систематического поиска исследуют пространство состояний итерационно и строят дерево частичных решений, добавляя в него элементы (листья) на каждой итерации основного цикла алгоритма. Корню дерева поиска соответствует начальное состояние. Путь от корня до любого листа – частичное решение задачи поиска, состоящие из переходов (ребер дерева) между состояниями поиска (узлы дерева). На каждой итерации поиска из листьев дерева поиска, совокупность которых именуется списком OPEN, выбирается наиболее перспективный элемент (при эвристическом поиске этот выбор определяется в том числе заданной эвристической функцией). Далее происходит генерация состояний-потомков, включающая этап отсечения доминируемых состояний (т.е. состояний рассмотрение которые заведомо не может привести к нахождению решения). В классическом алгоритме эвристического поиска `A*` [11], при использовании монотонной эвристики (подробнее о типах и свойствах эвристических функций см. Раздел 1.3) отсекаются потомки, которые являются дубликатами уже находящихся в дереве поиска состояний, не являющихся листами (т.н. список CLOSED) и некоторые потомки, которые являются дубликатами состояний в списке OPEN. Не отсеченные состояния-потомки добавляются в дерево в качестве листьев (т.е. добавляются в список OPEN) и происходит переход к следующей итерации цикла. Так происходит до тех пор пока не будет выбран для

рассмотрения лист дерева, соответствующий целевой вершине (в этом случае искомый путь может быть восстановлен с помощью трассировки ребер в дереве, ведущих от корня к целевому листу), либо пока не будет исчерпан список OPEN (в этом случае алгоритм возвращает специальный символ *failure*).

Алгоритм `LIAN` реализует вышеописанный принцип организации поиска, однако отличается способом определения состояния поиска (и, как следствие, способами определения состояний-дубликатов), а также процедурой генерации состояний-потомков. Опишем эти отличия более подробно и затем приведем псевдокод алгоритма.

Состояния поиска в алгоритме `LIAN` Для идентификации состояний при стандартном поиске (кратчайшего) пути на ГРД достаточно использовать в качестве идентификатора собственно вершину графа. Т.е. каждое состояние поиска n соответствует определенной вершине v: $n = v$. Если два состояния соответствуют одной и той же вершине, то они являются дубликатами, и одно из них подлежит отсечению. Для учета геометрических ограничений в задаче AC-PF этого не достаточно, т.к. попадание в одну и ту же вершину ГРД из различных предшествующих состояний, влечет к различным углам между секциями (частичного) пути. Поэтому для идентификации состояния предлагается использоваться пару вершина-предшествующее состояние, т.е. состояние, при раскрытии которого, было создано текущее. Формально это можно записать следующим образом: $n = (v,parent(n))$. Здесь $parent(n)$ – это и есть состояние-родитель в дереве поиска для n. Для обозначения вершины, соответствующей состоянию n (т.е. входящей в её индентификатор) будем использовать (достаточно распространенную) запись $n.v$.

С каждым состоянием поиска $n = (v, parent(n))$ в алгоритме `LIAN` ассоциированы (т.е. рассчитываются в ходе работы алгоритма и сохраняются в память) следующие числовые характеристики, необходимые для организации поиска:

- $g(n)$ – стоимость (длина) пути от начальной вершины ГРД до текущей вершины $n.v$ (g-значение состояния);
- $h(n)$ – эвристическая оценка стоимости (длины) пути от текущей вершины $n.v$ до целевой (g-значение состояния);
- $f(n) = g(n) + h(n)$ – оценка стоимости (длины) пути из начальной вершины в конечную, включающего переход $parent(n).v \to n.v$.

Заметим, что, как и в многих других алгоритмах эвристического поиска (в т.ч. представленных в работе), хранение в памяти f-значения состояния формально излишне (т.к. оно однозначно восстанавливается по g- и h-значениям). Тем не менее, поскольку в практических реализациях алгоритмов обычно возникает необходимость индексации множества состояний по f-значению, оно обычно отделяется от $g(n)$ и $h(n)$.

Здесь и далее будем считать, что используемая эвристическая функция является монотонной. Формальное определение такой эвристики приведено ранее в Разделе 1.3. Неформально это означает, что эвристика подчиняется неравенству треугольника. Это разумное предположение, т.к. обычно в подобных задачах в качестве эвристической функции используется Евклидово расстояние и эта эвристика является монотонной. Использование монотонной эвристики гарантирует, что в ходе поиска не может быть найден более короткий путь до состояний из списка CLOSED (т.е. состояний дерева поиска, не являющиеся листами), что весьма полезно для отсечения доминируемых состояний.

Завершим изложение описания состояний поиска алгоритма `LIAN` замечанием о том, что при используемом способе идентификации состояний одна и та же вершина ГРД может порождать множество различных состояний, которые отличаются друг от друга родительскими состояниями.

Генерация состояний-потомков в алгоритме `LIAN` В классических алгоритмах поиска состояния-потомки для состояния n соответствуют вершинам, смежным с вершиной $n.v$. В задаче AC-PF допускаются переходы между произвольными вершинами, в случае если функция los возвращает $true$ на этих вершинах. Более того без таких переходов, в изначальной топологии 4(8)-связного ГРД построить допустимое решение задачи AC-PF невозможно для любого углового ограничения $\alpha_{max} < 45°$. Следовательно, для каждого состояния n должны быть рассмотрены в качестве потенциальных потомков состояния, соответствующие всем вершинам ГРД за исключением $n.v$. Однако такой подход на практике является чрезвычайно не эффективным по двум причинам. Во-первых, коэффициент ветвления дерева поиска (т.е. средние число состояний-потомков) становится весьма большим, что заметно повышает число итераций поиска, необходимых для отыскания решения. Во-вторых, для проверки достижимости каждого потомка необходимо вызывать функцию los, которая

на практике обычно имеет как минимум линейную сложность (например, алгоритм Брезенхема). Следовательно, сложность процедуры генерации потомков может быть оценена как $O(V^2)$. С учетом того, что эта процедура вызывается в алгоритме на каждой итерации, число которых велико из-за высокого коэффициента ветвления, то алгоритм поиска становится мало-эффективным на практике. Для устранения указанных недостатков и повышения вычислительной эффективности в алгоритме `LIAN` предлагается рассматривать лишь Δ-решения, т.е. пути на ГРД, состоящие из Δ-секций – см. Рис. 4.5.

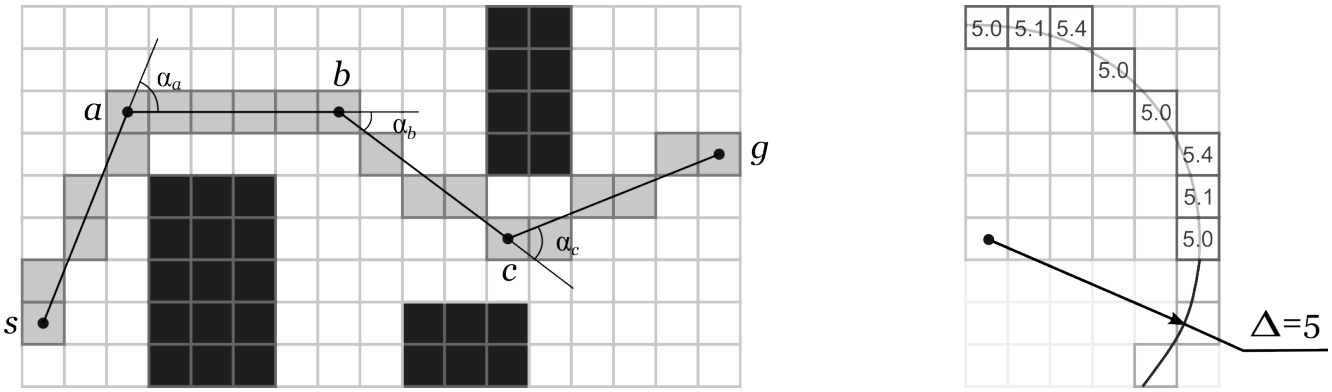


Рисунок 4.5 — Δ-путь на ГРД для $\Delta = 5$.

В алгоритме `LIAN` для генерации состояний-потомком для n сначала выполняется идентификация вершин, которые потенциально могут образовать Δ-секции с началом в $n.v$, т.е. вершин, образующих множество $CIRCLE(n.v, \Delta)$. Затем для каждой вершины проверяется, возможен ли в неё переход (возвращает ли функция los значение $true$) и не нарушается ли заданное ограничение на максимальный угол отклонения пути, если соответствующая Δ-секция будет в него добавлена. Если да, то вершина отбрасывается, если нет, то соответствующее состояние-потомок добавляется в множество потомков.

Псевдокод процедуры генерации состояний потомков алгоритма `LIAN` приведен ниже на Рис. 4.6.

В Строке 1 создается пустое множество потомков и идентифицируются все вершины, которые образуют Δ-секции с началом в вершине $n.v$. Все эти вершины добавляются в список вершин-кандидатов для последующей обработки $SUCC_V$. Если расстояние от текущей вершины, $n.v$, до целевой, v_{goal}, меньше чем Δ, целевая вершина тоже добавляется в список кандидатов $SUCC_V$ (т.к. последняя секция в искомом пути не обязана быть Δ-секцией, см. Определение 45) – Строки 2-3. Далее (Строки 4-13) каждая из вершин $v_{circ} \in SUCC_V$

Процедура `GenerateLIANSuccesors`(n, α_{max}, Δ, v_{goal}, los)**:**

Входные данные: Состояние поиска $n = (v, parent(n))$, ограничение на угол отклонения α_{max}, длина секции Δ, целевая вершина v_{goal}, функция, определяющая возможность перехода, los

Выходные данные: Множество состояний-потомков $SUCC$ для состояния n

1 $SUCC \leftarrow \varnothing$; $SUCC_V \leftarrow CIRCLE(n.v,\Delta)$
2 **if** $\|(n.v,v_{goal})\| < \Delta$ **then**
3 $\quad SUCC_V \leftarrow SUCC_V \cup \{v_{goal}\}$
4 **foreach** $v_{circ} \in SUCC_V$ **do**
5 $\quad$ **if** $los(n.v,v_{circ}) = false$ **then**
6 $\quad\quad$ **continue**
7 $\quad e_1 = (parent(n).v,n.v)$; $e_2 = (n.v,v_{circ})$
8 $\quad$ **if** $\alpha(e_1,e_2) > \alpha_{max}$ **then**
9 $\quad\quad$ **continue**
10 $\quad n_{succ} \leftarrow$ `GenerateSearchNode`(v_{circ}, n)
11 $\quad g(n_{succ}) \leftarrow g(n) + \|e_2\|$
12 $\quad parent(n_{succ}) \leftarrow n$
13 $\quad SUCC \leftarrow SUCC \cup \{n_{succ}\}$
14 **return** $SUCC$

Рисунок 4.6 — Процедура генерации состояний-потомков алгоритма `LIAN`.

обрабатывается последовательно в цикле. Сначала, в Строке 5, проверяется (с помощью вызова функции los) осуществим ли переход из $n.v$ в $v.circ$). Если нет, то вершина пропускается (и, соответственно, состояния-потомка из неё не создаётся). Если переход возможен, то далее (Строки 7-9) проверяется, не будет ли нарушено ограничение на максимальный угол отклонения, если соответствующая Δ-секция, т.е. Δ-секция с началом в $n.v$ и концом в v_{circ}, будет добавлена в путь. Если ограничение нарушается, то вершина отбрасывается, если нет, то в Строках 10-12 создается состояние-последователь $n_{succ} = (v_{circ},n)$ и в Строке 13 оно добавляется в множество $SUCC$. По завершению цикла это множество возвращается в качестве выходного параметра процедуры (Строка 14).

Псевдокод алгоритма `LIAN` С использованием описанной ранее процедуры генерации состояний-потомков `GenerateLIANSuccesors` псевдокод предлагаемого алгоритма решения задачи AC-PF – алгоритма `LIAN`, – представляется в виде, изображенном на Рис. 4.7.

Алгоритм `LIAN`($\mathcal{G}$, v_{start}, v_{goal}, α_{max}, Δ, h, los)**:**

Входные данные: Граф $\mathcal{G}$, начальная и целевая вершины v_{start}, v_{goal}, ограничение на угол отклонения α_{max}, длина секции Δ, монотонная эвристическая функция h, функция, определяющая возможность перехода, los

Выходные данные: Путь π

1 $n_{start} \leftarrow$ `GenerateSearchNode`(v_{start}, null)
2 $g(n_{start}) \leftarrow 0$; $parent(n_{start}) \leftarrow \text{null}$
3 $OPEN \leftarrow \{n_{start}\}$; $CLOSED \leftarrow \varnothing$
4 **while** $OPEN \neq \varnothing$ **do**
5 $n \leftarrow arg\,\min_{n \in OPEN}(f(n) = g(n) + h(n))$
6 $OPEN \leftarrow OPEN \setminus \{n\}$
7 $CLOSED \leftarrow CLOSED \cup \{n\}$
8 **if** $n.v = v_{goal}$ **then**
9 **return** `ReconstructPath`(n)
10 $SUCC \leftarrow$ `GenerateLIANSuccesors`$(n, \alpha_{max}, \Delta, v_{goal}, los)$
11 **foreach** $n_{succ} \in SUCC$ **do**
12 **if** `InClosed`(n_{succ}) **then**
13 **continue**
14 $n_{dubl} \leftarrow$ `FindNode`$(n_{succ}, OPEN)$
15 **if** $n_{dubl} = \text{null}$ **then**
16 $OPEN \leftarrow OPEN \cup \{n_{succ}\}$
17 **else if** $g(n_{dubl}) > g(n_{succ})$ **then**
18 $OPEN \leftarrow (OPEN \setminus \{n_{dubl}\}) \cup \{n_{succ}\}$
19 **return** *failure*

Рисунок 4.7 — Алгоритм эвристического поиска `LIAN`.

В Строках 1–3 производится инициализация. В частности создается начальное состояние поиска n_{start} и добавляется в список OPEN. Список CLOSED пуст. Далее идет основной цикл поиска – Строки 4-18. На каждой итерации цикла в Строке 5 идентифицируется наиболее перспективное состояние в списке OPEN, т.е. состояние с минимальным f-значением. Это состояние, n, извлекается из OPEN и добавляется в CLOSED (Строки 6–7). Если оно соответствует целевой вершине, т.е. $n.v = v_{goal}$, то основной цикл прерывается и в качестве решения задачи AC-PF возвращается путь, восстановленный с помощью родительских указателей ($parent(n)$).

Если же целевая вершина не достигнута, то происходит генерация состояний-потомков для n с помощью ранее введенной процедуры `GenerateLIANSuccessors` (Строка 10). После того, как потомки созданы, осуществляется отсечение доминируемых состояний. Сначала происходит поиск состояния-дубликата для очередного потомка n_{succ} (все потомки обрабатываются последовательно) в списке CLOSED, и, если таковой находится, то n_{succ} отсекается (это корректно, т.к. используется монотонная эвристика, которая гарантирует, что не может быть найдено пути более низкой стоимости к ранее рассмотренным состояниям). Если дублирующего состояния нет в CLOSED, то проверяется его наличие в OPEN (Строка 14). Если его нет и там (Строка 15), то это означает, что подобное состояние поиска встречено впервые, и оно должно быть добавлено в список OPEN. Если же дублирующее n_{succ} состояние – n_{dubl}, – имеется в OPEN, но его g-значение хуже, т.е. $g(n_{dubl}) > g(n_{succ}$ (Строка 17), то оно отсекается, в противном случае отсекается n_{succ}.

После того как все состояния-потомки обработаны осуществляется переход на следующую итерацию основного цикла. Эти итерации продолжаются либо до тех пор, пока не будет извлечено состояние соответствующее цели (Строки 8–9), либо не будет исчерпан список OPEN (Строка 4). В первом случае будет возвращен искомый путь, во втором – специальный символ *failure* (Строка 19), означающий, что путь построить не удалось.

Теоретические свойства алгоритма `LIAN` Пусть p – корректно-поставленная задача AC-PF, а $\mathbf{P}^+$ – множество таких задач, имеющих решение. Обозначим как $\mathbf{P}^\Delta$ подмножество задач $\mathbf{P}^+$, таких что хотя бы одно из их возможных решений является Δ-решением (т.е. Δ-путем на ГРД). Докажем теперь следующую теорему, устанавливающую основные свойства алгоритма `LIAN`.

Теорема 5. *Алгоритм* LIAN *является полным и оптимальным относительно класса задач* $\mathbf{P}^{\Delta}$*, то есть гарантирует отыскание* Δ*-решения, если оно существует, и гарантирует, что стоимость найденного решения не превосходит стоимость любого другого* Δ*-решения (т.е. гарантирует отыскание* Δ*-пути минимальной длины).*

Доказательство. Во-первых, алгоритм LIAN всегда завершает свою работу, либо возвращая решение, либо возвращая специальный символ *failure*. Действительно, алгоритм осуществляет свою работу, пока список OPEN не пуст, см. условие выхода из основного цикла в Строке 4 (Рис. 4.7). Очевидно, что в список OPEN на каждой итерации могут добавляться состояния, соответствующие клеткам ГРД, число которых конечно. Число потенциальных родителей у каждой клетки также конечно. При этом перед добавлением каждого вновь созданного состояния поиска в OPEN (в Строках 12 и 14) проверяется – не было ли оно рассмотрено ранее. Следовательно, число элементов в списке OPEN ограничено сверху. Поскольку на каждой итерации основного цикла алгоритма из списка OPEN всегда удаляется одно состояние и проверяется на соответствие целевой вершине, то рано или поздно этот список будет исчерпан, либо будет найден путь. В обоих случаях алгоритм завершает свою работу.

Во-вторых, любое состояние поиска n однозначно определяет уникальный путь от начальной вершины до вершины $n.v$ (с помощью родительских указателей). При этом при рассмотрении произвольного состояния n (в т.ч. для начального) генерируются все состояния-потомки n_{succ}, т.ч. секция $(n.v, n_{succ}.v)$ является Δ-секцией и при этом добавление её в путь не нарушает ограничение на максимальный угол отклонения. Следовательно, рассматриваются все возможные варианты продолжения Δ-пути от начальной вершины до текущей. Это, в свою очередь, означает, что рано или поздно будут рассмотрены все возможные пути, начинающиеся в начальной вершине и состоящие из Δ-секций и, если среди них будет путь, соответствующий решению задачи AC-PF (т.е. путь заканчивающийся в целевой вершине), то он будет идентифицирован (Строка 8 алгоритма) и возвращен в качестве решения.

Наконец, алгоритм LIAN использует такую же стратегию поиска в пространстве состояний, что и классический алгоритм эвристического поиска A*, который гарантирует отыскание кратчайшего пути при условии использования монотонной эвристики. Поскольку используемая LIAN эвристика (Евклидово

расстояние) является монотонной, то LIAN возвращает Δ-путь минимальной длины. ■

4.2.3 Модифицированный алгоритм решения задачи AC-PF

Основным недостатком представленного в предыдущем разделе алгоритма LIAN, предназначенного для решения задачи AC-PF, является тот факт, что неудачная, с точки зрения конкретной задачи AC-PF, инициализация параметра Δ может привести к не отысканию решения, т.к. LIAN гарантирует полноту только в классе задач $\mathbf{P}^{\Delta}$, т.е. задач среди возможных решений которых есть как минимум один Δ-путь на ГРД в соответствии с Определением 45.

Заметим, что если Δ мало, то мало и число последователей, генерируемых на каждой итерации основного цикла для для пополнения списка OPEN, т.к. чем меньше Δ, тем меньше вершин ГРД принадлежит множеству $CIRCLE(n.v,\Delta)$, из которых и формируются состояния-потомки для текущего рассматриваемого состояния n. При определенной конфигурации непроходимых областей (препятствий) на ГРД это может привести в конечном итоге к исчерпанию списка OPEN и завершению работы алгоритма без нахождения пути. Подобная ситуация представлена на Рис. 4.8.

На Рис. 4.8 изображен фрагмент ГРД, на котором начальная вершина находится слева, целевая – справа за препятствием (не показана на рисунке). Необходимо найти путь с максимальным ограничением на угол отклонения $\alpha_{max} = 25°$. Серым цветом отмечены вершины, которые были рассмотрены алгоритмом LIAN при $\Delta = 3$. Как видно из рисунка, число потенциальных последователей, генерируемых на каждом шаге алгоритма и удовлетворяющих ограничению на угол отклонения в 25°, достаточно мало (2-3 шт. на каждой итерации алгоритма), вследствие чего алгоритм не способен найти путь, огибающий препятствие (хотя такой существует).

Аналогично, использование слишком большого (для конкретной задачи планирования) значения параметра Δ может не приводить к успеху по схожей причине – малого размера списка OPEN. Несмотря на то, что при больших значениях Δ, число потенциальных последователей на каждой итерации алго-

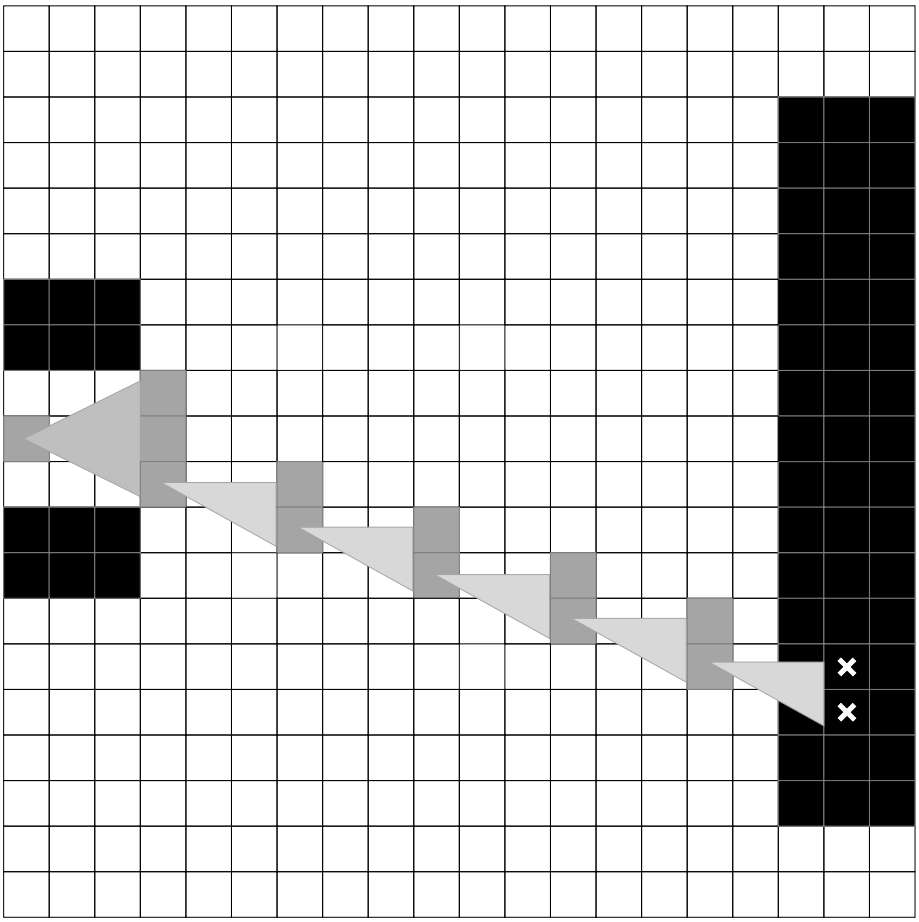


Рисунок 4.8 — Фрагмент пространства поиска алгоритма LIAN при малом значении входного параметра Δ ($\Delta = 3$).

ритма весьма велико, в случае, если рассматриваемая область поиска (фрагмент ГРД) содержит большое число непроходимых клеток (препятствий), то велика вероятность, что значительная часть потенциальных последователей не будет добавлена в OPEN из-за нарушения условий на проходимость секции. Подобная ситуация проиллюстрирована на Рис. 4.9.

На Рис. 4.9 изображен фрагмент ГРД, на котором алгоритм LIAN осуществляет поиск решения задачи AC-PF для $\alpha_{max} = 25°$. При этом значение параметра Δ выбрано равным 10. На рисунке показана секция, для которой существует допустимое продолжение пути, однако при $\Delta = 10$ для концевой вершины этой секции не удается создать ни одного потомка, т.к. переход в любого из них оказывается невозможен из-за имеющихся ограничений (проходимость секции, ограничение на максимальный угол отклонения).

Для устранения указанных выше проблем, связанных с неподходящей (для конкретной задачи AC-PF) инициализацией параметра Δ, предлагается

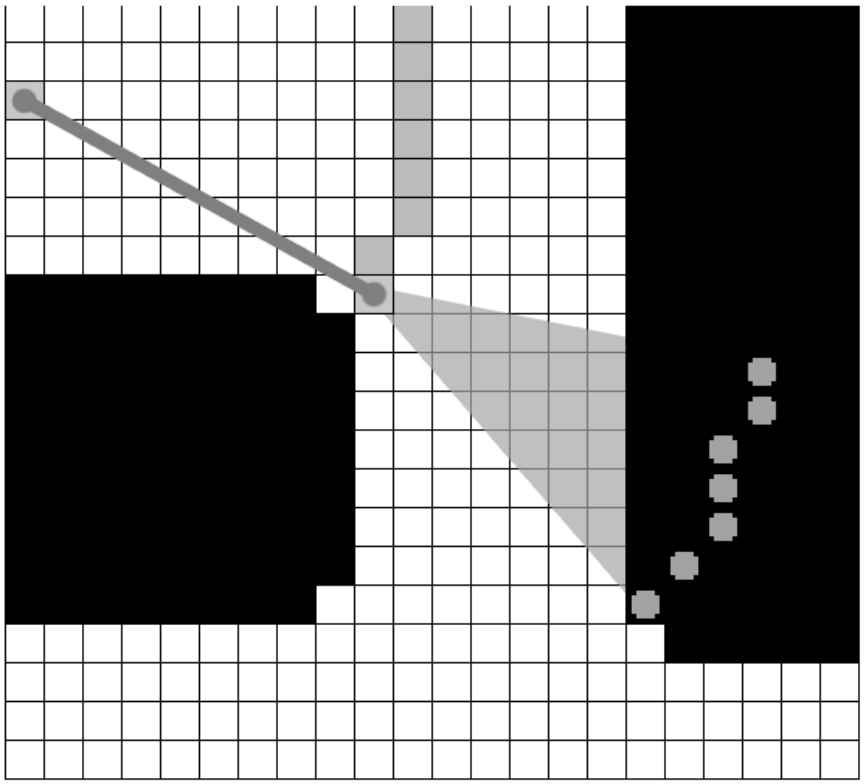


Рисунок 4.9 — Фрагмент пространства поиска алгоритма LIAN при большом значении входного параметра Δ ($\Delta = 10$).

использовать динамическое изменение значения этого параметра в ходе работы алгоритма с учетом особенностей конкретной задачи планирования. Опишем процедуру такой динамической подстройки значения параметра более подробно.

Динамическая настройка параметра Δ в ходе работы алгоритма Будем считать, что каждое состояние поиска, как и ранее идентифицируется парой 'вершина-родитель', т.е. $n = (v,parent(n))$ и ассоциируется помимо g-, h-, f-значений с Δ-значением, $\Delta(n)$, равным тому значению параметра Δ, с помощью которого было сгенерировано состояние n. Обозначим как Δ_{cur} значение параметра в момент обработки текущего состояния поиска. Тогда $\Delta(n_{succ}) = \Delta_{cur}$ для любого состояния-потомка $n_{succ} \in SUCC(n)$.

Пусть теперь на очередной итерации алгоритма рассматривается состояние n и для него сформировано множество допустимых состояний-последователей $SUCC$, которое соответствует вершинам ГРД, лежащих на аппроксимации

окружности радиуса Δ_{cur} с центром в клетке $n.v$, удовлетворяющих ограничению на максимальный угол отклонения и образующих проходимые секции. Если множество $SUCC$ пусто, то в стандартном алгоритме `LIAN` состояние n помещается в список CLOSED без потомков, т.е. становится листом дерева, дальнейший рост ветвей из которого не возможен. Вместо этого же предлагается оставить это состояние в списке OPEN, но с меньшим Δ-значением, а именно $\Delta(n) = \Delta_{cur}/2$, т.е. значение параметра уменьшается в два раза[1] – см. Рис. 4.10. Если при повторном раскрытии состояния n список $SUCC$ опять оказывается пуст, то значение параметра снова уменьшается в два раза и т.д. Процесс уменьшения Δ-значения состояния происходит до тех пор, пока не будет выполнено одно из двух условий: либо множество $SUCC$ будет не пусто, либо текущее значение параметра окажется ниже заранее предустановленного порогового значения Δ_{min}. В первом случае у состояния n появляются потомки, которыми пополняется список OPEN (т.е. текущая ветвь поиска не прерывается).

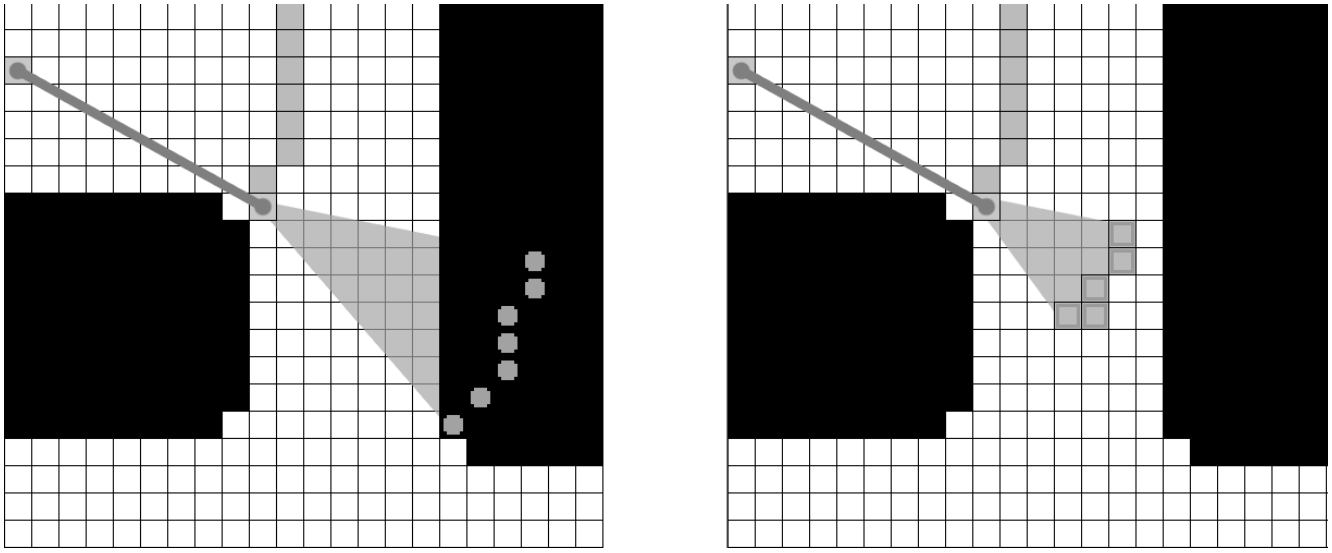


Рисунок 4.10 — Динамическое изменение длины Δ-секции в ходе поиска пути.

На Рис. 4.10 изображено пространство поиска при динамическом изменении значения параметра Δ. Ограничение на максимальный угол отклонения составляет 25°. Начальное значение параметра Δ равняется 10. При неудачном раскрытии элемента (изображено на рисунке слева), Δ приравнивается к 5 и происходит повторное раскрытие, позволяющее не прерывать текущую ветвь поиска (изображено на рисунке справа).

[1]Поскольку параметр Δ является целочисленным, то здесь и далее в разделе везде предполагается целочисленное деление и умножение.

Заметим далее, что уменьшение $\Delta(n)$ при поиске происходит тогда, когда текущая ветвь поиска приводит к локальной области ГРД, содержащей большое число непроходимых клеток. Логично предположить, что после того как эта область пройдена, значение $\Delta(n)$ может быть опять увеличено для уменьшения числа поворотов на оставшемся фрагменте пути и ускорения поиска. Для такого увеличения предлагается следующая стратегия: если текущее значение параметра для раскрываемого элемента, $\Delta(n)$, меньше некоторого изначально заданного значения Δ_{max}, а значение $\Delta(parent(n))$ совпадает с $\Delta(n)$ (то есть совершено два перехода с одинаковой длинной Δ-секции), то текущее значение увеличивается, т.е. выполняется присваивание $\Delta(n) \leftarrow 2 \cdot \Delta(n)$. Интуитивно это можно трактовать следующим образом – область с нетривиальным расположением препятствий пройдена и теперь можно продвигаться к цели б*о*льшими шагами (т.е. используя Δ-секции б*о*льшей блины).

Модификацию алгоритма `LIAN`, использующую вышеуказанные техники динамического изменения (уменьшения и обратного увеличения) входного параметра, будем называть `D-LIAN` (от англ. Dynamic LIAN). Благодаря возможности динамического изменения Δ-значений состояний поиска в ходе работы алгоритма, последний обладает возможностью подстраиваться под конкретную задачу AC-PF, а именно – под расположение непроходимых областей (препятствий) на ГРД с учетом начальной и целевой вершины. Как и `LIAN`, алгоритм `D-LIAN` является параметризованным – в нём используются используются следующие два параметра: Δ_{max} – начальное, максимальное значение радиуса окружности, используемой для генерации множества потенциальных последователей, и Δ_{min} – минимальное значение.

Псевдокод алгоритма `D-LIAN` Одно из отличий алгоритма `D-LIAN` от предшественника заключается в способе генерации состояний-последователей, включающем динамическую подстройку параметра Δ. Приведем сначала псевдокод этой процедуры – см. Рис. 4.11.

Приведенный на Рис. 4.11 псевдокод во многом повторяет псевдокод процедуры генерации состояний-потомков в алгоритма `LIAN` (см. Рис. 4.6). Ключевое отличие – в Строках 10-13. Так в Строке 10 проводится проверка, совпадает ли Δ-значение текущего состояния (т.е. состояния для которого генерируются потомки) с Δ-значением его родительского состояния. Если да (и при этом

Процедура `GenerateDLIANSuccesors`(n, α_{max}, Δ_{max}, v_{goal}, los)**:**

Входные данные: Состояние поиска $n = (v, parent(n))$, ограничение на угол отклонения α_{max}, максимальная длина секции Δ_{max}, целевая вершина v_{goal}, функция, определяющая возможность перехода, los

Выходные данные: Множество состояний-потомков $SUCC$ для состояния n

1 $SUCC \leftarrow \varnothing$; $SUCC_V \leftarrow CIRCLE(n.v,\Delta(n))$
2 **if** $\|(n.v,v_{goal})\| < \Delta(n)$ **then**
3 $\quad SUCC_V \leftarrow SUCC_V \cup \{v_{goal}\}$
4 **foreach** $v_{circ} \in SUCC_V$ **do**
5 $\quad e_1 = (parent(n).v,n.v)$; $e_2 = (n.v,v_{circ})$
6 $\quad$ **if** $los(n.v,v_{circ}) = false$ **or** $\alpha(e_1,e_2) > \alpha_{max}$ **then**
7 $\quad\quad$ **continue**
8 $\quad n_{succ} \leftarrow$ `GenerateSearchNode`(v_{circ}, n)
9 $\quad g(n_{succ}) \leftarrow g(n) + \|e_2\|$
10 $\quad$ **if** $\Delta(n) = \Delta(parent(n))$ **and** $\Delta(n) < \Delta_{max}$ **then**
11 $\quad\quad \Delta(n_{succ}) \leftarrow 2 \cdot \Delta(n)$
12 $\quad$ **else**
13 $\quad\quad \Delta(n_{succ}) \leftarrow \Delta(n)$
14 $\quad parent(n_{succ}) \leftarrow n$
15 $\quad SUCC \leftarrow SUCC \cup \{n_{succ}\}$
16 **return** $SUCC$

Рисунок 4.11 — Процедура генерации состояний-потомков алгоритма `D-LIAN`.

Δ-значение меньше, чем максимальная допустимая длина секции), то для состояний потомков Δ-значение увеличивается (Строка 11), если нет – остается таким-же, как и у текущего состояния (Строка 13). Это и есть реализация механизма подстройки параметра Δ в сторону увеличения, когда с текущим значением Δ было сгененировано уже два состояния подряд (что, интуитивно, означает, что в данный момент путь строится в области свободной от препятствий и поэтому целесообразно использовать Δ-секции увеличенной длины для

быстрого продвижения к цели). Механизм подстройки параметра в сторону уменьшения инкорпорирован в основной алгоритм, псевдокод которого представлен ниже на Рис. 4.12.

Алгоритм `D-LIAN`($\mathcal{G}$, v_{start}, v_{goal}, α_{max}, Δ_{min}, Δ_{max}, h, los)**:**

Входные данные: Граф $\mathcal{G}$, начальная и целевая вершины v_{start}, v_{goal}, ограничение на угол отклонения α_{max}, минимальная и максимальная длина секции Δ_{min}, Δ_{max}, монотонная эвристическая функция h, функция, определяющая возможность перехода, los

Выходные данные: Путь π

1 $n_{start} \leftarrow$ `GenerateSearchNode`(v_{start}, null)
2 $\Delta(n_{start}) \leftarrow \Delta_{max}$; $g(n_{start}) \leftarrow 0$; $parent(n_{start}) \leftarrow$ null
3 $OPEN \leftarrow \{n_{start}\}$; $CLOSED \leftarrow \varnothing$
4 **while** $OPEN \neq \varnothing$ **do**
5 $n \leftarrow arg\min_{n \in OPEN}(f(n) = g(n) + h(n))$
6 $OPEN \leftarrow OPEN \setminus \{n\}$
7 **if** $n.v = v_{goal}$ **then**
8 **return** `ReconstructPath`(n)
9 $SUCC \leftarrow$ `GenerateDLIANSuccesors`(n,α_{max},Δ_{max},v_{goal},los)
10 **if** $SUCC = \varnothing$ **and** $\Delta(n) > \Delta_{min}$ **then**
11 $\Delta(n) \leftarrow \Delta(n)/2$
12 $OPEN \leftarrow OPEN \cup \{n\}$
13 **continue**
14 `UpdateOPENandCLOSED()`
15 $CLOSED \leftarrow CLOSED \cup \{n\}$
16 **return** *failure*

Рисунок 4.12 — Алгоритм эвристического поиска `D-LIAN`.

Поскольку псевдокод алгоритма `D-LIAN` в целом аналогичен коду предшественника (см. Рис 4.7), опишем далее лишь основные отличия. Во-первых, при создании начального состояния (корня дерева поиска) n_{start} теперь задается и его Δ-значение равное Δ_{max}. Во-вторых, теперь на каждой итерации

основного цикла, после извлечения из списка OPEN наиболее перспективного состояния (Строки 5-6), это состояние не добавляется сразу в список CLOSED как ранее. Это делается, т.к. возможна ситуация, когда с текущим Δ-значением невозможно сгенерировать ни одного допустимого состояния-потомка (Строки 9-10) и тогда Δ-значение уменьшается, как это было писано ранее, а само состояние помещается обратно в OPEN (для последующей генерации потомков уже с обновленным $\Delta(n)$) – Строки 10-13. Только если множество допустимых потомков не пусто (либо $\Delta(n)$ уже не может быть более уменьшено), происходит обновление списков OPEN и CLOSED – Строка 14. Это обновление полностью аналогично тому, что было в алгоритме `LIAN` (Строки 11-18 на Рис. 4.7), и поэтому в данном псевдокоде используется компактное обозначение `UpdateOPENandCLOSED`. Лишь после этого текущее состояние добавляется в список CLOSED (Строка 15), чем исключается из дальнейшего рассмотрения.

Теоретические свойства алгоритма `D-LIAN` Докажем ряд утверждений, характеризующих основные теоретические свойства алгоритма `D-LIAN`.

Утверждение 6. *Алгоритм* `D-LIAN` *корректно завершается за конечное число шагов.*

Доказательство Алгоритм `D-LIAN` осуществляет свою работу, пока список OPEN не пуст. В список OPEN на каждой итерации могут добавляться состояния, соответствующие клеткам ГРД, число которых конечно. Число потенциальных родителей у каждой клетки также конечно. При этом перед добавлением каждого вновь созданного состояния поиска в OPEN проверяется – не было ли оно рассмотрено ранее. Следовательно, число элементов в списке OPEN ограничено сверху. Теперь необходимо показать, что любое состояние, попавшее в список OPEN, рано или поздно будет из него извлечено и добавлено в список CLOSED (список состояний, не подлежащих дальнейшему рассмотрению).

Для этого заметим, что, если после извлечения состояния n из OPEN удалось сгенерировать его потомков, то условие проверки в Строке 10 не выполняется, Строки 11-13 не выполняются, а выполняются Строка 14 и Строка 15, на которой и происходит добавление состояния n в список CLOSED (т.е. исключение его из дальнейшего рассмотрения).

Если же после извлечения состояния n из OPEN сгенерировать его потомков не удалось, то производится попытка уменьшения его Δ-значения и состояние остается в OPEN. Однако число таких попыток ограничено, т.к. при целочисленном делении и присваивании $\Delta(n) \leftarrow \Delta(n)/2$ (Строка 11), значение $\Delta(n)$ рано или поздно достигнет величины, меньшей либо равной Δ_{min}, и тогда условие в Строке 10 перестанет выполняться и в Строке 15 состояние будет опять же добавлено CLOSED.

Итак, любое состояние, попавшее в список OPEN, рано или поздно будет из него удалено и добавлено в список CLOSED (более рассматриваться не будет), что и было необходимо показать. ■

Алгоритм `D-LIAN` имеет два параметра – Δ_{min} и Δ_{max}, задающие интервал изменения длин Δ-секций, из которых производится построение результирующего пути, удовлетворяющего ограничению на заданный максимальный угол отклонения. При этом инициализация начального состояния в Строке 2 алгоритма происходит так, что его Δ-значение равняется Δ_{max}. Интуитивно это означает, что в первую очередь производится попытка построения решения, состоящего из Δ-секций длины Δ_{max}. Более того, можно доказать, что алгоритм `D-LIAN` обладает строгими гарантиями относительно класса Δ_{max}-решений (т.е. путей на ГРД образованных такими секциями, что каждая из них, за исключением быть может последней, является Δ_{max}-секцией).

Утверждение 7. *Алгоритм* `D-LIAN` *является полным относительно класса задач* $\mathbf{P}^{\Delta_{max}}$*, то есть гарантирует отыскание решения задачи AC-PF, если для неё существует* Δ_{max}*-решение, и гарантирует корректное завершение в противном случае.*

Доказательство Предположим, что Δ_{max}-решение существует. Рассмотрим первую итерацию основного цикла алгоритма. На ней список OPEN содержит лишь одно состояние поиска – начальное, обозначаемое n_{start}, и это состояние будет извлечено из OPEN для генерации потомков. При этом генерация состояний-потомком будет проводится с Δ-значением, равным Δ_{max}, т.к. на этапе инициализации в Строке 2 было выполнено присваивание $\Delta(n_{start}) \leftarrow \Delta_{max}$. Следовательно, состояния-потомки для n_{start} будут соответствовать вершинам ГРД, образующим Δ-секции, где величина Δ равняется Δ_{max}. Т.к. Δ_{max}-решение задачи существует (по предположению), то множество потомков,

$SUCC$, не будет пустым, условие проверки в Строке 10 не будет выполнено и, соответственно, элементы множества $SUCC$ будут добавлены в список OPEN в Строке 14. То есть в список OPEN будут добавлены состояния-потомки n_{succ}, такие что секция $(n_{start}.v, n_{succ}.v)$ является Δ_{max}-секцией. Среди этих состояний найдется по крайней мере одно такое, что при его обработке в список OPEN будут добавлены элементы $n_{succ'}$, т.ч. любая секция $(n_{succ}.v,n_{succ'}.v)$ является Δ_{max}-секцией (иначе нарушается условие существования Δ_{max} решения). Те же самые рассуждения касаются и $n_{succ'}$. Следовательно, список OPEN на любой итерации алгоритма, содержит по крайней мере один элемент $n_{\Delta_{max}}$, такой что путь до него, определяемый последовательностью $(n_{start}, ..., parent(parent(n_{\Delta_{max}})), parent(n_{\Delta_{max}}), n_{\Delta_{max}})$, состоит лишь из Δ_{max}-секций. Список OPEN может (но не обязан) содержать и другие элементы (состояния поиска), соответствующие частичным путям, включающим в себя Δ-секции, не являющиеся Δ_{max}-секциями. Эти частичные пути могут рассматриваться алгоритмом благодаря технике динамического изменения Δ-значения состояния. Таким образом множество частичных путей, рассматриваемых алгоритмом `D-LIAN` с параметрами Δ_{max} и Δ_{min} включает в себя множество частичных путей алгоритма `LIAN` с параметром Δ равным Δ_{max}. Поскольку `LIAN`, гарантирует, что при существовании решения в классе Δ-путей, один из таких путей будет найден, то и алгоритм `D-LIAN` гарантирует отыскание пути при существовании Δ_{max}-пути (т.к. `D-LIAN` не содержит никаких механизмом отсечения состояний поиска (частичных путей) отличных от `LIAN`).

Если же Δ_{max}-решения не существует, то алгоритм `D-LIAN` даже в этом случае корректно завершится (см. Утверждение 6). ■

Заметим, что в общем случае класс решений, отыскиваемых алгоритмом, не ограничивается только Δ_{max}-решениями. Действительно, корректное завершение алгоритма `D-LIAN` в случае отсутствия Δ_{max}-решения, не означает, что алгоритм завершится, не вернув искомый путь. Возможна ситуация, когда из списка OPEN уже исключены все состояния (частичные пути), соответствующие Δ_{max}-решениям (т.к. Δ_{max}-решения нет), но в нём же находятся состояния, не соответствующие Δ_{max}-решениям, которые были добавлены в список OPEN благодаря технике динамической подстройки Δ-значения состояния. Вполне возможно (но не гарантируется), что рассмотрение этих состояний приведет к отысканию решения не являющемуся Δ_{max} путём. Следовательно, класс

решений, отыскиваемых алгоритмом D-LIAN шире, чем класс решений, отыскиваемых LIAN.

Рассмотрим теперь произвольную задачу AC-PF, допускающую Δ_{max}-решение, т.е. $p \in \mathbf{P^{\Delta_{max}}}$. Обозначим оптимальное решение этой задачи как $\pi^*_{p,\Delta_{max}}$, а решение этой задачи, найденное алгоритмом D-LIAN (оно гарантировано будет найдено в силу Утверждения 7) как $\pi_{p,\texttt{D-LIAN}}$. Тогда, в силу вышесказанного справедливо следующее утверждение.

Утверждение 8. *$len(\pi_{p,\texttt{D-LIAN}}) \leqslant len(\pi^*_{p,\Delta_{max}})$ (длина пути, отыскиваемого алгоритмом* D-LIAN*, не превосходит длину оптимального Δ_{max}-пути).*

Доказательство. Для доказательства утверждения достаточно заметить, что стратегия выбора элемента для рассмотрения на очередном шаге алгоритма D-LIAN аналогична стратегии, применяемой в алгоритме A*. Поскольку последний гарантирует нахождение пути наименьшей стоимости при использовании монотонной эвристики, а эвристика, используемая D-LIAN (Евклидово расстояние) – монотонна, то можно утверждать, что D-LIAN в качестве решения возвращает кратчайший из рассмотренных путей. ■

Утверждения 6, 7, 8 можно объединить и сформулировать в виде следующей теоремы.

Теорема 6. *Алгоритм* D-LIAN *является полным и оптимальным относительно класса задач $\mathbf{P}^{\Delta_{max}}$, то есть гарантирует отыскание решения задачи AC-PF, если для неё существует Δ_{max}-решение (и гарантирует корректное завершение в противном случае), более того стоимость отыскиваемого решения (длина пути) не превосходит стоимости оптимального Δ_{max}-решения.*

Перейдём теперь к экспериментальным исследованиям предложенных алгоритмов (LIAN и D-LIAN).

4.3 Экспериментальные исследования

Проводились экспериментальные исследования следующих алгоритмов решения задачи AC-PF: предложенные в работе алгоритмы LIAN и D-LIAN, известные ранее алгоритмы, предназначенные для решения аналогичной задачи

(адаптированные под рассматриваемую постановку задачи AC-PF) – `Theta*-LA` и `wTheta*-LA`. Все алгоритмы были реализованы на языке C++ с использованием одинаковых техник программирования и идентичных структур данных для обеспечения честного сравнения. Исследования проводились на настольном компьютере следующей конфигурации: Intel Core 2 Duo, Q8300 @ 2.5Ghz, 2Gb RAM, Windows 7 Ultimate SP1 64 bit.

Используемые исходные данные Для проведения исследований использовались 3 коллекции ГРД, которые как и ранее в работе, будем называть картами. В качестве основной коллекции карт была использована коллекция Moscow Maps (MM). MM – авторская коллекция, основанная на открытой геоинформационной базе данных OpenStreetMaps [238]. Из этой базы были извлечены описания 100 реальных фрагментов городского ландшафта Москвы размером 1347 на 1347 метров. Каждый фрагмент был преобразован в ГРД размером 500×500 клеток (с вершинами в центрах клеток), т.е. отдельной клетке соответствовала область размером 2.7×2.7 м^2. Непроходимыми были помечены клетки, соответствующие высотным зданиям. Начальная и целевая клетка выбирались так, что расстояние между ними составляло не менее 400 клеток (т.е. не менее 1 км.). Для каждой карты было сгенерировано 5 различных заданий (т.е. пар старт-финиш). Общее число задач планирования в коллекции составило 500 штук. Эти задачи моделируют реальные навигационные задания, возникающие при планировании траектории маловысотного полета беспилотного летательного аппарата в городских условиях. Шаг дискретизации, использованный при построении графовой модели по гео-данным (2.7 м.), является подходящим для обеспечения полета компактных аппаратов мультироторного типа, диаметр которых находится в диапазоне 0.3-0.5 метров. В работе [233] был произведен расчет максимального угла отклонения для одного из подобных летательных аппаратов (а именно для AscTech Hummingbird): при полете на крейсерской скорости 4.5 м/с, он составляет (приблизительно) 25°. Именно это ограничение на α_{max} было взято в качество основного. Дополнительно использовались ограничения в 20° и 30°. Можно предположить, что эти ограничения характерны для этого же аппарата, но при более высокой и низкой скоростях полета соответственно.

Коллекции Warcraft III (WIII) и Baldurs Gate (BG) являются частью открытой коллекции MovingAI [219], часто используемой исследователями в об-

ласти планирования траектории на плоскости для экспериментальной оценки качества и скорости работы предлагаемых алгоритмов. Карты этих коллекций имеют размер 512×512 клеток и являются моделями виртуальных миров, используемых в одноименных компьютерных играх. При этом для WIII характерны открытые пространства, т.е. ГРД этой коллекции можно считать сходными с графами коллекции MM, в то время как для BG наоборот характерны закрытые пространства (коридоры, комнаты и т.д.). Примеры карт всех трех коллекций представлены на Рис.4.13.

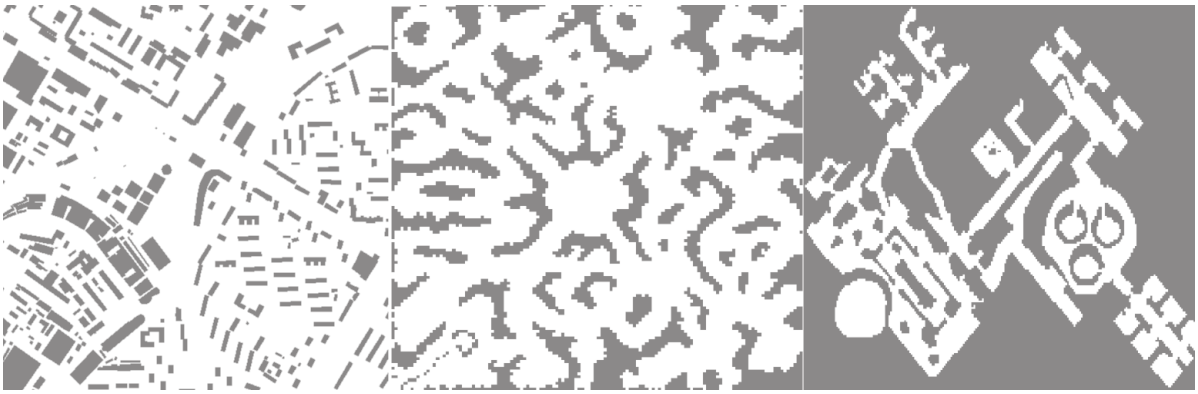


Рисунок 4.13 — Карты, используемые для экспериментального исследования алгоритмов решения задачи AC-PF.

Примеры карт, использованных в ходе экспериментальных исследований алгоритмов решения задачи AC-PF.

Для каждой карты WIII и BG были выбраны 5 различных заданий из общей базы заданий MovingAI. Для выбора этих заданий использовалась следующая стратегия. Все возможные задания, привязанные к конкретным картам, сортировались по убыванию длин путей, не учитывающих ограничение на максимальный угол отклонения (информация о таких длинах хранится в базе данных MovingAI). Далее выбирались задания, соответствующие первым 5 путям для каждого графа. Как показали результаты дальнейших исследований, такая стратегия позволила добиться однородности заданий на всех коллекциях – в среднем, длина путей, учитывающих геометрические ограничения, одинакова для всех трёх используемых коллекций.

В завершении описания исходных данных заметим, что все задания для всех коллекций были подобраны так, что для них существует решение при ограничениях на угол отклонения $\alpha_{max} > 20°$, т.е. все задания являются потенциально разрешимыми с учетом всех имеющихся ограничений. Существование этих путей было подтверждено с помощью запуска алгоритма полного перебора.

Отслеживаемые показатели эффективности Одиночный эксперимент состоял в поиске пути, учитывающего заданные геометрические ограничения. В каждом эксперименте отслеживались следующие выходные параметры, характеризующие эффективность работы алгоритма:

- *success* – показатель успешности выполнения задания, т.е. найден ли искомый путь или нет;
- *time* – время работы алгоритма;
- *treesize* – число обработанных и сохраненных в оперативной памяти состояний поиска (размер дерева поиска на последней итерации алгоритма);
- *pathlength* – длина найденного пути;
- *rotations* – число поворотов в найденном пути.

Прокомментируем далее эти выходные параметры, на которые далее будем ссылаться как на индикаторы (эффективности алгоритма).

Поскольку все исследуемые алгоритмы не гарантируют отыскание решения в общем случае, то с практической точки наиболее важным выходным параметром является *success* – было ли решено задание алгоритмом или нет. Для более удобного сравнения алгоритмов предлагается использовать агрегированное значение этого параметра, *success rate*, равное проценту (или доле) успешно успешно решенных заданий от общего числа обработанных заданий.

Индикаторы *time* и *treesize* – это основные параметры, характеризующие вычислительную эффективность алгоритма. Чем меньше значения *time* и *treesize*, тем меньше времени тратит алгоритм на поиск решения и тем меньше хранит он в памяти промежуточных состояний поиска. Очевидно, параметр *treesize* косвенно характеризует затраты по памяти. В связи с этим для удобства обозначения будем использовать запись *memory* для обозначения этого индикатора.

Наконец, индикаторы *pathlength* и *rotations* характеризуют качество решения конкретной задачи AC-PF, т.е. длину найденного пути (чем меньше, тем лучше) и число поворотов в пути (чем меньше, тем лучше). Заметим, что ни один из рассматриваемых алгоритмов в ходе своей работы никак не пытается минимизировать число поворотов, тем не менее с практической точки зрения это важный показатель, который разумно использовать для оценки алгоритма.

Предварительная серия экспериментов Перед началом работы была проведена предварительная серия экспериментальных исследований, направленная на установление подходящих границ значений основного параметра предлагаемых в работе алгоритмов `LIAN` и `D-LIAN` – Δ. Для проведения этих экспериментов была использована усеченная выборка заданий: использовалась только коллекция карт MM, значение максимального угла отклонения составляло 25°. По результатам экспериментов было установлено, что при $\Delta<5$ и при $\Delta>20$, существенно снижается процент успешно решенных алгоритмами задач. Поэтому для последующих серий экспериментов значения параметра Δ выбирались из диапазона $[5, 20]$.

Также были проведены эксперименты по настройке веса эвристической функции для предлагаемых алгоритмов. Известно, что любой алгоритм эвристического поиска допускает использование взвешенной эвристики вида $\hat{h}(n) = w \cdot h(n)$, где $h(n)$ – исходная эвристическая функция, а $w > 1$ – вес эвристики. Интуитивно, чем больше значение $w > 1$, тем сильнее осуществляется фокусировка поиска на целевое состояние. На практике это ведёт к повышению скорости работы, при незначительном увеличении длины отыскиваемых путей. Для алгоритмов `LIAN`, `D-LIAN` эффект от техники взвешивания эвристической функции оказывается весьма ярко выраженным – число рассмотренных состояний сокращается в десятки раз, а вместе с тем кратно повышается и скорость работы. Этот эффект наглядно проиллюстрирован на Рис. 4.14.

На Рис. 4.14 изображен фрагмент одной из карт коллекции MM (старт находится справа, финиш – слева). На верхней части рисунка изображен результат решения этого задания алгоритмом `LIAN` без взвешивания эвристики, справа – с взвешиванием. Цветом изображены вершины, соответствующие рассмотренным состояниям. Чем интенсивней цвет, тем больше состояний поиска, определяемых этой вершиной, было рассмотрено[2]. Тёмно-розовым цветом изображен найденный алгоритмом путь. Очевидно, что найденный при использовании взвешенной эвристики путь (нижняя часть рисунка) лишь незначительно отличается по длине от пути, найденного без взвешивания (верхняя часть рисунка). При этом число рассмотренных состояний в первом случае на порядок меньше.

В ходе экспериментов вес эвристики w варьировался в интервале $[1.01; 10]$. Было установлено, что наиболее выгодный баланс между повышением скоро-

[2]Здесь уместно напомнить, что в алгоритме `LIAN` одна и та же вершина может определять множество различных состояний поиска, отличающихся родительскими указателями.

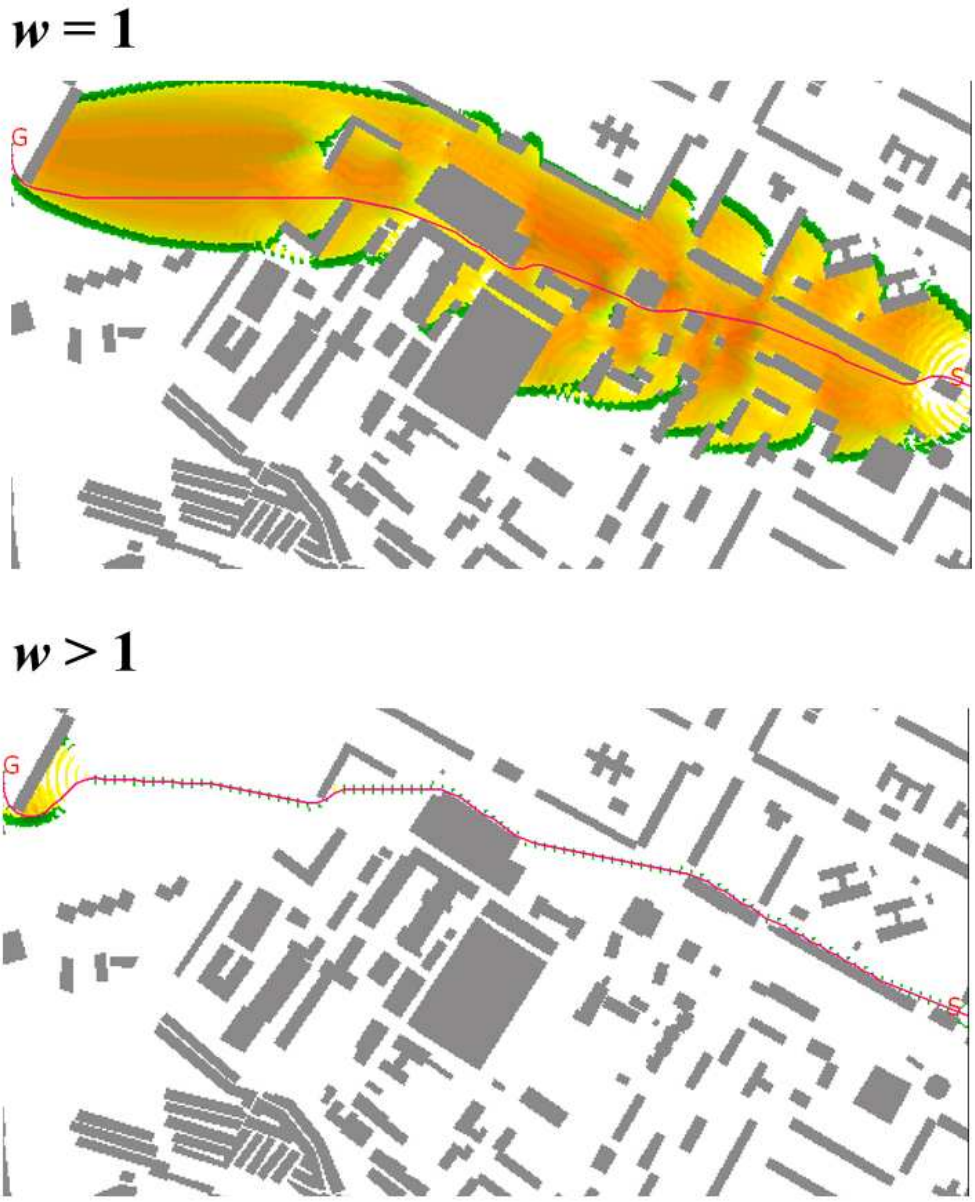


Рисунок 4.14 — Динамическое изменение длины Δ-секции в ходе поиска пути.

сти работы и (незначительным) снижением качества решений обеспечивает использованием веса $w = 2$. Именно этот вес и использовался в дальнейших экспериментах.

Первая серия экспериментов В первой серии экспериментов исследовались алгоритмы `LIAN`, `Theta*-LA` и `wTheta*-LA`. Для алгоритма `LIAN` значение параметра Δ равнялось (в соответствии с результатами предварительной серии экспериментов) 5, 10, 20. Для обозначения алгоритма `LIAN` с $\Delta = x$ здесь и далее будет использоваться запись `LIAN-x`. Например, `LIAN-10` означает, что речь идёт об алгоритме `LIAN` с $\Delta = 10$.

Результаты экспериментов представлены в Табл.4.1 и Рис. 4.15-4.16. Рассмотрим их более подробно.

Таблица 4.1 — Процент успешно решенных задач AC-PF для алгоритмов `Theta*-LA`, `wTheta*-LA` и `LIAN`.

	Moscow Maps			Baldur's Gate			Warcraft III		
	20°	25°	30°	20°	25°	30°	20°	25°	30°
`LIAN-5`	99.4%	100.0%	100.0%	100.0%	100.0%	100.0%	99.4%	100.0%	100.0%
`LIAN-10`	96.8%	99.0%	99.6%	87.7%	96.2%	98.4%	92.7%	96.1%	98.9%
`LIAN-20`	87.2%	91.4%	94.0%	61.6%	72.8%	81.3%	76.6%	80.5%	84.4%
`wTheta*-LA`	56.4%	86.8%	93.8%	33.8%	76.2%	91.4%	49.4%	80.0%	87.7%
`Theta*-LA`	4.4%	6.2%	14.0%	10.6%	10.9%	11.7%	4.0%	5.5%	10.0%

Табл. 4.1 содержит информацию о проценте успешно решенных задач каждым из алгоритмов для каждой коллекции карт и значения максимально-допустимого угла поворота пути. Представленные данные позволяют сделать следующие выводы. Во-первых, алгоритм `Theta*-LA` (который представляет собой наивную адаптацию алгоритма `Theta*` для решения задач AC-PF), оказывается малоприменим на практике из-за неспособности отыскивать решения в подавляющем числе случае (более 90% задач остались нерешенными вне зависимости от типа карты и ограничения на максимальный угол отклонения). Продвинутая модификация этого алгоритма, `wTheta*-LA`, демонстрирует более высокую эффективность в плане способности решать предъявляемые задачи. Процент успешно решенных заданий существенно выше – в некоторых случаях он возрастает более, чем в 10 раз (см., например, результаты на коллекции карт Moscow Maps для $\alpha_{max} = 25°$). Тем не менее при низких значениях ограничения на максимальный угол отклонения, $\alpha_{max} = 20°$, около половины предъявляемых заданий не решается.

Алгоритм `LIAN-20` показывает схожий с `wTheta*-LA` процент успеха при ограничениях на максимальный угол отклонения в 25° и 30°, и гораздо более высокий результат при $\alpha_{max} = 20°$. Алгоритм `LIAN-10` показывает ещё более высокий результат, его процент успеха в среднем составляет 95%. Наконец `LIAN-5` решает почти все предъявляемые задачи.

С учетом вышесказанного можно сделать однозначный вывод о превосходстве предлагаемого в работе алгоритма `LIAN` над известными аналогами по способности решать задачи AC-PF.

Проанализируем теперь вычислительную эффективность алгоритма LIAN – см. Рис. 4.15.

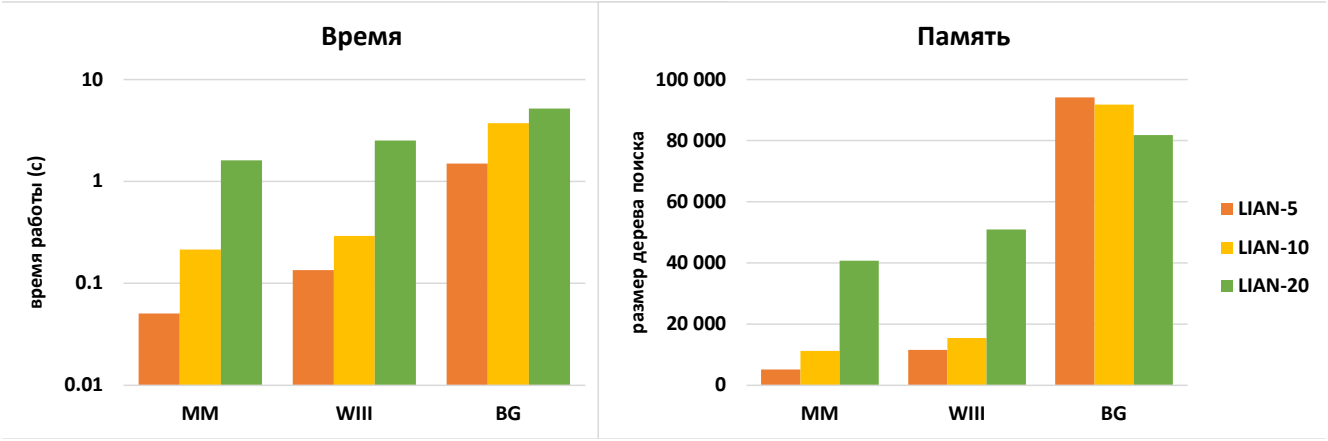


Рисунок 4.15 — Время и память, затрачиваемые алгоритмами LIAN-5, LIAN-10, LIAN-20 на решение задач AC-PF.

На Рис. 4.15 слева показано усредненное время работы алгоритма LIAN для различных значения параметра Δ на всех коллекциях карт, справа – усредненное число хранимых в памяти состояний поиска (размер дерева поиска на последней итерации алгоритма). Усреднение проводилось по всем успешно решенным конкретным алгоритмом заданиям с $\alpha_{max} = 25°$ (результаты для $\alpha_{max} = 20°$, $\alpha_{max} = 30°$ аналогичны представленным и опущены для удобства восприятия). Отдельно заметим, что на диаграмме времени работы ось значений (ось Y) имеет логарифмический масштаб.

Как видно из диаграмм, LIAN-5 заметно превосходит LIAN-10 и LIAN-20 по показателям временной и емкостной эффективности. Например, на коллекции карт Moscow Maps среднее время работы LIAN-5 составляет менее 0.1 секунды, в то время как для LIAN-20 – более 1 секунды, т.е. разница во время работы – более, чем на порядок (по числу хранимых состояний поиска, т.е. по памяти, разница аналогичная).

дополнительно стоит заметить, что задания из коллекции BG требует гораздо больше времени на обработку. Это можно объяснить тем фактом, что карты в этой коллекции моделируют преимущественно закрытые помещения с большим количеством узких проходов, коридоров, комнат и т.д. Очевидно, что поиск пути с ограничением на максимальный угол отклонения на таких картах – более трудоемкая задача по сравнению с поиском на картах, представляющих собой преимущественно открытую местность, как в коллекциях MM и WIII. Здесь уместно обратить внимание и на то, что размер дерева поиска для

LIAN-5 выше аналогичного показателя для LIAN-10, LIAN-20 на картах коллекции BG. Это объясняется тем, что на диаграммах приведены усредненные значения показателя, и усреднение производилось по всем успешно решенным конкретным алгоритмом заданиям. Очевидно, что часть наиболее сложных заданий, которые были решены LIAN-5, но не были решены LIAN-10, LIAN-20, внесли существенный вклад в рост значения показателя для LIAN-5. Этим же можно объяснить то обстоятельство, что именно на коллекции BG разница во времени работы между LIAN-5 и LIAN-20 менее выражена, чем на коллекции MM. LIAN-20 просто не смог решить достаточно большое число заданий из коллекции BG, тем самым в его усредняемую выборку попали лишь достаточно легкие для решения задания.

Сравним теперь такие показатели качества работы алгоритмов как длина отыскиваемых путей и число поворотов – см. Рис. 4.16.

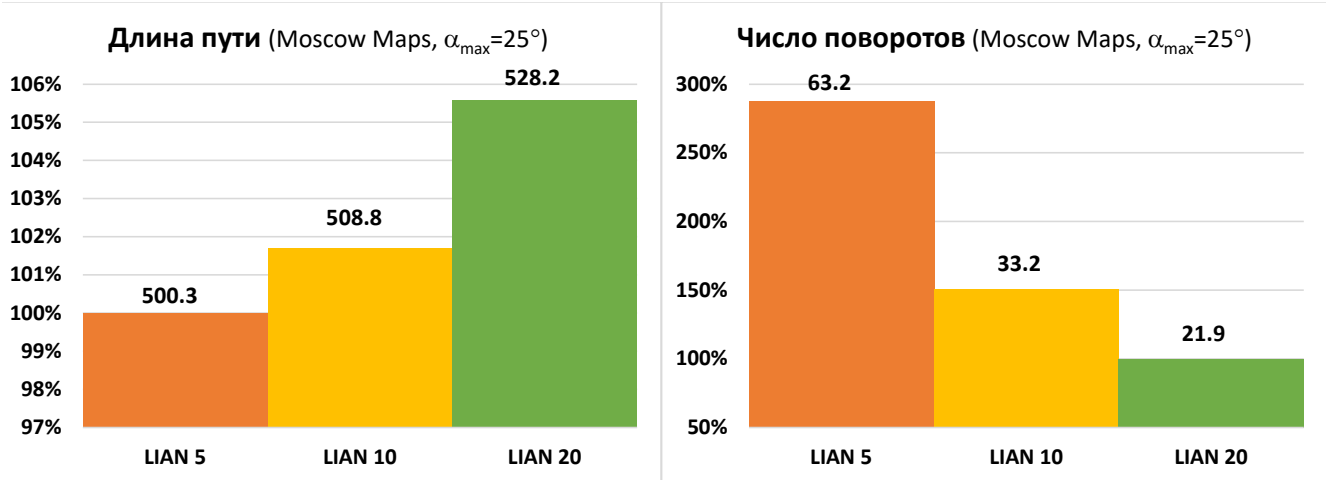


Рисунок 4.16 — Длина путь и число поворотов в пути для алгоритмов LIAN-5, LIAN-10, LIAN-20 при решении задач AC-PF.

На Рис. 4.16 слева приведена усредненная длина отыскиваемых различными вариантами алгоритма LIAN путей, справа – число поворотов (т.е. изменений направления движения) в этих путях. Усреднение проводилось для каждого алгоритма в отдельности по заданиям коллекции Moscow Maps с $\alpha_{max} = 25°$. Поскольку это случай характеризуется почти 100% успешностью решения заданий, усредненные данные можно корректно сравнивать между собой. Данные нормализованы по лучшему результату. Для показателя качества “длина пути” показатель алгоритма LIAN-5 взят за 100%, для числа поворотов – LIAN-20. Абсолютные значения индикаторов качества приведены над столбцами диаграммы.

Очевидно, что по длине путей все варианты алгоритма LIAN отличаются незначительно. Максимальная разница наблюдается между алгоритмами LIAN-5 и LIAN-20 и составляет 5.6% в пользу LIAN-5. Разница же между LIAN-5 и LIAN-10 и вовсе составляет менее 2%. С другой стороны по числу поворотов алгоритмы различаются между собой гораздо сильнее и здесь явный фаворит LIAN-20 – пути, построенные этим алгоритмом в среднем содержат в 1.5 раза меньше поворотов по сравнению с LIAN-10 и почти в 3 раза меньше поворотов по сравнению с LIAN-5. С практической точки зрения в мобильной робототехнике (где наиболее востребовано решение задач AC-PF) число поворотов может быть даже более важным показателем качества построенного пути, т.к. вдоль более гладкого пути гораздо легче обеспечить точное следование, что повышает безопасность выполнения миссии автономным роботом. Поэтому представляет особый интерес вопрос – можно ли объединить в одном алгоритме высокую вычислительную эффективность LIAN-5 и низкое число поворотов LIAN-20. Для ответа на этот вопрос была проведена вторая серия экспериментов. Перейдём к её рассмотрению.

Вторая серия экспериментов Во второй серии экспериментов исследовалось то, насколько алгоритм D-LIAN, т.е. алгоритм, использующий технику адаптивной подстройки параметра Δ по ходу поиска, способен повысить процент решаемых задач и скорость работы по сравнению с LIAN-20 и LIAN-10 (т.е. приблизиться по этим показателям к LIAN-5) и при этом сохранить низкое число поворотов в пути.

Эксперименты проводились на тех же коллекциях данных (задания), что и раньше – Moscow Maps, Warcraft III, Baldur's Gate, использовались такие же ограничения на максимальный угол отклонения – $\alpha_{max} \in [20°,25°,30°]$. Были исследованы три варианта алгоритма D-LIAN: D-LIAN-10-5, D-LIAN-20-10, D-LIAN-20-5. Здесь запись D-LIAN-x-y означает, что в алгоритме использовались следующие значения параметров: $\Delta_{max} = x$, $\Delta_{min} = y$. Т.е. параметр Δ динамически варьировался в указанных пределах – алгоритм начинает свою работу, используя секции длины Δ_{max} и, при необходимости, уменьшает эту длину (и затем увеличивает обратно).

Результаты экспериментов представлены в Табл. 4.2- 4.3 и на Рис. 4.17- 4.18. Опишем их более подробно.

Таблица 4.2 — Процент успешно решенных задач AC-PF для алгоритмов LIAN и D-LIAN.

	Moscow Maps			Baldur's Gate			Warcraft III		
	20°	25°	30°	20°	25°	30°	20°	25°	30°
LIAN-5	99.4%	100.0%	100.0%	100.0%	100.0%	100.0%	99.4%	100.0%	100.0%
LIAN-10	96.8%	99.0%	99.6%	87.7%	96.2%	98.4%	92.7%	96.1%	98.9%
LIAN-20	87.2%	91.4%	94.0%	61.6%	72.8%	81.3%	76.6%	80.5%	84.4%
D-LIAN-10-5	98.2%	99.4%	99.8%	90.4%	98.1%	98.6%	96.7%	99.4%	100.0%
D-LIAN-20-10	92.4%	96.0%	97.4%	74.7%	84.5%	88.0%	83.3%	86.7%	93.9%
D-LIAN-20-5	94.8%	97.0%	98.2%	83.7%	89.1%	89.9%	88.3%	96.1%	98.9%

Как видно из таблицы, процент успешно решенных заданий при использовании алгоритма D-LIAN заметно повышается по сравнению с LIAN. Например, для коллекции Baldur's Gate и $\alpha_{max} = 20°$ процент успешно решенных заданий для LIAN-20 составлял 61.6%, а для D-LIAN-20-5 – 83.7%, т.е. рост составил 22.1% (почти треть). На других картах и для других значений ограничения на максимальный угол отклонения аналогично наблюдается стабильный рост показателя. Вполне ожидаемо, что наименьший рост характерен для $\alpha_{max} = 30°$, т.к. в этом случае изначально алгоритмы LIAN-10 и LIAN-20 демонстрировали относительно высокий процент успеха. Также ожидаемо, что повышение эффективности более заметно при переходе от LIAN-20 к D-LIAN-20-5, нежели при переходе от LIAN-10 к D-LIAN-10-5, т.к. именно LIAN-20 число нерешенных заданий было наибольшим.

Представляет интерес вопрос – какой процент заданий, нерешенных алгоритмом LIAN-x, решил алгоритм D-LIAN-x-y? Ответ на него дан в Табл. 4.3.

Таблица 4.3 — Рост числа решенных задач при переходе от LIAN к D-LIAN (в процентах).

	Moscow Maps			Baldur's Gate			Warcraft III		
	20°	25°	30°	20°	25°	30°	20°	25°	30°
D-LIAN-10-5	43.8%	40.0%	50.0%	21.7%	50.0%	12.5%	53.8%	85.7%	100.0%
D-LIAN-20-10	40.6%	46.5%	43.3%	34.0%	56.9%	64.3%	28.6%	68.6%	39.3%
D-LIAN-20-5	59.4%	65.1%	70.0%	57.6%	59.8%	45.7%	50.0%	80.0%	92.9%

Прокомментируем, как следует читать таблицу на примере ячейки первого столбца последней строки, т.е. Moscow Maps, 20°, `D-LIAN-20-5`. Значение 59.4% в этой ячейке говорит о том, что алгоритм `D-LIAN-20-5` нашел решения для 59.4% задач, которые не были решены ранее алгоритмом `D-LIAN-20`. Как видно из представленных данных, техника динамического варьирования параметра Δ позволяет существенно повысить шанс отыскания решения. В среднем более половины заданий, нерешаемых алгоритмом `LIAN`, решаются алгоритмом `D-LIAN`. В ряде случаев, этот показатель превышает 80% и доходит до 100% (см., например, коллекцию Warcraft III для $\alpha_{max} = 25°$ и $\alpha_{max} = 30°$), т.е. алгоритм `D-LIAN` решает все задания, с которыми не справился `LIAN` (для соответствующих значений параметров Δ, Δ_{min}, Δ_{max}).

Проанализируем теперь быстродействие алгоритма `D-LIAN` в сравнении с `LIAN`. Поскольку разные версии этих алгоритмов характеризуются разным число успешно решенных задач для сравнения разумно использовать показатель "общее число успешно решенных задач за время x с." (чем больше, тем лучше). Подобный график для коллекции заданий Moscow Maps и $\alpha_{max} = 25°$ (аналогичные графики для других коллекций и ограничений на максимальный угол отклонения демонстрируют схожие тренды) приведен на Рис. 4.17.

На рисунке по оси X отложено время в секундах, а по оси Y – процент заданий, который решен алгоритмом за соответствующий временной бюджет. Например, алгоритм `D-LIAN` за 1 секунду смог найти решение 83% задач (из всей коллекции), в то время как такой же показатель для `D-LIAN-20-10` составляет – 87% (+5%), а `D-LIAN-20-5` – 93% (+10%). Очевидно, что использование техники динамической подстройки параметра Δ заметно повышает производительность при $\Delta = 20$. При этом для $\Delta = 10$ эффект менее выражен. Это можно объяснить тем, что при таком значении параметра изначально решается достаточно большое число заданий и дальнейший рост эффективности затруднителен. Стоит дополнительно прокомментировать почему алгоритмы `D-LIAN-10-5` и `D-LIAN-20-5` не достигают такой же эффективности, как и `LIAN-5`, несмотря на то, что они имеют возможность использовать использовать Δ-секции длины 5. Дело в том, что изначально алгоритмы семейства `D-LIAN-x-y` (где $x > y$) начинают построение дерева частичных решений, используя Δ-секции длины x (20 в рассматриваемом случае) и только в случае невозможности генерации потомков на некотором шаге начинают использовать секции меньшей длины (10 и 5), при этом возможно, что для отыскания реше-

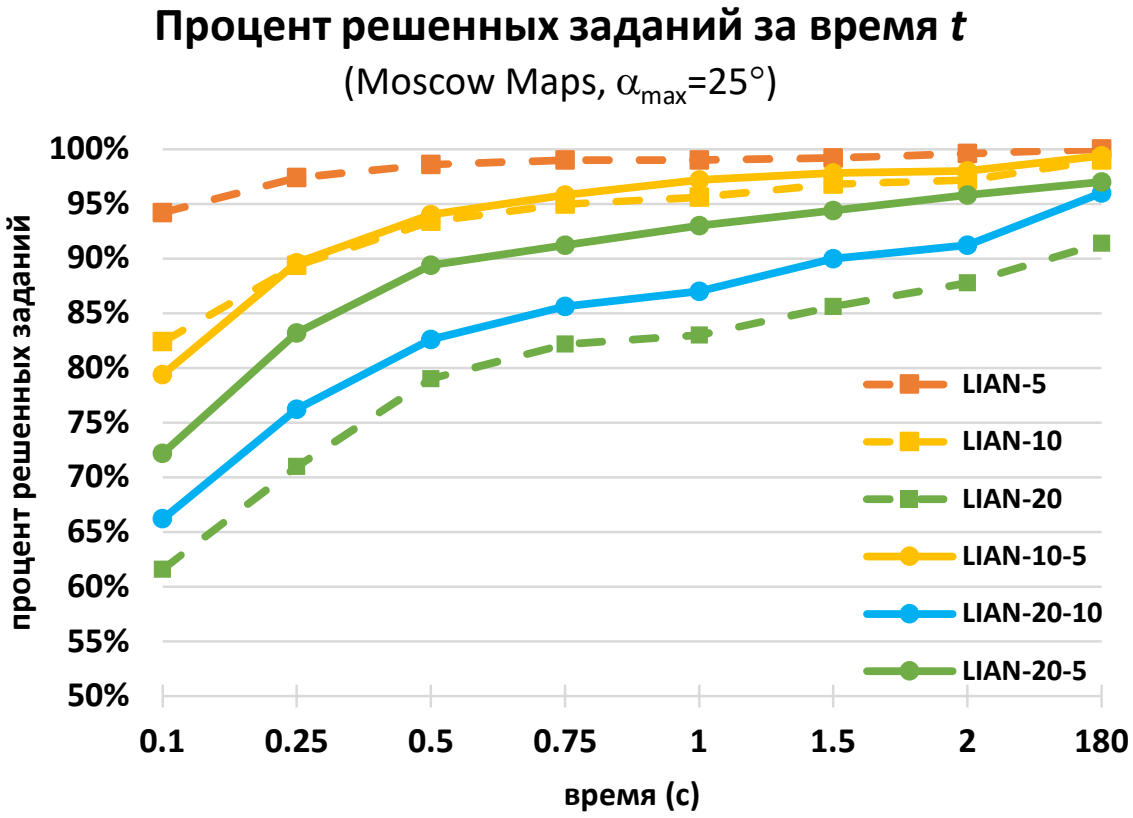


Рисунок 4.17 — Быстродействие алгоритмов `D-LIAN` и `LIAN` при решении задач AC-PF.

ния, подобные секции необходимо было использовать с начала поиска (что и делает `LIAN-5`), иначе решение не может быть найдено.

Проанализируем, наконец, качество решений (длина, число поворотов в пути), отыскиваемых алгоритмами `D-LIAN` – см. Рис. 4.18.

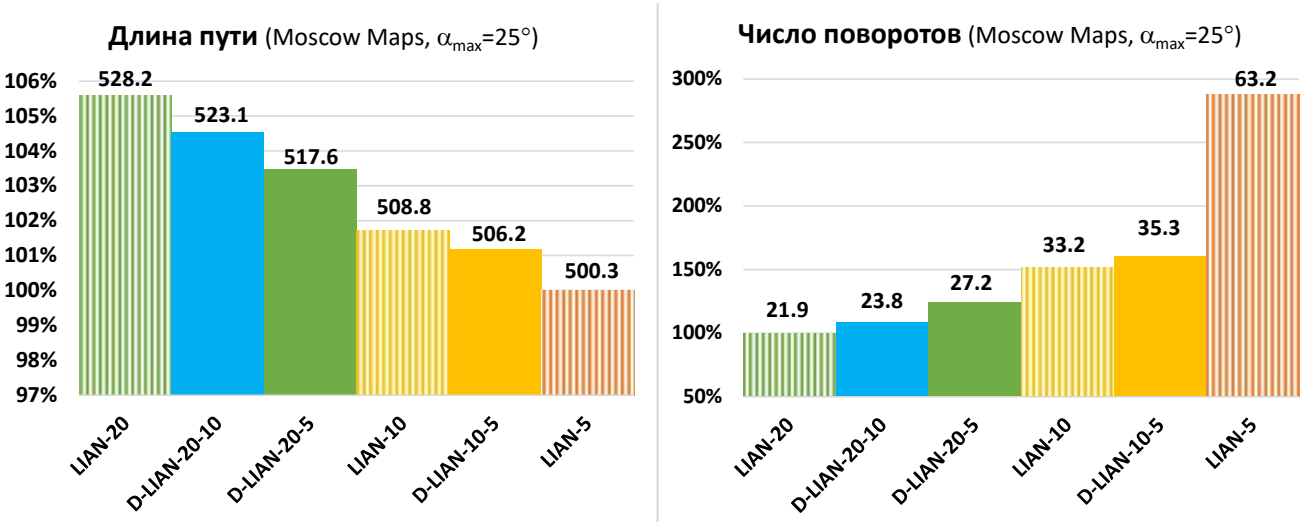


Рисунок 4.18 — Длина пути и число поворотов в пути для алгоритмов `D-LIAN` и `LIAN` при решении задач AC-PF.

На рисунке приведены диаграммы усредненной нормированной длины пути и числа поворотов для коллекции Moscow Maps, $\alpha_{max} = 25°$ (для этих заданий процент успеха для всех алгоритмов близок к 100%, поэтому сравнение усредненных значений корректно). Нормировка производится на наилучший результат: для длины пути – это показатель алгоритма `LIAN-5`, для числа поворотов – `LIAN-20`. Значения, достигаемые алгоритмами семейства `LIAN` отмечены штрихом, `D-LIAN` – сплошной заливкой. Абсолютные значения выходных параметров показаны над соответствующими столбцами.

Как и ранее, видно, что повышение значения Δ (Δ_{max} для `D-LIAN`) приводит к незначительному повышению длины пути – не более чем на 6% для любого из алгоритмов относительно лучшего результата. При этом применение техники динамического варьирования параметра Δ, т.е. использование алгоритма `D-LIAN`, позволяет снизить длину пути по сравнению с соответствующей версией `LIAN`. Так, например, длина пути для `D-LIAN-20-5` составляет в среднем 517.6, а для `LIAN` 528.2. Что касается числа поворотов, то оно повышается, но весьма незначительно. Так наилучшее значение этого показателя составляет 21.9 для алгоритма `D-LIAN`, а для алгоритма `D-LIAN-20-5` это значение равняется 27.2, в то время как у `LIAN-5` – 63.2 (т.е. почти в три раза больше). Таким образом, можно сделать вывод о том, что предлагаемый в работе алгоритм `D-LIAN` обеспечивает незначительное изменение показателей качества отыскиваемых путей по сравнению с алгоритмом `LIAN` – в сторону небольшого уменьшения для длины пути и в сторону увеличения для числа поворотов. При этом, число успешно решенных задания для `D-LIAN` заметно выше, чем для соответствующий версий `LIAN`.

В завершении приведем несколько визуализацией путей, построенных предложенными алгоритмами и путей, которые были построены для этих же карт и начальных и конечных положений, но без учета ограничения на максимальный угол отклонения – см. Рис. 4.19.

На рисунке сверху изображены фрагменты трёх различных заданий из коллекции Moscow Maps и пути, построенные предлагаемыми в работе алгоритмами; в нижнем ряду изображены пути для этих же заданий, но построенные с помощью алгоритма `Theta*`, который не учитывает геометрических ограничений. Как видно из рисунка, предлагаемые алгоритмы строят более плавные траектории, не содержащие резких изменений направления движения, что делает следование по ним более безопасным для различных мобильных робо-

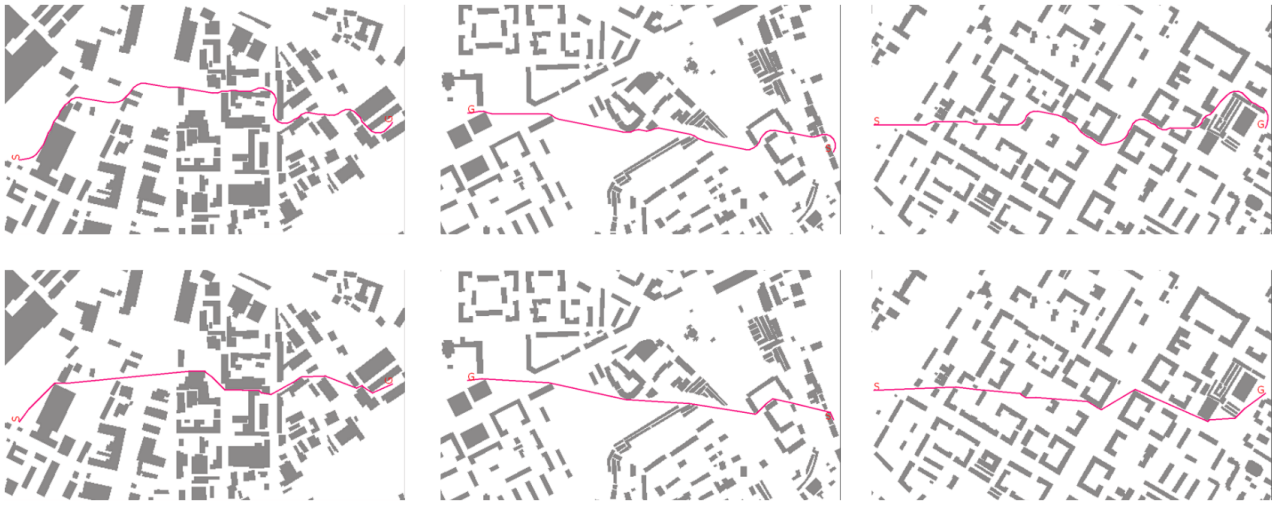


Рисунок 4.19 — Примеры построенных путей с учетом ограничения на максимальный угол отклонения и без учета.

тотехнических систем (например, малых беспилотных летательных аппаратов при совершении полета на достаточно высокой скорости).

4.4 Выводы по главе

В данной главе работы рассматривалась задача поиска пути на графе регулярной декомпозиции с учетом геометрических ограничений, а именно с учетом ограничения на максимальный угол отклонения между прямолинейными секциями, образующими путь. Такая постановка задачи может быть весьма актуальной в робототехнических приложениях, в частности при планировании траектории для мобильных роботов, для которых следование вдоль пути, содержащего резкие повороты может быть небезопасно (из-за невозможности мгновенного изменения направления движения). Для решения поставленной задачи был предложен новый алгоритм эвристического поиска – `LIAN`. Предложенный алгоритм является параметризованным и обладает строгими теоретическими гарантиями относительно класса решений, определяемых входным параметром. Предложена техника динамического варьирования параметра по ходу поиска, направленная на расширение класса допустимых решений, и алгоритм планирования, использующий эту технику, – `D-LIAN`. Проведено обширное экспериментальное исследование алгоритмов и сравнение их с

имеющимися мировыми аналогами. Для такого исследования использовались графы регулярной декомпозиции, являющиеся моделями реальной городской местности, и графы, включенные в известную в сообществе коллекцию заданий MovingAI. Результаты экспериментального исследования показали, что предложенный алгоритм `LIAN` существенно превосходит известные аналоги по успешности решения задач с различными ограничениями на максимальный угол отклонения, а алгоритм `D-LIAN` позволяет ещё более повысить число успешно-решаемых заданий при сохранении таких показателей качества конструируемых путей как длина пути и число поворотов в пути. Таким образом, предложенные методы и алгоритмы могут применяться в реальных робототехнических приложениях, связанных с автономной навигацией мобильного робота в сложно-структурированной среде, когда необходимо не только строить кратчайший путь для достижения целевой позиции, но и учитывать при этом такие ограничения робототехнической системы, как невозможность мгновенного изменения направления движения.

Глава 5. Методы машинного обучения для поиска пути на графах регулярной декомпозиции

Предыдущие главы работы были посвящены различным вариациям задачи поиска (кратчайшего) пути на графе регулярной декомпозиции, вложенном в метрическое пространство. Базовой методологией решения этих задач является эвристический поиск, при котором исследуется пространство состояний, индуцируемое вершинами графа. Алгоритмы эвристического поиска исследуют это пространство, осуществляя итеративное построение дерева частичных решений и выбирая на каждой итерации лист дерева для дальнейшей генерации состояний-потомков. Этот выбор производится с помощью аддитивной оценочной функции, состоящей из двух компонент: рассчитанной стоимости достижения текущего состояния из начального состояния и эвристической оценки стоимости пути из текущего состояния в целевое. Последняя компонента – эвристика (эвристическая функция) – интуитивно предназначена для направления роста дерева поиска в направлении целевого состояния с тем, чтобы достичь этого состояния как можно скорее. От того, насколько верно эвристика оценивает реальную стоимость напрямую зависит число итераций алгоритма поиска и, следовательно, его вычислительная эффективность (как по времени так и по памяти). Для оценки стоимости пути до целевого состояния на ГРД обычно используется расстояние Манхеттена или Евклидово расстояние. Эти функции удовлетворяют ряду условий (т.н. допустимость и монотонность эвристической функции – см. Главу 1), что позволяет алгоритму поиска их использующему предоставлять важные теоретические гарантии (отыскание оптимального решения, отсутствие многократного рассмотрения одного и того же состояния поиска и др.). Однако эти функции являются малоинформативными в практически важных случаях, когда ГРД моделирует собой окружающее пространство мобильного агента (например, колесного робота), содержащее препятствия, т.к. они неверно оценивают расстояние от произвольной вершины ГРД до целевой, из-за того что не учитывают наличие препятствий и их конкретную конфигурацию. Как следствие это приводит к плохой фокусировке поиска: число исследуемых состояний возрастает, и поиск становится не эффективным. Пример подобного эффекта приведен на Рис. 5.1.

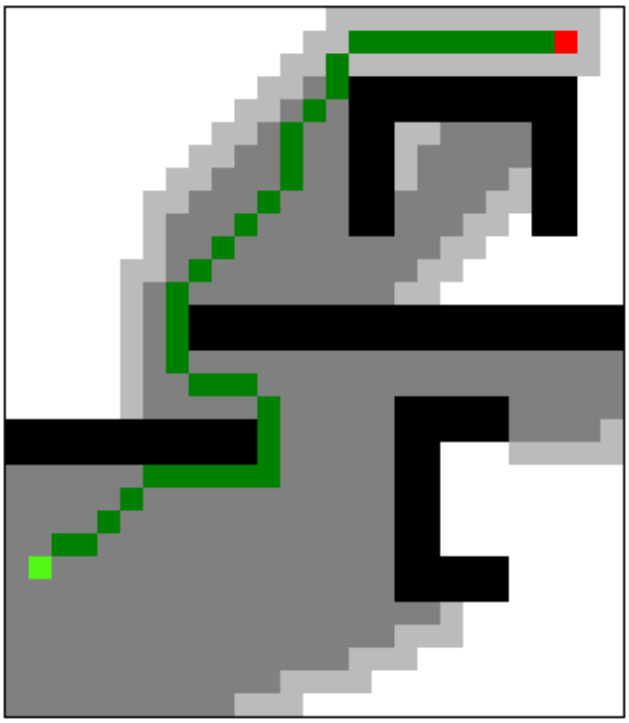


Рисунок 5.1 — Решение задачи поиска пути на ГРД при наличии препятствий между начальной и целевой позициями.

На рисунке показан ГРД с отмеченными начальной (зеленый цвет) и целевой (красный цвет) вершинами. Темно-зеленым цветом обозначен найденный алгоритмом эвристического поиска путь. Серым цветом показаны исследованные состояния поиска. Очевидно, что в ходе поиска рассматривались многие состояния, которые не имеют отношения к состояниям, входящим в искомый путь (или являющимися родительскими для них). Это происходит из-за того, что для многих вершин ГРД разница между оценочной стоимостью пути до целевой вершины, согласно эвристической функции, и реальной стоимостью пути, чрезмерно велика (т.к. эвристика никак не учитывает препятствия). Если бы эвристика более точно оценивала реальную стоимость пути для всех вершин ГРД, т.е. учитывала бы конкретное расположение препятствий относительно старта и финиша, то эффективность поиска была бы заметно выше. Так, известно, что если эвристика обладает абсолютной точностью, т.е. для каждой вершины её значение равно длине кратчайшего пути до цели, то поиск с такой эвристикой рассматривает только лишь состояния, образующие кратчайший путь (и никакие более). Следовательно, перспективной представляется задача автоматического построения эвристик более точно оценивающих стоимость достижения целевого состояния для каждого конкретного экземпляра задачи

поиска и разработки механизмов встраивания подобных синтетических эвристик в методы систематического поиска для повышения их вычислительной эффективности на практике. Именно подобным вопросам и посвящена данная глава работы.

В главе будет предложен оригинальный подход к синтезу эвристических функций с помощью современных методов и моделей нейросетевого машинного обучения (в частности будут использоваться архитектуры искусственных нейронных сетей с блоками внимания, т.н. трансформеры). Для обучения будут созданы репрезентативные коллекции данных, содержащие нетривиальные для решения задания. Будут описаны гибридные методы поиска, основанные на интеграции классических алгоритмов эвристического поиска и обучаемых эвристик (в частности метод, гарантирующий построение ограниченно-субоптимальных решений). Будет проведено обширное экспериментальное исследование, показывающее существенное превосходство разработанных методов над имеющимися мировыми аналогами.

5.1 Рассматриваемые задачи (PF-G, PF-RGB)

Задача PF-G Рассмотрим задачу поиска пути для некоторого мобильного агента, функционирующего в (плоской) среде с препятствиями, которая моделируется конечным 8-связным взвешенным ГРД $\mathcal{G} = (V, E, w)$, вложенным в метрическое пространство $\mathcal{M} = \mathbb{R}^2$ так, как это описано в Главе 1. То есть каждой вершине ГРД соответствует точка на плоскости. С каждым ребром графа $e = (u, v)$ будем ассоциировать отрезок AB, т.ч. $coord(u) = A$, $coord(v) = B$, где $coord : V \to \mathbb{R}^2$ – функция, сопоставляющая вершины графа точкам в $\mathbb{R}^2$ (функция вложения графа в пространство). Определим вес ребра как $w(e) = \|AB\|$.

Без ограничения общности будем считать, что горизонтально-смежные и вертикально-смежные вершины ГРД отстоят друг от друга на расстоянии 1, вес ортогональных ребер ГРД равен 1, а диагональных ребер – $\sqrt{2}$.

Будем полагать, что множество вершин ГРД разбито на два подмножества:

$$V = V^+ \cup V^-,$$
$$V^+ \cap V^- = \varnothing$$

где V^+ – множество проходимых (свободных, незаблокированных) вершин, V^- – множество непроходимых (занятых, заблокированных) вершин. Последним соответствуют те вершины, нахождение агента в которых недопустимо, например те, которые расположены внутри препятствий или в непосредственной близости от них.

Будем полагать, что на ребрах ГРД определена функция:

$$los: \ E \to \{true, false\},$$

которая определяет возможность перехода по ребру[1].

По умолчанию переход из/в заблокированную вершину невозможен. Т.е.:

$$source(e) \in V^- \ \vee target(e) \in V^- \implies los(e) = false.$$

Здесь, как и ранее в работе, запись $source(e)$ означает начальную вершину ребра e, а $target(e)$ – конечную. Т.е. для ребра $e = (v_1,v_2)$, $source(e) = v_1$, а $target(e) = v_2$.

Переход может быть невозможен и между смежными проходимыми вершинами. Например, в данной главе предполагается, что переходы запрещены между диагонально-смежными вершинами, если их общая ортогонально-смежная вершина непроходима – см. Рис. 5.2.

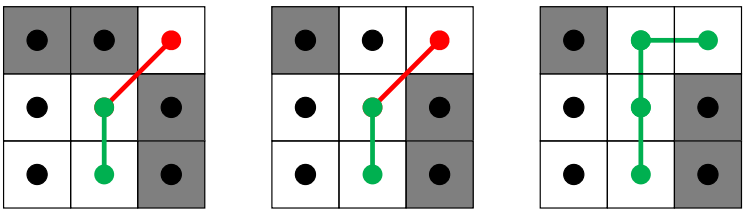


Рисунок 5.2 — Один из вариантов реализации функции los, определяющей допустимость переходов по ребрам 8-связного ГРД.

Определение 46. Путь на ГРД из фиксированной начальной вершины, v_{start}, в фиксированную конечную вершину, v_{goal}, – это последовательность смежных проходимых вершин, переход между которыми допускается функцией los:

[1]Заметим, что в отличие от предыдущих глав работы в данной главе возможность перехода между несмежными вершинами ГРД, т.е. вершинами не образующими ребро, не допускается.

$$\begin{aligned}
&\pi(v_{start}, v_{goal}) = v_0, v_1, ..., v_n, \\
&\text{т.ч.:} \\
&v_0 = v_{start}, \\
&v_n = v_{goal}, \\
&\forall i \in [0,n] : \ v_i \in V^+, \\
&\forall i \in [0,n-1] : \ los(v_i,v_{i+1}) = true.
\end{aligned} \tag{5.1}$$

Здесь и далее будем обозначать путь как π, опуская упоминание конкретных начальной и целевой вершин.

Определение 47. Стоимость пути на ГРД – это сумма весов ребер, входящих в путь:

$$\begin{aligned}
&cost(\pi) = \sum_{i=0}^{n-1} cost(v_i,v_{i+1}), \\
&cost(v_i,v_{i+1}) = w(e), \text{т.ч. } e = (v_i,v_{i+1}).
\end{aligned} \tag{5.2}$$

То есть стоимость пути эквивалентна его длине. Поэтому здесь и далее будем использовать оба этих термина.

Определение 48. Задача поиска пути на ГРД – это набор:

$$\text{PF-G} = (\mathcal{G},v_{start}, v_{goal}, los), \tag{5.3}$$

где $\mathcal{G} = (V,E)$ – заданный ГРД, $v_{start} \in V$, $v_{goal} \in V$ – зафиксированные начальная и целевая вершины соответственно, los – функция, определяющая возможность перехода по ребру.

Аббревиатура PF-G для обозначения введенной в рассмотрение задачи используется как сокращение следующего словосочетания на английском языке: **P**ath **F**inding on a **G**rid (дословно – поиск пути на графе регулярной декомпозиции). Необходимость явного упоминания ГРД возникает, т.к. в данной главе работы будет дополнительно рассматриваться модификация задачи поиска, в которой входными данными является изображение, а не граф в явном виде.

Решение задачи PF-G – это путь, соответствующий Определению 46.

Обозначим множество всех решений некоторой фиксированной задачи PF-G (т.е. множество путей) как Π. Решение назовём оптимальным, если соответствующий путь, $\pi^* \in \Pi$, является кратчайшим, т.е. обладает минимальной

стоимостью (среди всех возможных путей между заданными вершинами):

$$\pi^* = arg \min_{\pi \in \Pi}(cost(\pi)) \tag{5.4}$$

.

Аналогично, $\forall \pi \in \Pi : cost(\pi*) \leqslant cost(\pi)$.

Среди субоптимальных решений задачи PF-G, т.е. решений, не являющихся оптимальными, выделим отдельно класс т.н. ограниченно-субоптимальных решений, определенных относительно фиксированной константы $w > 1$.

Определение 49. Пусть $\pi \in \Pi$ – путь, являющийся решением некоторой задачи PF-G, а $w > 1$ – фиксированная константа. Тогда решение $\pi' \in \Pi$ называется ограниченно-субоптимальными, если $cost(\pi') \leqslant w \cdot cost(\pi^*)$.

Неформально, ограниченно-субоптимальное решение – это такой путь, длина которого не превышает длину кратчайшего пути, более чем в w раз. При этом саму константу w часто именуют (в рассматриваемом контексте) фактором субпотимальности.

Примеры решения задачи PF-G приведены на Рис. 5.3.

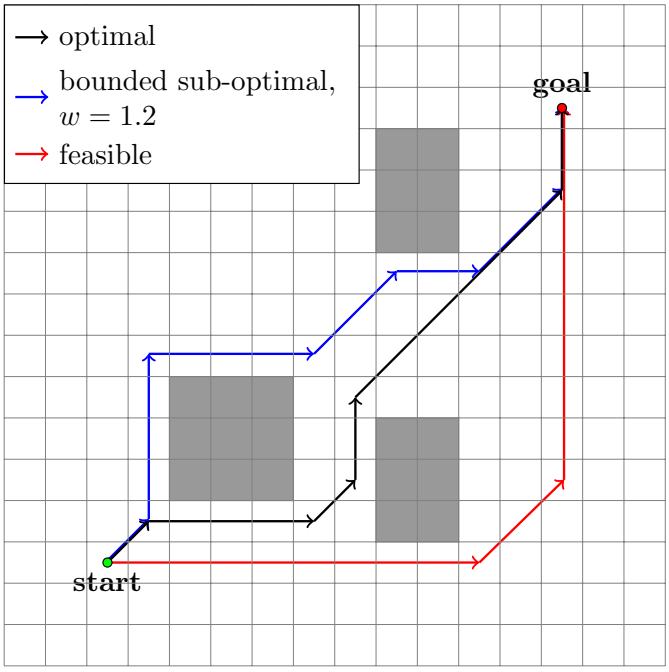


Рисунок 5.3 — Примеры решения задачи PF-G.

Черным цветом показан путь, соответствующий оптимальному решению задачи PF-G, красным – субоптимальному. Синим цветом показан путь, соответствующий ограниченно-субоптимальному решению задачи PF-G для фактора

субоптимальности $w = 1.2$, т.е. длина этого пути не превосходит длину кратчайшего более чем в 1.2 раза (на 20%).

В данной главе особое внимание будет уделяться задаче построения субоптимальных решений, в том числе – ограниченно-субоптимальных. Задача поиска оптимальных решений рассматриваться не будет.

Задача PF-RGB Помимо определенной выше задачи поиска пути на ГРД, имеющую явную связь с задачей планирования траектории для мобильного агента, перемещающегося в сложно-структурированной среде (более подробно об этом см. в Главе 1), рассмотрим следующую задачу.

Пусть некоторому мобильному роботу необходимо проложить маршрут перемещения (траекторию) по неровной местности, информация о которой содержится в многомерном массиве $\mathcal{MAP}$ размерности $m \times n \times 4$, представимом в виде последовательности 4-х матриц $m \times n$, именуемых каналами.

Первые три канала соответствуют цветовым каналам (красный, зеленый, синий) фотографического изображения местности (вид сверху). Будем именовать эти три канала в совокупности как RGB (стандартная аббревиатура для обозначения цветного изображения на английском языке: Red-Green-Blue).

Четвертый канал содержит данные о высотах в виде матрицы $m \times n$. Этот канал будем именовать как DEM, от английского Digital Elevation Model (цифровая модель рельфа).

Каждый элемент многомерного массива $\mathcal{MAP}$ может быть проиндексирован с помощью трех целочисленных значений и записан в виде (i,j,k) : $i \in [1,m]$; $j \in [1,n]$; $k \in [1,4]$. Элемент $(i,j,4)$ кодирует высоту некоторого участка местности, которому соответствует пиксель с индексом (i,j) на RGB-снимке этой местности. Будем обозначать его как $dem(i,j)$.

Сопоставим массиву $\mathcal{MAP}$ 8-связный граф регулярной декомпозиции $\mathcal{G}$ с шагом дискретизации 1, состоящий из $m \times n$ вершин, так что четырём элементам массива с индексами $(i, j, \cdot)$ соответствует единственная вершина ГРД v, т.ч. $coord(v) = (i,j)$.

Все вершины и ребра графа $\mathcal{G}$ будем считать *допустимыми*, т.е. возможно нахождение агента в любой из них и соответствующие переходы. При этом стоимость перехода по произвольному ребру $e = (u,v)$, или, что то же самое, вес ребра – $w(e)$, определяется следующим образом:

$$w(e) = cost(u, v) = dist(u, v) + \alpha \cdot |dem(u) - dem(v)| \tag{5.5}$$

где:

- $dist$ – расстояние между вершинами ГРД, т.е. $dist(u, v) = \|coord(u) - coord(v)\|_2$ (т.е. 1 для ортогонально-смежных вершин и $\sqrt{2}$ для диагонально-смежных вершин в рассматриваемом случае).
- $dem(v)$ – высота соответствующего элемента, т.е. $dem(v) \equiv dem(coord(v))$.
- α - параметр, задаваемый пользователем и регулирующий важность перепада высот, а также используемый для выравнивания масштаба при несовпадении пространственного и высотного разрешений.

Итак, в графе $\mathcal{G}$ возможны переходы между любыми ортогонально- и диагонально-смежными вершинами, но при этом стоимость перехода может существенно разниться. Интуитивно, переходы с резким изменением высоты имеют бо́льшую стоимость (регулируется параметром α), т.к. являются менее безопасными для мобильного робота, для которого требуется планирование траектории.

Путь на $\mathcal{G}$ между двумя заданными вершинами, v_{start} и v_{goal}, определяется как и ранее – см. Определение 46. Стоимость пути равна сумме стоимостей переходов его образующих: $cost(\pi) = \sum_{i=0}^{n-1} cost(v_i, v_{i+1})$.

Теперь определим следующую задачу поиска.

Определение 50. Задача поиска пути по многомерному массиву $\mathcal{MAP}$ – это набор:

$$\text{PF-MAP} = (\mathcal{MAP}, \mathcal{G}, i_{start}, j_{start}, i_{goal}, j_{goal}), \tag{5.6}$$

где $\mathcal{MAP} = (\text{RGB}, \text{DEM})$ – это заданный многомерный массив, содержащий информацию о местности (спутниковое изображение и карта высот); $\mathcal{G} = (V, E, w)$ – соответствующий взвешенный 8-связный ГРД, i_{start}, j_{start}, i_{goal}, i_{goal} – целочисленные координаты начального и целевого положения.

Решение задачи PF-MAP – это путь на ГРД $\mathcal{G}$ от вершины v_{start}, т.ч. $coord(v_{start}) = (i_{start}, j_{start})$, до вершины v_{goal}, т.ч. $coord(v_{goal}) = (i_{goal}, j_{goal})$. Путь минимальной стоимости есть оптимальное решение PF-MAP.

Пример задачи PF-MAP и её решения показан на Рис. 5.4.

На рисунке слева показан спутниковый снимок местности – первые три канала многомерного массива $\mathcal{MAP}$ (т.е. RGB каналы). На этом же снимке

Рисунок 5.4 — Пример задачи PF-MAP и её решения.

отмечены начальная и целевая позиции. По центру изображен DEM-канал, т.е. карта перепадов высот – чем темнее пиксель, тем больше высота. Справа на рисунке показано оптимальное решение задачи PF-MAP, т.е. путь наименьшей стоимости, которая учитывает перепады высот.

В данной главе будет рассматриваться модификация задачи PF-MAP, в которой часть входной информации, а именно канал DEM в массиве $\mathcal{MAP}$, не доступна для построения решения, но доступна для проверки качества этого решения. Иначе говоря, требуется построить путь, опираясь лишь на изображение (спутниковый снимок) местности, не имея доступа к информации о перепадах высот, которые определяют стоимость пути. Обозначим эту задачу как PF-RGB.

Предположение о выборке задач Выше были сформулированы две интересующие нас задачи поиска пути – PF-G и PF-RGB. Отметим, что если предъявлен отдельный экземпляр задачи PF-G, то есть требуется найти (кратчайший) путь на 8-связном ГРД, то для построения решения может быть использован один из известных алгоритмов эвристического поиска, например, алгоритм A* [11]. Однако, как было сказано в начале раздела, вычислительная эффективность подобного подхода обычно невысока из-за того, что известные функции, оценивающие расстояние между вершинами ГРД и используемые в качестве эвристик, не учитывают расположение препятствий (непроходимых вершин на ГРД). Также заметим, что если предъявлен для решения отдельный экземпляр задачи PF-RGB, то, с одной стороны, построить решение не представляет труда, т.к. по определению в данной задаче все переходы допустимы и можно выбрать в качестве решения любую последовательность вершин (пикселей на изображении), соединяющий заданные старт и финиш. С другой стороны, весьма вероятно, что построенное таким способом решение окажется низкого качества, т.е. стоимость пути будет высокой.

При этом на практике часто возникает необходимость решения большого числа схожих задач планирования, когда необходимо строить маршруты перемещения некоторого мобильного робота по местности определенного типа (например, местности городского типа, или же горной местности и др.). В таком случае можно предположить, что имеется некоторая *выборка* задач, часть из которых может быть использована для извлечения полезной информации, с целью последующего использования этой информации в том или ином виде для более эффективного решения других задач этого типа. Здесь и далее будем считать это предположение верным.

Когда речь о необходимости решения задач PF-G, будем считать, что доступна конечная выборка таких задач, при этом часть этой выборки может быть использована для обработки и анализа без каких-либо ограничений на вычислительные ресурсы. Оставшаяся часть используется для проверки эффективности разрабатываемого метода. Аналогично, когда речь о необходимости решения задач PF-RGB, будем предполагать, что доступна некоторая выборка задач, для которых известна информация о перепадах высот (т.е. по сути задач PF-MAP), и эти задачи могут использоваться для разработки метода, который должен успешно экстраполироваться на решение задач без такой информации.

В дальнейших разделах будет представлен общий подход к решению задач PF-G и PF-RGB при упомянутых предположениях. Этот подход опирается на современные нейросетевые модели машинного обучения, поэтому по традиции, принятой в этой области, будем называть выборку задач, доступных для предварительной обработки, обучающей выборкой, а выборку задач, предназначенных для проверки эффективности разработанного метода, – тестовой выборкой.

5.2 Методы и алгоритмы

5.2.1 Систематический поиск с использованием обучаемых эвристик

Сосредоточимся на способах решения задач PF-G, т.е. задач поиска пути на взвешенном ГРД, а случай, когда требуется искать путь по изображениям, т.е. решать задачи PF-RGB, рассмотрим позже.

Одним из наиболее распространенных способов решения отдельной задачи PF-G является систематический эвристический поиск с использованием эвристической функции, оценивающей стоимость пути от произвольной вершины ГРД до целевой вершины, – алгоритм A* [11] и его вариации. Предлагаемое в работе решение основано на этом подходе, поэтому приведем краткое описание базового алгоритма (и его часто используемых вариантов) и затем опишем предлагаемый подход. Более подробно принципы работы и свойства алгоритмов поиска, использующих эвристические функции, были изложены ранее в Главе 1.

Алгоритм A* и его модификации При поиске пути на ГРД алгоритм A* исследует пространство состояний, каждое из которых соответствует вершине графа и содержит в себе дополнительные данные, необходимые для организации поиска. Эти данные включают g-значение состояния, т.е. вычисленную к текущей итерации алгоритма стоимость пути от начального состояния до текущего, h-значение – эвристическую оценку стоимости пути от текущего состояния до целевого. Сумма g- и h-значений называется f-значением состояния. Фактически, f-значение состояния n есть оценка стоимости пути, проходящего через n. Поскольку для задачи PF-G состояние поиска взаимно-однозначно определяется вершиной графа, в этой главе при описании алгоритмов будем использовать термины *состояние поиска* и *вершина графа* как синонимы. Например, говоря о пути, проходящем через состояние n, будем иметь ввиду путь проходящий через вершину ГРД, определяющую состояние n.

Процесс исследования пространства состояний заключается в итерационном построении дерева частичных решений – дерева поиска, – с корнем

в начальном состоянии. Каждый узел дерева соответствует пути от начального состояния до текущего и относится либо к состояниям-кандидатам на дальнейшее рассмотрение, т.н. список OPEN, либо к уже обработанным состояниям, т.е. список CLOSED. На каждой итерации из списка OPEN выбирается (и удаляется из этого списка) состояние с минимальным f-значением, т.е. $n = arg\min_{n \in OPEN} f(n)$. Затем происходит т.н. *раскрытие* состояния n, т.е. генерация состояний-потомков, соответствующих допустимым переходам на ГРД, и добавление части из них в дерево поиска. Более конкретно, для каждого состояния потомка, n', рассчитывается g-значение как сумма g-значения раскрываемого состояния и стоимости перехода (веса ребра): $g(n') = g(n) + cost(n,n')$. Далее, если в дереве поиска отсутствует состояние n', то оно добавляется в него и помечается как кандидат на дальнейшее рассмотрение (т.е. добавляется в список OPEN). Если же состояние уже имеется в дереве поиска (что означает, что уже известен какой-то путь до данной вершины), то происходит сравнение вновь рассчитанного и хранимого g-значений. Если первое больше второго (новый путь хуже, чем старый), то потомок отбрасывается. Если же первое меньше второго (новый путь лучше), то происходит обновление состояния в дереве поиска (запоминается более короткий путь). После того, как все потомки состояния n были созданы и дерево поиска соответствующим образом обновлено, само состояние n добавляется в список CLOSED. Алгоритм завершается при извлечении целевого состояния из OPEN (в этом случае искомый путь может быть восстановлен с помощью родительских указателей), либо если список OPEN становится пустым (в этом случае алгоритм возвращает специальный символ *failure*, сигнализирующий о невозможности построить путь).

Несколько вариантов псевдокода алгоритма A* уже приводилось ранее в работе в Разделе 1.3, поэтому здесь опустим псевдокод. Пример дерева поиска алгоритма A* показан на Рис. 5.5

На рисунке начальное состояние (корень дерева поиска) выделено желтым. Светло-зеленым цветом показаны состояния списка CLOSED, т.е. раскрытые состояния, синим цветом показаны состояния списка OPEN – эти состояния являются листьями дерева поиска (показано зеленым цветом) и кандидатами на дальнейшее рассмотрение (раскрытие).

Эффективность алгоритма A* (количество итераций и гарантии стоимости найденного пути) существенно зависит от используемой эвристической функции h, которая направляет поиск к цели. Так эвристика называется *идеальной*

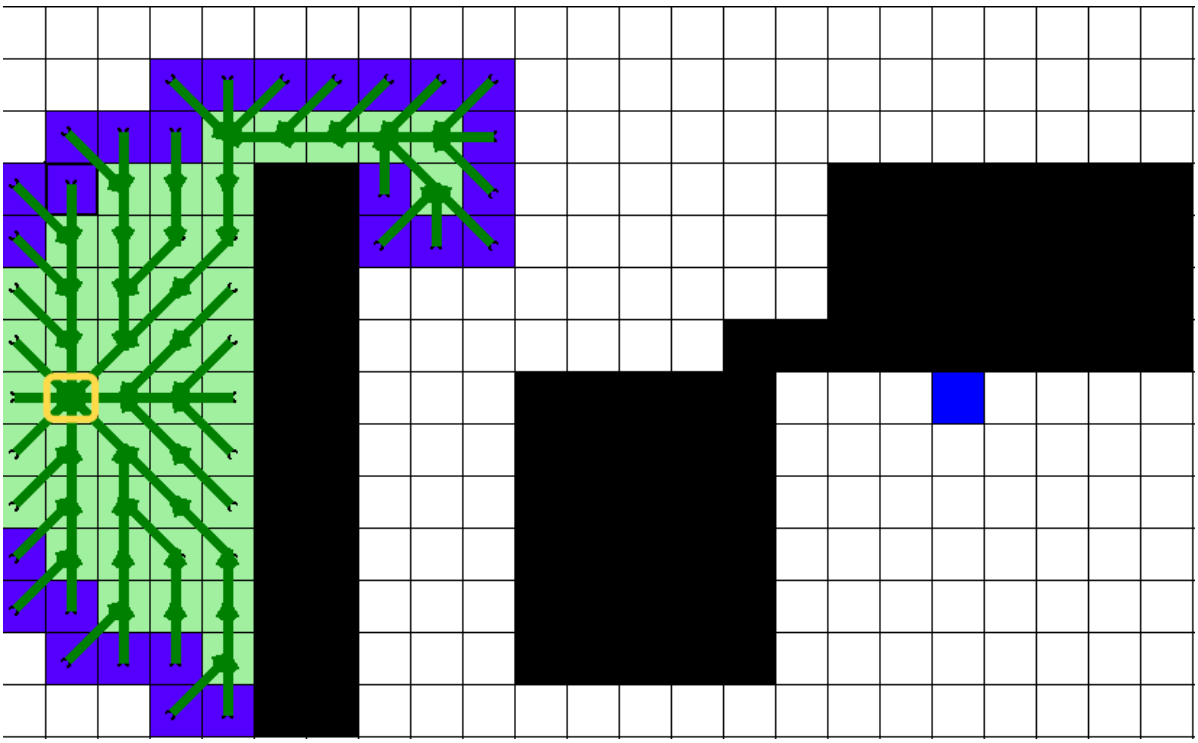


Рисунок 5.5 — Дерево поиска алгоритма A*.

(абсолютно точной) и обозначается как h^*, если для любого состояния её значение равно истинной стоимости достижения цели: $h^*(n) = cost(\pi^*(n, n_{goal}))$. A* с идеальной эвристикой, естественно, гарантирует построение оптимальных решений, более того – в ходе работы алгоритма раскрываются лишь состояния, принадлежащие одному из кратчайших путей на ГРД. То есть алгоритм облает максимально возможной вычислительной эффективностью. К сожалению, идеальная эвристика зависит от конкретного экземпляра задачи и на практике недоступна.

Эвристика называется *допустимой*, если она никогда не переоценивает истинную стоимость достижения целевого состояния: $h(n) \leqslant h^*(n)$, и *согласованной* (монотонной), если $\forall n, n' : h(n) \leqslant h(n') + cost(\pi^*(n, n'))$.

Для 8-связных ГРД известны различные универсальные (т.е. не зависящие от конкретного экземпляра задачи) монотонные и допустимые эвристические функции, оценивающее стоимость пути от текущего состояния до целевого: расстояние Чебышёва, Евклидово расстояние и др. Здесь и далее будем подразумевать, что эвристическая функция определена как:

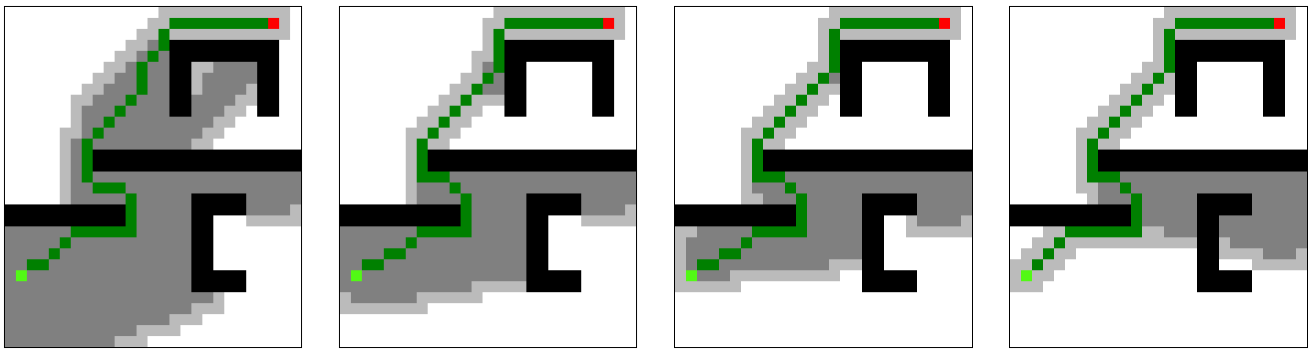
$$h_{octile} = \sqrt{2} \cdot min(\Delta_x, \Delta_y) + 1 \cdot |\Delta_x - \Delta_y|, \tag{5.7}$$

где $\sqrt{2}$ и 1 – стоимости перехода между диагонально- и ортогонально-смежными вершинами (в общем случае могут быть произвольными), а Δ_x, Δ_y –

абсолютные разности координат вершин. Эта эвристическая функция является монотонной и допустимой.

Известно, что алгоритм A* с допустимой эвристикой гарантированно находит оптимальное решение. При этом, если эвристика является монотонной, как в рассматриваемом случае, то гарантируется, что в ходе построения дерева поиска никогда не может быть найден более короткий путь к любому из уже рассмотренных (раскрытых) состояний. Это обстоятельство несколько упрощает реализацию алгоритма (в части обновления дерева поиска). Тем не менее, поскольку приведенная выше эвристическая функция h_{octile}, является универсальной, т.е. никак не учитывает особенности конкретного экземпляра задачи, в частности – расположение заблокированных вершин на ГРД, на практике ее использование приводит к чрезмерному числу итераций алгоритма, что негативно сказывается на временной и емкостной эффективности.

Одним из распространенных способов повышения вычислительной эффективности алгоритма за счет отказа от поиска оптимальных решений является использование *взвешенной эвристики*. При этом подходе для расчета f-значения состояния используется не исходная эвристика h, а её модификация $\hat{h} = w \cdot h$, где $w \geqslant 1$ – входной параметр алгоритма, именуемый весом эвристики. То есть f-значение состояния n рассчитывается как: $f(n) = g(n) + \hat{h}(n) = g(n) + w \cdot h(n)$. Такая модификация A*, обычно называемая WA* (от англ. Weighted A*), гарантирует отыскание ограниченно-субоптимального решения (см. Определение 49). Эффект взвешивания эвристики показан на рис. 5.6.



а) A*, (WA*, $w = 1$) б) WA*, $w = 1.5$ в) WA*, $w = 2$ г) WA*, $w = 10$

Рисунок 5.6 — Эффект от применения техники взвешивания эвристической функции.

Как видно на рисунке, при увеличении веса поиск становится более сфокусированным на цели, что уменьшает количество исследуемых состояний. Однако эффект от взвешивания эвристики может быть ограничен, а именно –

число исследуемых состояний может увеличиваться, а не уменьшаться, с увеличением веса после определенного порога, как показано на правой части рисунка, при $w = 10$. Таким образом, эта техника имеет свои ограничения.

Другим способом сокращения пространства состояний является фокусировка поиска за счет введения в рассмотрение дополнительного списка состояний-кандидатов, FOCAL⊆OPEN, и выбора из этого списка состояния с помощью дополнительной эвристической функции h_{FOCAL}. Алгоритм, использующий эту идею, был впервые предложен в [12] и часто обозначается как `Focal Search` (`FS`). В `FS` в список FOCAL, обновляемй каждой итерации, помещаются те состояния из OPEN, чьи f-значения не превышают минимальное f-значение в OPEN, f_{min}, более чем в w раз, где $w \geqslant 1$ – задаваемый пользователем параметр. Формально: $\text{FOCAL} = \{n \mid n \in \text{OPEN}, f(n) \leqslant w \cdot f_{min}\}$.

Список FOCAL упорядочивается согласно вторичной эвристике h_{FOCAL}, к которой не предъявляется никаких требований, т.е. она не обязательно должна быть монотонной или даже допустимой. Очередной состояние для раскрытия теперь выбирается из FOCAL в соответствии с порядком, задаваемым h_{FOCAL}, т.е. $n = arg\min_{n \in \text{FOCAL}} h_{FOCAL}(n)$. После выбора и удаления этого состояния из FOCAL (и из OPEN), его обработка происходит так же как и ранее – создаются состояния-потомки, обновляется дерево поиска и пр. Критерий остановки аналогичен `A`*. Доказано, что `FS` гарантирует отыскание ограниченно-субоптимальных решений. Количество итераций поиска и, следовательно, вычислительная эффективность `FS` существенно зависят от выбора h_{FOCAL}. К сожалению, на практике для стандартных задач поиска, таких как рассматриваемая в главе задача PF-G, не известно общего способа эффективного задания h_{FOCAL}.

Поиск с обучаемыми эвристическими функциями Основная идея предлагаемого подхода, заключается в использовании одного из описанного выше алгоритмов эвристического поиска и применения в нём дополнительной эвристической функции, которая в отличие от стандартной эвристики h_{octile} (5.7), учитывает особенности конкретного экземпляра задачи и позволяет эффективно фокусировать поиск с учетом этих особенностей (форма препятствий, их взаимное расположение и т.д.). Эта дополнительная эвристическая функции аппроксимируются искусственной нейронной сетью и конструируется, т.е.

обучается, за счет обработки доступной выборки задач (обучающей выборки). После такой обработки (обучения), отдельные экземпляры задач PF-G из отложенной (тестовой) выборки решаются следующим образом. Сначала по входным данным (ГРД, расположение начальной и целевой вершины) нейросеть предсказывает дополнительную эвристику, которая затем передается на вход алгоритму поиска, её использующую, и уже алгоритм поиска решает задачу. Такой подход отличается концептуальной простотой и простотой реализации и позволяет гарантировать *полноту*, т.е. тот факт, что для любой решаемой задачи будет найдено корректное решение, а если задача решения не имеет, то алгоритм корректно завершится. Более того при определенном варианте комбинирования обучаемой эвристики и алгоритма поиска можно гарантировать и ограниченную субоптимальность решения, т.е. то, что стоимость найденного решения не будет превосходить стоимость оптимального решения, более чем в любое, наперёд-заданное число раз. Подобных гарантий не получится добиться, если полагаться только на методы машинного обучения, на т.н. сквозные (в англоязычной терминологии – end-to-end) планировщики.

Предлагается аппроксимировать (обучать) два типа эвристических функций, различающихся концептуально, и подходящих для комбинирования с различными алгоритмами эвристического поиска.

Первый тип эвристики – коэффициент (фактор) коррекции, который обозначим как cf (от англ. correction factor) и определим как отношение значения исходно-заданной (универсальной) эвристики, h, к значению идеальной эвристики, h^*:

$$cf(n) = \frac{h(n)}{h^*(n)} \tag{5.8}$$

При решении задачи PF-G предлагается использовать эту эвристику в алгоритме $\texttt{WA}^*$ в качестве индивидуального веса для для каждого отдельного состояния поиска, т.е. f-значение состояния n предлагается вычислять не по формуле $f(n) = g(n) + w \cdot h(n)$, где $w > 1$ – константа, не зависящая от n, а используя индивидуальный вес $w(n) = 1/cf(n)$:

$$\begin{aligned} f(n) &= g(n) + w(n) \cdot h(n) \\ &= g(n) + \frac{1}{cf(n)} \cdot h(n) \\ &= g(n) + h(n)/cf(n). \end{aligned} \tag{5.9}$$

Очевидно, что чем точнее аппроксимировано cf-значение, тем ближе второе слагаемое к точной оценке длины пути до цели. То есть аппроксимация cf-значений концептуально схожа с предсказанием значения идеальной эвристики. Заметим однако, что диапазон значений, который может принимать поправочный коэффициент для любой задачи PF-G, есть по определению интервал [0,1], и такой диапазон естественным образом подходит для обучения современных нейросетей, т.к. они архитектурно устроены так, что их выходные числовые значения обычно содержатся в диапазоне $[-1,1]$ или [0,1]. В то же время диапазон значений идеальной эвристики в общем случае не ограничен сверху, вернее – ограничен размерами входного ГРД, т.е. зависит от конкретного экземпляра задачи PF-G. И для аппроксимации идеальной эвристики с помощью нейросетвых моделей на практике необходимо вводить дополнительный нормирующий коэффициент, который зависит от размера входа. Второе отличие cf-значения от h^*-значение, состоит в том, что поправочный коэффициент содержит больше эвристической информации, так как является по сути комбинацией универсальной эвристики $h(n)$, не учитывающей никакие особенности конкретного экземпляра задачи, и идеальной эвристики $h^*(n)$, учитывающей эти особенности. Последующие эксперименты, проведенные в работе, подтверждают, что обучение cf-значений вместо h^*-значений дает заметный значительный прирост в эффективности.

Разница между универсальной эвристикой, идеальной эвристикой и предлагаемой к обучению (фактора коррекции) эвристикой – фактором коррекции, показана на Рис. 5.7

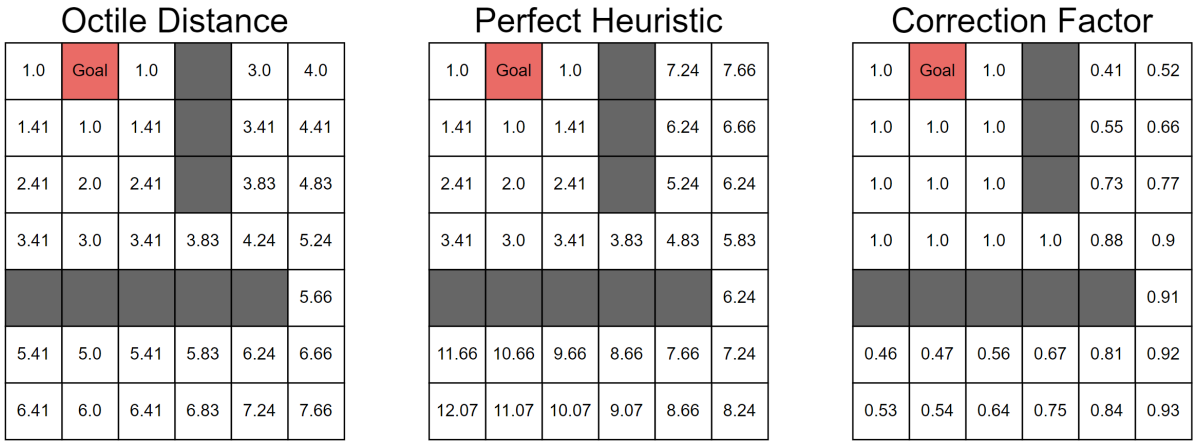


Рисунок 5.7 — Различные типы эвристик для решения задачи поиска пути на ГРД: универсальная эвристика (слева), идеальная эвристика (по центру) и фактор коррекции (справа).

Как видно из рисунка для вершин ГРД, расположенных в непосредственной близости от целевой (отмеченной красный цветом), значения универсальной эвристики совпадают со значениями идеальной эвристики, поэтому фактор коррекции для таких вершин равняется 1. С другой стороны для вершин расположенных на удалении от цели, за препятствиями – например, для левой нижней вершины, – значения универсальной и идельной эвристики отличаются почти в 2 раза – 6.41 против 12.07. Поэтому и фактор коррекции для подобных вершин меньше 1, например, он составляет 0.53 для левой нижней вершины.

Второй тип эвристики, предлагаемый к использованию, представляет собой вероятность вхождения вершины ГРД в итоговый путь. Будем ссылаться на эту эвристику как на *pp*-значение (от англ. path probability), а на совокупность всех таких значений для ГРД как на карту вероятностей вхождения в путь, PPM (от англ. path probability map). По определению *pp*-значения состояний заключены в том же интервале, что и cf-значения, т.е. $pp(n) \in [0,1]$ для любого состояния поиска n. Пример PPM представлен на Рис. 5.8.

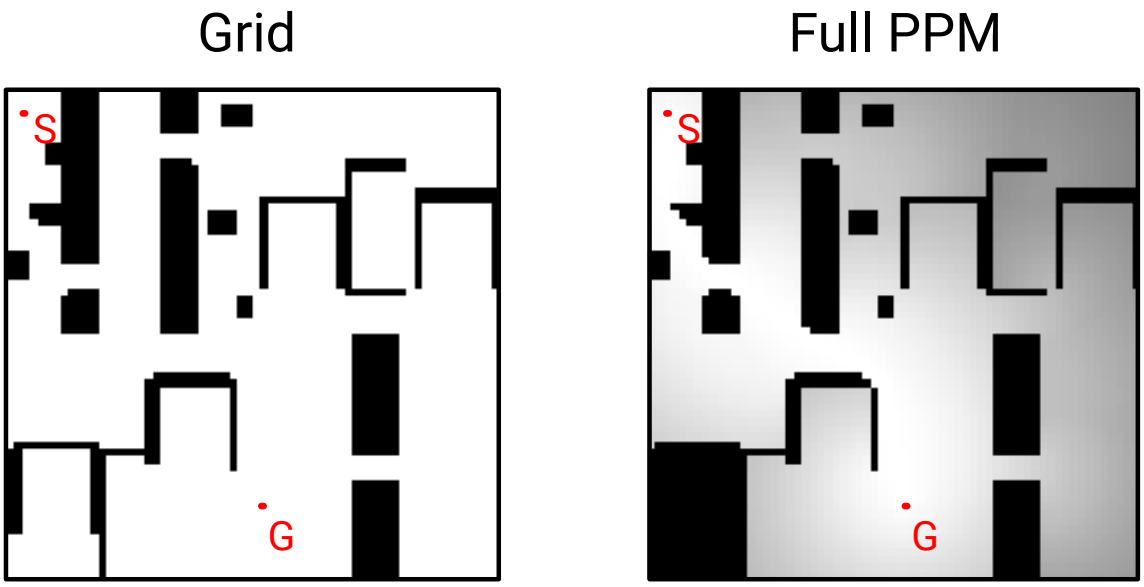


Рисунок 5.8 — Пример карты вероятностей вхождения в путь (PPM).

На рисунке слева показан исходный ГРД, на котором отмечены начальная и целевая вершины, справа – карта вероятностей вхождения вершин в путь, т.е. PPM. Чем светлее пиксель, тем ближе *pp*-значение соответствующей вершины ГРД к 1, т.е. светлые вершины с б*о*льшей вероятностью входят в (кратчайший) путь на ГРД.

При наличии PPM, её можно естественным образом комбинировать с алгоритмом эвристического поиска `FS`, а именно – использовать *pp*-значения в

качестве вторичной эвристики: $h_{FOCAL}(n) = pp(n)$. Интуитивно эта эвристика призвана на каждом шаге работы алгоритма определять какое из состояний поиска с б*о*льшей вероятностью входит в искомый путь и, следовательно, должно быть рассмотрено на текущей итерации.

Точное предсказание *pp*-значений можно рассматривать как попытку непосредственного решения задачи поиска пути. Действительно, если для произвольной задачи PF-G получить PPM, в которой *pp*-значения равны 1 для вершин, лежащих на единственном кратчайшем пути, и 0 – для остальных вершин, то для решения задачи алгоритм поиска не требуется. Достаточно на каждой итерации выбирать из вершин, смежных с текущей ту, для которой $pp = 1$ (и не выбранную ранее). Однако, очевидно, что на практике точное предсказание PPM с помощью нейронной сети невозможно, именно поэтому предлагается использовать предсказанные *pp*-значения в качестве вторичной эвристики в алгоритме `FS`.

Псевдокод На Рис. 5.9 представлен универсальный псевдокод алгоритмов поиска, используемых в данной главе. Различные цвета соответствуют различным модификациям поиска, как будет разъяснено дальше. Предполагается, что все необходимые эвристические функции передаются на вход алгоритму: h – универсальная эвристика (стандартная эвристика для решения задачи PF-G), h_{FOCAL} – вторичная эвристика для алгоритма `FS`, cf – фактор коррекции для предлагаемой модификации алгоритма `WA*`, использующей индивидуальный вес для каждого состояния поиска. Последние две эвристики предварительно конструируются из данных (обучаются по имеющейся выборке заданий PF-G) и передаются на вход алгоритма для поиска решения задачи PF-G.

На рисунке черным цветом показаны строки, выполняемыми всеми из трех рассматриваемых алгоритмов, `A*`, `WA*`, `FS`. Эти строки отвечают за базовую логику исследования пространства состояний с использованием эвристической функции (т.е. это алгоритм `A*`). Для алгоритма `WA*`, при расчете f-значения состояния используется модифицированная формула, в которой эвристика домножается на заданный коэффициент $w > 1$. В Строке 15 это отмечено красным цветом. Аналогично в предлагаемой авторской модификации алгоритма `WA*`, опирающейся на выученный фактор коррекции, домножение производится на коэффициент $1/cf(n)$ – эта модификация отмечена синим цветом в Строке 15. Наконец алгоритм `FS`, как было описано выше, оперирует дополнительным

```
1  Алгоритм USearch(𝒢, v_start, v_goal, h, w, h_FOCAL, cf):
       Входные данные: ГРД 𝒢, начальная и целевая вершины v_start,
                       v_goal, эвристическая функция h, фактор
                       субоптимальности w, дополнительная
                       эвристическая функция h_FOCAL, фактор
                       коррекции cf
       Выходные данные: путь π
2      n_start ← GenerateSearchNode(v_start)
3      g(n_start) ← 0; parent(n_start) ← null
4      OPEN ← {n_start}; CLOSED ← ∅
5      while OPEN ≠ ∅ do
6          n ← GetBestNode(OPEN, FOCAL, h_FOCAL)
7          OPEN ← OPEN \ {n};  FOCAL ← FOCAL \ {n}
8          CLOSED ← CLOSED ∪ {n}
9          UpdateFocal()
10         if n.v = v.goal then
11             return ReconstructPath(n)
12         foreach n′ ∈ GetSuccessors(n) do
13             if g(n′) > g(n) + cost(n,n′) then
14                 g(n′) ← g(n) + cost(n,n′)
15                 f(n′) ← g(n′)+w·h(n′)/cf(n′)
16                 updateOPEN(n′)
17                 if f(n′) ⩽ w · f_min then
18                     updateFOCAL(n′)
19     return failure
```

Рисунок 5.9 — Универсальный псевдокод алгоритмов A*, WA*, FS. Различные цвета соответствуют различным модификациям.

списком FOCAL, в который попадают все вершины f-значение которых, не превышает $w \cdot f_{min}$ (см. Строку 17), где f_{min} – это минимальное f-значение по всем вершинам из списка OPEN. Изменения, вызванные необходимостью работы с этим списком, показаны коричневым цветом.

Отметим, что алгоритм FS (в отличие от WA*) допускает использование значения фактора субоптимальности $w = \infty$. В этом случае алгоритм выполняется корректно, при этом в список FOCAL попадают все те же состояния, что и в список OPEN, т.е. эти списки для $w = \infty$ идентичны. Выбор наилучшего состояния на очередной итерации поиска производится из всех листьев дерева поиска (из списка OPEN), но не по f-значению, как в алгоритме A*/WA*, а по h_{FOCAL}-значению, т.е. выбирается состояние с минимальным значением вторичной эвристики, h_{FOCAL}. На такую версию алгоритма FS будем ссылаться как на алгоритм GBFS, от англ. Greedy Best-First Search. По сути это жадный поиск с возвратами по эвристике h_{FOCAL}.

Свойства алгоритмов Выше были введены в рассмотрение 5 различных алгоритмов эвристического поиска, часть из которых использует дополнительные входные параметры, которые могут задаваться в т.ч. нейросетевыми моделями:

- A* – классический алгоритм систематического поиска, использующий универсальную (т.е. не зависящую от конкретного экземпляра задачи) эвристику, оценивающую стоимость пути от произвольного состояния до цели.
- WA* – модификация алгоритма A*, отличающаяся использованием взвешенной (т.е. домноженной на заданный пользователем коэффициент w) эвристики.
- WA*+CF – модификация алгоритма WA*, в которой для каждого состояния используются индивидуальный коэффициент взвешивания равный $1/cf(n)$, где $cf(n)$ – это аппроксимированное значение фактора коррекции (см. 5.8), передаваемое в качестве входного параметра алгоритма.
- FS – алгоритм поиска, использующий дополнительную эвристику h_{FOCAL} для организации поиска. Дополнительно этот алгоритм подразумевает задание фактора субоптимальности $w > 1$, по аналогии с WA*.
- GBFS – алгоритм поиска, использующий лишь эвристику h_{FOCAL} для фокусировки поиска. Технически этот алгоритм – есть модификация FS, при $w = \infty$

Обозначим последние два алгоритма, использующие в качестве дополнительной эвристики h_{FOCAL} карту вероятностей вхождения в путь как FS+PPM и

GBFS+PPM, соответственно, и приведем несколько очевидных утверждений касающихся теоретических гарантий, которые предоставляют приведенные выше алгоритмы[2].

Утверждение 9. Все *описанные алгоритмы являются полными, т.е. гарантируют отыскание решения задачи, если оно существует, и если решения нет, корректно завершаются, вернув специальный символ* failure.

Это утверждение справедливо, т.к. рассмотренные алгоритмы являются модификациями полных алгоритмов A*, WA*, FS, полнота которых не зависит от используемых дополнительных входных параметров – веса эвристики w и вторичной эвристики h_{FOCAL}. Даже если значение этих параметров установлено как выход произвольных искусственных нейронных сетей, как в алгоритмах WA*+CF, FS+PPM, GBFS+PPM, то это никак не сказывается на гарантиях корректности завершения и отыскания решения.

Утверждение 10. *Алгоритм* FS+PPM *гарантирует отыскание ограниченно-субоптимального решения, т.е. пути, длина которого не превышает длину кратчайшего пути более, чем в w раз.*

Это утверждение справедливо, т.к. известно, что стоимость решения, конструируемого алгоритмом FS, не превышает указанный порог для любой вторичной эвристики h_{FOCAL} [12]. Пусть даже эта эвристическая функция выражается нейросетью.

О способах решения задачи PF-RGB Для решения задачи PF-RGB, т.е. задач поиска пути по изображению рельефа местности, предлагается подход, который аналогичен описанному ранее. Этот подход основан на использовании глубоких нейронных сетей, которые принимают на вход RGB-изображение и формируют информативный выход для последующего применения в классических алгоритмах эвристического поиска.

Предлагается использовать два варианта выхода сети:

1. Основной вариант: восстановление PPM (карты вероятностей пути) с последующим использованием в алгоритме GBFS. Как и ранее обозначим этот вариант как GBFS+PPM.

[2]Утверждения приводятся без формальных доказательств, т.к. они очевидны.

2. Альтернативный вариант: восстановление цифровой модели рельефа (DEM) по изображению, с последующим запуском алгоритма A*. Обозначим этот вариант как A*+DEM.

Оба метода не гарантируют оптимальность или ограниченную субоптимальность результирующих путей, так как в задаче PF-RGB допустимы переходы между любыми смежными состояниями (пикселями на карте), но неизвестна их стоимость, т.к. цифровая модель местности (DEM), которая и определяет эту стоимость (см. (5.1)), не передается в качестве входа. Поэтому использование FS с фиксированной границей субоптимальности нецелесообразно, и в данной работе предлагается ограничиться GBFS.

Отметим, что для решения задач PF-RGB не предлагается использовать алгоритм WA*+CF, в отличие от задач PF-G. Это объясняется тем, что фактор коррекции для задачи PF-RGB малоинфомативен из-за того, что стандартная эвристика, используемая для поиска пути на ГРД, h_{octile} (см. 5.7), не учитывает перепады высот.

5.2.2 Аппроксимация эвристических функций с помощью методов машинного обучения

Выше были описаны две оригинальные эвристические функции: cf – фактор коррекции, pp – вероятность вхождения в путь, и предложен подход к их интеграции с алгоритмами эвристического поиска. Опишем далее способ автоматического конструирования (обучения) предлагаемых эвристик по имеющейся выборке задач (обучающей выборке). Как и ранее, сначала опишем метод в контексте решения задачи PF-G, а затем дополним его описанием модификаций, используемых в контексте решения задач PF-RGB.

Итак, предлагается аппроксимировать эвристические функции pp и cf многослойной нейронной сетью, на вход которой подается закодированная информация о задаче PF-G, а выходом которой является матрица pp- или cf-значений, которые обозначим как PPM и CFM (суффикс M – от англ. Map (карта)).

Для настройки весовых коэффициентов нейросети используется подход обучения с учителем. Такой подход подразумевает первичное формирование

эталонных эвристик для каждой из задач обучающей выборки, с которым впоследствии происходит сравнение ответа нейросети для настройки (обучения) её параметров с помощью стандартного метода обратного распространения ошибки.

Опишем далее способ конструирования эталонных значений более подробно.

Построение эталонных эвристик Формирование эталонных cf-значений для произвольной задачи PF-G сводится к расчету для каждой вершины ГРД, v, стоимости кратчайшего пути из неё до целевой вершины v_{goal} – $cost(\pi^*(v,v_{goal}))$. Это значение и есть по определению значение идеальной эвристики для вершины v для данного экземпляра задачи PF-G – $h^*(v)$[3]. Затем для этой же вершины вычисляется по формуле 5.7 значение стандартной эвристики $h(v)$ и, наконец, вычисляется фактор коррекции: $cf(v) = h(v)/h^*(v)$.

Для расчета $h^*(v)$ на практике удобно пользоваться техникой обратного распространения, когда из целевой вершины запускается алгоритм неинформированного систематического поиска (алгоритм Дейкстры) с критерием остановки “до исчерпания не рассмотренных вершин” (до исчерпания списка OPEN). После его завершения для каждой вершины v будет известна стоимость кратчайшего пути из v_{goal}, которая для ГРД, очевидно, равна стоимости кратчайшего пути из v в v_{goal}, т.е. $h^*(v)$.

Построение эталонных карт вероятностей вхождения в путь, PPM, производится менее наивным способом. Предполагается, что значения элементов эталонной PPM должны быть равны 1 для вершин, лежащих на кратчайшем пути, тогда как все остальные вершины должны иметь меньшие значения. При этом на 8-связных ГРД может существовать множество кратчайших путей, отличающихся только порядком ортогональных/диагональных перемещений – см. Рис. 5.10. Возникает вопрос, каким именно вершинам, лежащим на кратчайших путях присваивать pp-значение равным единице.

Наиболее простой ответ заключается в том, что для любой вершины, принадлежащей хотя бы одному из кратчайших путей, устанавливать $pp = 1$.

[3]Заметим, что h^*-значение не зависит от начальной вершины поиска, что естественно, т.к. эвристика оценивает стоимость пути от произвольной вершины до цели, без привязки к стартовой позиции.

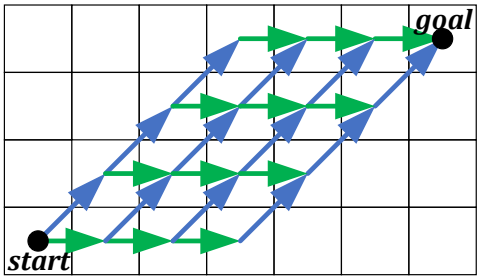


Рисунок 5.10 — Пример симметрии путей на ГРД.

Однако, использование построенных таким способом PPM, ведет к нежелательным эффектам при последующем поиске пути – см. Рис. 5.11.

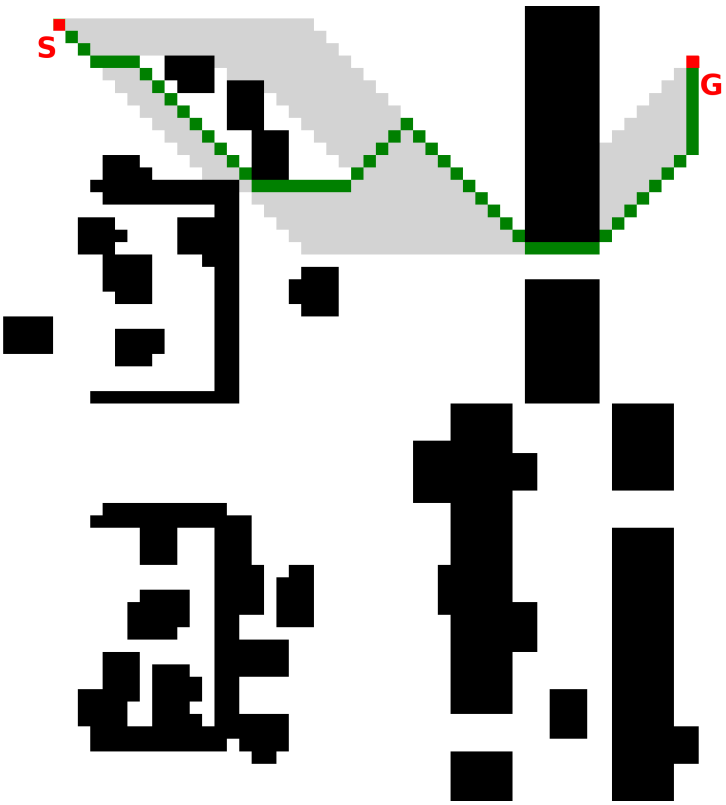


Рисунок 5.11 — Пример пути на ГРД, найденного с помощью эталонной карты вероятностей (PPM), при построении которой не учитывалась симметрия путей.

На рисунке изображен ГРД и путь (отмечен зеленым), найдённый алгоритмом `FS+PPM` (с $w = 2$), в котором использовалась PPM, построенная по указанному выше принципу: для вершин v, т.ч. $v \in \pi^* : (v_{start}, v_{goal})$, $pp(v) = 1$, для всех остальных – $pp(v) = 0$. Вершины, pp-значение которых равно 1, отмечены на рисунке серым цветом.

Акцентируем внимание на том, что эта PPM является *эталонной* в том смысле, что это не выход нейросети, а алгоритмически построенная PPM, которая впоследствии будет выступать эталоном при обучении нейросети, т.е.

последняя будет обучаться воспроизводить эту PPM. При этом на рисунке видно, что путь построенный алгоритмом `FS+PPM` по такой эталонной PPM имеет странную форму и не является кратчайшим. Следовательно, даже если нейросеть будет обладать 100% точностью (что на практике невозможно), то использование генерируемых ей PPM будет приводить к построению путей чрезмерно высокой стоимости.

Эффект, наблюдаемый на Рис. 5.11 можно объяснить следующим образом. При построении пути с помощью алгоритма `FS+PPM` на каждой итерации основного цикла при рассмотрении очередного состояния поиска (вершины ГРД), среди его потомков (смежных вершин, не рассмотренных ранее) всегда будет *несколько* состояний с *pp*-значением равным 1 (из-за множества симметричных путей между стартом и финишем). Выбор состояния для дальнейшего рассмотрения (и вхождения в путь) будет происходить с помощью вторичного критерия сортировки[4].

Наиболее интуитивным критерием является правило предпочтения состояния с меньшим h-значением, т.е. того, что расположено ближе к цели (подобный вторичный критерий наиболее часто используется на практике в различных задачах поиска и планирования – см. [86]). С использованием этого правила алгоритм `FS+PPM` будет выбирать состояния один за одним на каждой итерации поиска и достигнет целевого состояния, не совершив ни одного лишнего раскрытия. при этом для рассматриваемого примера построенный путь будет иметь вид, изображенный на Рис. 5.11. Очевидно, это не оптимальное решение, как ожидалось. Дело в том, что используемый вторичный критерий ведет к переходу от вершины, принадлежащей одному кратчайшему пути, π_1^*, к вершине, лежащей на другом кратчайшем пути π_2^*, которая лежит ближе к цели нежели следующая вершина π_1^*. В то время как такой переход, может не являться оптимальным (несмотря на то, что обе вершины принадлежат кратчайшим путям). Иначе говоря, поиск начинает переключаться между различными кратчайшими путями, из-за чего результирующий путь содержит лишние переходы, негативно сказывающиеся на его стоимости.

Для того чтобы избежать описанного выше эффекта предлагается строить эталонную PPM так, чтобы лишь вершины, принадлежащие единственному

[4]Такие критерии в литературе по эвристическому поиску носят название tie-breaking rule, дословно – правило разрешения ничьи (в некоторых видах спорта, например – в большом теннисе, для обозначения такого правила принята русская калька с этого англоязычного термина – тай-брейк).

(из множества возможных) кратчайшему пути, имели бы *pp*-значение 1. Для нахождения такого пути предлагается использовать алгоритм Theta* [7] – известный алгоритм построения путей на ГРД, подразумевающий возможность перехода между произвольными вершинами ГРД. Пути, возвращаемые этим алгоритмом, состоят из опорных вершин, смежных с углами препятствий, и вершин ГРД расположенных вдоль прямолинейных сегментов между опорными вершинами пути – см. Рис. 5.12.

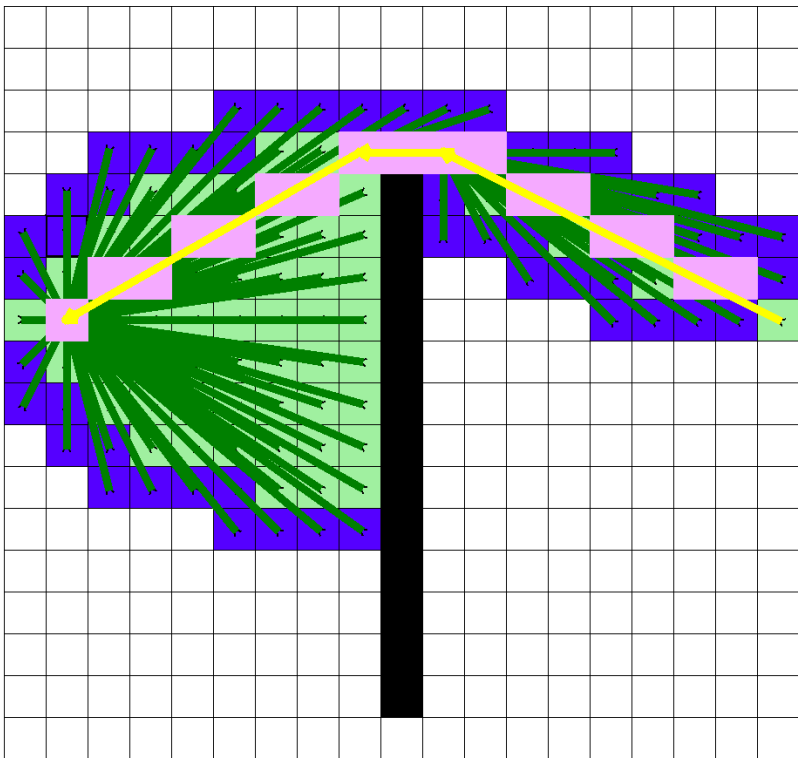


Рисунок 5.12 — Пример пути на ГРД, найденного с помощью алгоритма Theta*.

На рисунке дерево поиска показано зелеными линиями, результирующий путь в виде секций отмечен желтым цветом, а образующие его вершины (клетки) ГРД – розовым. В результирующей PPM именно этим вершинам предлагается присваивать значение $pp = 1$, а для всех остальных вершин предлагается определять *pp*-значение по следующей формуле, устанавливающей, насколько близка стоимость пути через вершину v к стоимости пути Theta*:

$$pp(v) = \frac{cost(\pi_{\texttt{Theta*}}(v_{start},v_{goal}))}{cost(\pi_{\texttt{Theta*}}(v_{start},v)) + cost(\pi_{\texttt{Theta*}}(v,v_{goal}))}, \tag{5.10}$$

где $\pi_{\texttt{Theta*}}(u,v)$ – путь Theta* от вершины u до вершины v.

Технически, для вычисления PPM производятся два запуска алгоритма Theta*. Первый начинается в начальной вершине v_{start} и используется критерий остановки – пока список OPEN не пуст. Т.е. исследуется всё пространство

поиска и, как следствие, для каждой вершины после завершения работы алгоритма становится известная стоимость Theta*-пути до этой вершины. Второй запуск алгоритма аналогичен первому, но теперь поиск ведётся от целевой вершины. После выполнения этих поисков для любой вершины v известно значение как $cost(\pi_{\texttt{Theta*}}(v_{start},v))$, так и $cost(\pi_{\texttt{Theta*}}(v,v_{goal}))$, что позволяет вычислить $pp(v)$ для всех вершин ГРД по формуле 5.10.

Кроме того, в ходе предварительных экспериментов было получено подтверждение, что применение дополнительных техник для создания более сфокусированных PPM, т.е. PPM с меньшим количеством ненулевых значений, сгруппированных вокруг единственного пути, дает положительный эффект. Первая такая техника – возведение pp-значений в степень. Пример показан на рис. 5.13, на третьем фрагменте слева, где все pp-значения возведены в 10-ю степень. В результате этой операции элементы PPM со значениями 1 остаются неизменными, тогда как pp-значения остальных элементов становятся существенно меньше. Вторая техника – обрезание (клиппинг) PPM, при котором pp-значения элементов, не превышающие определенный порог, устанавливаются в 0. Пример приведен на Рис. 5.13 справа, где обнулены все pp-значения ниже 0.95 (после возведения в степень).

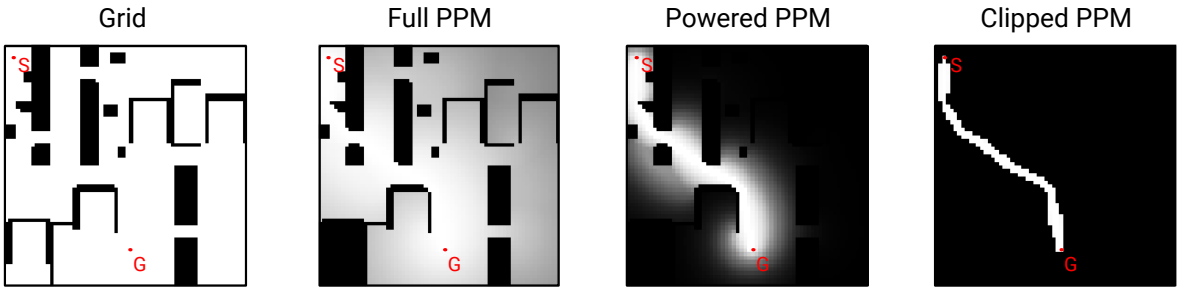


Рисунок 5.13 — Исходная задача PF-G (слева) и соответствующие карты вероятности пути: стандартная, после возведения в степень, после обнуления pp-значений, не превышающих заданный порог.

Эмпирические результаты, подтверждающие преимущество применения указанных техник, будут представлены далее в работе. Здесь же приведем неформальную гипотезу, объясняющую данный эффект. Возможно, использование эталонных PPM, содержащих лишь небольшое количество достаточно высоких pp-значений, сгруппированных вдоль единственного пути, на этапе обучения позволяет нейронной сети не тратить ресурсы на предсказание низких pp-значений, которые фактически не нужны для нахождения пути, а сосредоточиться на предсказании значений только для наиболее важных областей ГРД.

Эталонная разметка для задачи PF-RGB Как было изложено ранее, для решения задач PF-RGB предлагается использовать два варианта гибридных решателей: `A*+DEM`, когда по изображению рельефа происходит восстановление его карты высот (DEM) с последующим поиском пути алгоритмом `A*` с учетом этих высот, и `GBFS+PPM`, когда по изображению восстанавливается карта вероятностей вхождения в путь с последующим планированием по этой карте с помощью алгоритма `GBFS`. Следовательно, для обучения нейросетевой модели требуется два типа эталонных меток: эталонная цифровая модель рельефа (DEM) и эталонная карта вероятности вхождения в путь (PPM).

Процесс получения эталонных DEM не сопряжен с существенными затратами, т.к. на этапе обучения данные DEM доступны (каждому изображению сопоставлена модель рельефа). Единственная проблема здесь состоит в том, что исходные DEM хранят информацию о высотах в абсолютных значениях, и максимальная высота может существенно отличаться от одной карты к другой. Такое представление неудобно, т.к. выходные значения искусственных нейронных сетей обычно ограничены некоторым интервалом. Поэтому предлагается производить предварительную обработку каждой DEM с тем, чтобы итоговые значения высот лежали в интервале [0,1], где 0 соответствует минимальной высоте, а 1 – максимальной. Такая обработка облегчает последующее обучение нейросетевых моделей. С другой стороны, при таком подходе теряется информация о том, как соотносится пространственный масштаб и масштаб по высоте, что требует дополнительной настройки поправочного коэффициента α при расчете веса перехода между соседними пикселями (см. 5.1) в ручном режиме.

Процесс получения эталонных PPM для задач PF-RGB в целом аналогичен ранее описанному процессу для задач PF-G. Сначала производится запуск двух алгоритмов нефокусированного поиска: одного из начальной вершины, другого – из целевой. Это позволяет, во-первых, найти путь минимальной стоимости из начальной вершины в целевую: $\pi^*(v_{start}, v_{goal})$, во-вторых – определить для любой вершины, v, путь минимальной стоимости как из начальной вершины, так и до конечной: $\pi^*(v_{start}, v)$, $\pi^*(v, v_{goal})$. Далее производится ещё один запуск алгоритм несфокусированного поиска, но теперь список OPEN инициализируется не одной вершиной, как раньше (начальной или конечной), а всеми вершинами, образующими ранее найденный путь $\pi^*(v_{start}, v_{goal})$. Это позволяет вычислить для каждой вершины v стоимость пути из неё до пути $\pi^*(v_{start}, v_{goal})$

с учетом перепадов высот. Теперь итоговое *pp*-значение для вершины v вычисляется по формуле:

$$pp(v) = \frac{cost(\pi^*(v_{start}, v_{goal}))}{cost(\pi^*(v_{start}, v)) + cost(\pi^*(v, v_{goal})) + cost(\pi^*(v, \pi(v_{start}, v_{goal})))} \quad (5.11)$$

Совокупность этих значений и образует эталонную карту вероятности вхождения в путь – PPM. Каждой элемент PPM принадлежит интервалу (0,1]. Чем выше *pp*-значение элемент тем ближе расположен элемент к пути минимальной стоиомости. При этом *pp*-значение 1 присваивается лишь элементам, входящим в один из таких путей.

Заметим, что в отличие от формирования эталонных PPM для задачи PF-G, при формирование эталонных PPM для задачи PF-RGB не применяются такие техники как возведение *pp*-значений в степень и клиппирование PPM, т.к. эффективность этих техник для задач PF-RGB не была подтверждена экспериментально. Более подробно об этом будет сказано в разделе, содержащим результаты экспериментальных исследований.

5.2.3 Используемые модели и наборы данных

Нейросетевые модели Предлагается использовать несколько различных вариантов нейронных сетей для решения задачи предсказания PPM и CFM (в контексте решения задач PF-G) и PPM и DEM (в контексте решения задачи PF-RGB).

В качестве основного варианта архитектуры искусственной нейронной сети, используемой как для решения задач PF-G, так и для задач PF-RGB, предлагается архитектура, состоящая из трех основных блоков:

1. Блок кодирования входных данных (на основе сверточных операций).
2. Блок-трансформер, реализующий механизм (пространственного) внимания (т.н. механизм attention).
3. Блок декодирования.

Предлагаемая архитектура изображена на Рис. 5.14.

Блок кодирования входных данных предназначен для извлечения локальных особенностей конкретного экземпляра задачи планирования. Например,

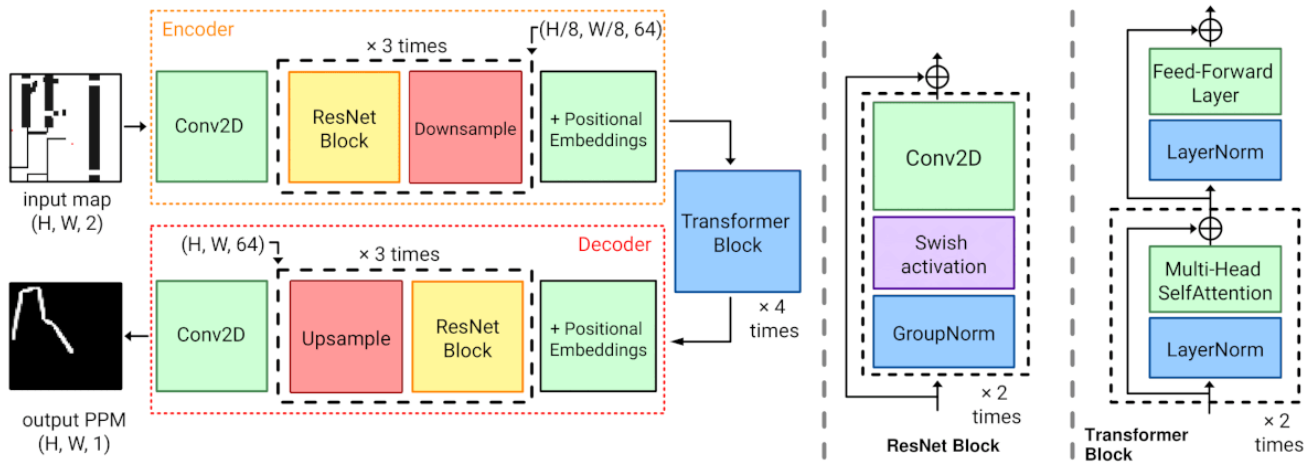


Рисунок 5.14 — Базовая архитектура глубокой нейронной сети, предлагаемая в работе для предсказания PPM, CFM и DEM.

для задач PF-G такими особенностями могут быть углы препятствий или узкие проходы, для задач RF-RGB – области с резкими перепадами высот. Этот блок основан на использовании т.н. ResNet-архитектуры [239], особенностью которой является использование остаточных соединений (skip connection) для передачи информации между слоями нейросети. Более детальная архитектура используемого ResNet блока показана на Рис. 5.14, справа от изображения основной архитектуры.

Далее следует блок-трансформер, основанный на использовании т.н. механизма внимания. На рисунке выше блок-трансформер показан синим цветом, а его более подробное устройство изображено на правой части Рис. 5.14. Основная задача этой части нейросети – установление взаимосвязей между выявленными ранее особенностями [240]. Стоит отметить, что точное понимания эффекта, который достигается с помощью блоков/слоев внимания (attention) в настоящее время отсутствует, однако, есть множество оснований считать, что с их помощью происходит оценка того, насколько различные выявленные особенности соотносятся между собой в контексте решения задачи. Например, в задаче PF-G, где особенностями могут, предположительно, выступать узкие проходы между препятствиями, с помощью блоков внимания может быть установлено, что для решения конкретной задачи планирования необходимо, чтобы путь включал себя прохождение нескольких проходов.

Стоит отметить, что блоки внимания изначально разрабатывались для задач моделирования текстовых последовательностей, которые имеют линейную структуру. Поскольку рассматриваемые задачи характеризуются

двумерной структурой, предлагается использовать технику позиционного кодирования [241; 242] (т.н. визуальный трансформер). Эта техника позволяет преобразовывать 2D-карты признаков в векторы (перед блоками внимания) и обратно (после), обеспечивая возможность сохранения пространственных отношений между признаками.

Наконец, последний блок искусственной нейронной сети, декодер, предназначен для формирования итогового выхода, т.е. PPM/CFM (для задачи PF-G) или PPM/DEM (для задачи PF-RGB). Интуитивно, он осуществляет преобразование выявленных ранее взаимосвязей (между особенностями задачи планирования) в конкретный, ожидаемый выходной формат (например, матрицу значений вероятностей вхождения в искомый путь).

Общее число параметров в предлагаемой искусственной нейронной сети составляет порядка 1 миллиона. Это достаточно небольшое число по меркам современных нейросетей. Так трансформерные нейросети для больших языковых моделей содержат обычно десятки или даже сотни миллиардов параметров.

Для предсказания PPM/DEM в контексте решения задач PF-RGB в качестве альтернативных архитектур предлагаются архитектуры, активно использующиеся для задач анализа и обработки изображений, а именно для задач трансформации изображений [243], т.к. задачу предсказания DEM или PPM по цветному изображению можно рассматривать, как задачу обусловленной трансформации входного изображения (снимка рельефа местности). Предлагается использовать два типа архитектур: генеративно-состязательные сети [244; 245] и латентные диффузионные модели [246; 247].

Генеративно-состязательные сети, ГСС, (в англоязычной терминологии – GAN, от generative adversarial networks) представляют собой класс моделей машинного обучения, предназначенных для аппроксимации заданного распределения данных и последующей генерации отдельных элементов (семплов) этого распределения. Такие сети впервые были предложены в работе [248] и с тех пор получили широкое распространение для решения широкого круга задач, в том числе для задач (обусловленной) трансформации изображений.

Архитектурно ГСС состоят из двух основных компонент: генератора и дискриминатора. Роль генератора заключается в создании новых экземпляров данных, в то время как дискриминатор учится различать реальные данные из обучающей выборки и данные, созданные генератором. Конечная цель обучения

состоит в том, чтобы получить генератор, способный создавать данные, которые дискриминатор не сможет отличить от реальных.

Процесс обучения ГСС представляет собой состязательную игру, в которой генератор и дискриминатор обучаются одновременно. Веса дискриминатора обновляются так, чтобы лучше различать реальные данные от сгенерированных, в то время как веса генератора обновляются на основе того, насколько успешно дискриминатор смог классифицировать его вывод как реальный или поддельный. Этот одновременный процесс обучения часто сравнивают с игрой двух игроков с минимаксной стратегией, где дискриминатор стремится максимизировать свою точность (минимизировать потери), а генератор – максимизировать ошибку дискриминатора (максимизировать свой выигрыш). Со временем этот состязательный процесс приводит к тому, что генератор производит все более реалистичные данные.

В данной работе предлагается использовать один из варинатов ГСС, а именно вариант `StyleGAN3`, предложенный в работе [249] и показавший отличные (для своего времени) результаты в задачах условной генерации изображений.

Помимо ГСС, предлагается использовать диффузионные модели, которые демонстрируют значительный прогресс и высокие результаты в синтезе высококачественных изображений. Идейно диффузионные модели осуществляют итеративное удаление шума из нормально-распределенной случайной величины (в контексте проводимого исследования – случайного изображения нужного размера) и трансформацию случайного семпла в элемент требуемого распределения данных.

В данной работе предлагается использовать один из возможных вариантов диффузиионных моделей, а именно – латентную диффузию [250], в которой процесс расшумления осуществляется в латентном пространстве, которое преобразуется в пространство исходных данных с помощью отдельных (обучаемых) блоков кодирования и декодирования. Переход к работе в латентном пространстве достаточно распространен в соврменном машинном обучении, т.к. характеризуется универсальностью и демонстрирует высокую эффективность при решении практических задач.

Общая схема предлагаемых к использованию в работе подходов к аппроксимации DEM и PPM в контексте решения задачи PF-RGB представлена на Рис. 5.15.

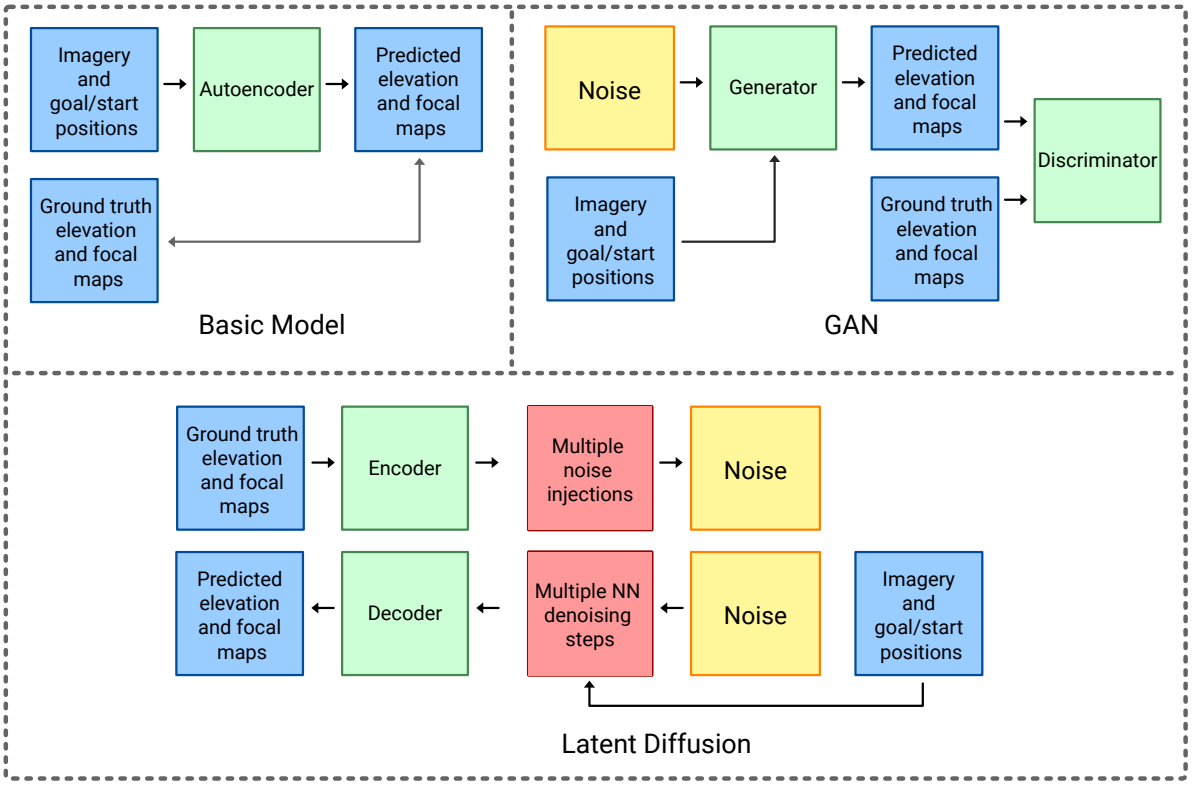


Рисунок 5.15 — Укрупненные схемы базовой и дополнительных нейросетевых архитектур, используемых для решения задачи PF-RGB.

Наборы данных Для решения задач PF-G и PF-RGB были разработаны авторские наборы данных, характеризующиеся большим объемом и разнобразием, что позволяет проводить как обучение нейросетевых моделей, так и тестирование результирующих гибридных алгоритмов поиска пути.

Набор данных для решения задачи PF-G был разработан с опорой на ранее используемый в этой области (области интеграции искусственных нейронных сетей и алгоритмов эвристического поиска для решения задач поиска пути на ГРД) датасет MP, предложенный в [39]. Каждый ГРД в наборе MP имеет размер 32×32 и относится к одной из 8 топологий, представленных на Рис. 5.16 слева. Поиск пути на ГРД таких топологий, очевидно, представляет собой непростую задачу из-за сложной конфигурации непроходимых областей (препятствий). Наличие таких конфигураций приводит к тому, что для многих вершин ГРД стандартная эвристическая функция сильно недооценивает длину пути до цели и её применения на практике приводит к чрезмерному числу итераций алгоритма поиска (как было показано выше).

В работе [40], в которой предлается один из наиболее сильных современных вариантов метода планирования, основанного на интеграции искусственных нейронных сетей и алгоритма эвристического поиска, `NeuralA*`, на основе

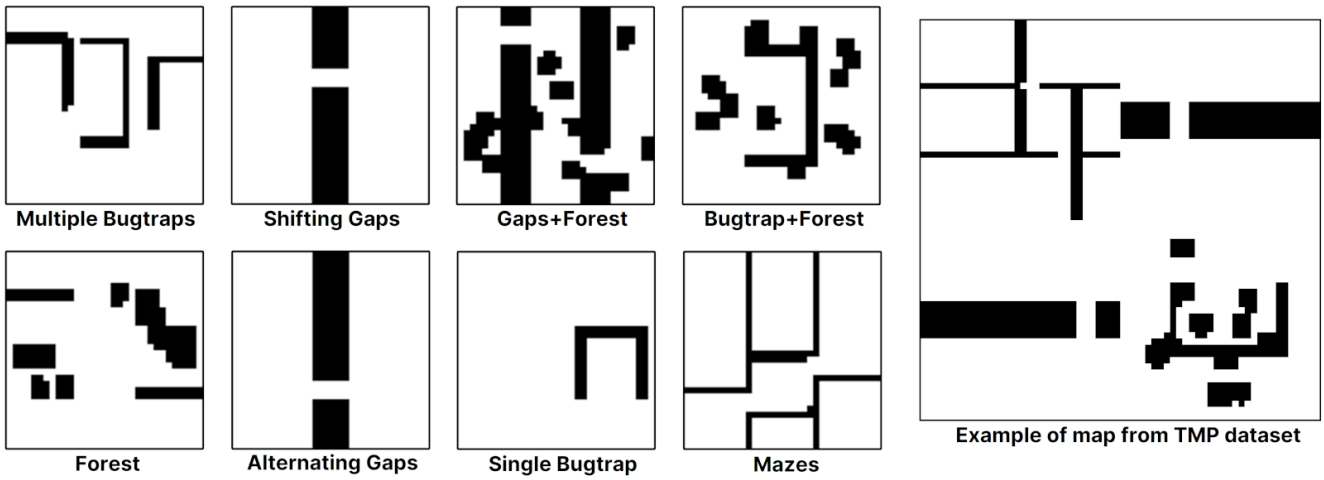


Рисунок 5.16 — Примеры карт из наборов данных, используемых для решения задачи PF-G.

датасета MP был предложен датасет TMP, содержащий более сложные задачи поиска. Каждый ГРД в наборе данных TMP имеет размер 64×64 и представляет собой комбинацию четырех различных ГРД из датасета MP. Пример такого ГРД изображен на Рис. 5.16 справа. Именно карты такого вида предлагается использовать в работе. Однако оригинальный данных данных TMP насчитывает всего 4 000 карт. Это достаточно ограниченный набор, требующий дальнейшего расширения для проведения полноценных исследований.

Для расширения датасета TMP в работе предлагается воспользоваться техникой аугментации данных, т.е. модификации уже имеющихся в наборе данных карт для создания на их основе новых элементов датасета. Для этого каждая карта TMP разбивается на 4 фрагмента 32×32 (т.е. на карты MP, из которых она была составлена), далее к каждому фрагменту применяется случайная комбинация поворотов и зеркальных отображений. Вновь полученные фрагменты заново объединяются в одну карту размера 64×64. С помощью этой техники, объем начальной выборки ГРД был увеличен в 16 раз, т.е. было сгенерировано 64 000 различных карт.

Далее для отдельного ГРД производилась генерация задания. Сначала случайным образом выбиралась целевая вершина. Затем осуществлялся запуск алгоритма неинформированного поиска из целевой вершины для того, чтобы для каждой вершины ГРД определить длину кратчайшего пути. С помощью рассчитанных значений выделялась треть наиболее удаленных от цели вершин и уже из них случайным образом выбиралась вершина, которая будет являться начальной. Подобный способ генерации заданий обеспечивает гене-

рацию нетривиальных заданий для последующего использования. Более того, по аналогии с [19], задание исключалось из выборки, если для него соотношение $cost(\pi^*(v_{start}, v_{goal}))/h(v_{start})$ было меньше 1.05. Указанное соотношение является разумным индикатором сложности задания, – чем ближе оно к 1.0 тем более путь от старта до финиша представляет собой прямую, которой не требуется огибать никакие препятствия.

Всего для каждой карты было сгенерировано 10 различных заданий. Общий объем выборки составил 640 000. Эта выборка была разбита в пропорции 8:1:1 на обучающую, валидационную и тестовую. Т.е. на этапе обучения предполагается использование 576 000 задач, из них для обновления весов нейросети (обучения) используется 512 000 заданий, а оставшиеся 64 000 используются для дополнительного контроля значений функции потерь во время обучения (т.н. валидации), с тем чтобы избежать нежелательных эффектов, например – переобучения. Наконец, отложенная для проведения итоговых экспериментов выборка (тестовая выборка) тоже насчитывает 64 000 заданий.

Для решения задачи PF-RGB в работе создан новый датасет пар RGB-DEM, представляющих собой реальные земные ландшафты. Основным источником для составления набора данных послужил инструмент `NOAA: Data Access Viewer`[5], предназначенный для обработки и визуализации геопространственных данных.

Для создания датасета были использованы данные о фрагменте земной поверхности, площадью примерно 128×180 километров и характеризующейся холмистым рельефом. Исходное RGB-изображение (и соответсвующая DEM) имеет размер $12\ 875 \times 18\ 000$ пикселей – см. Рис. 5.17. Пространственное разрешение составляет 10 метров на пиксель, что обеспечивает достаточную детализацию физических особенностей ландшафта.

Из этих данных было создано 18 316 согласованных пар RGB-DEM. Сначала исходное изображение было разбито на фрагменты трех масштабов: 256×256, 512×512 и 1024×1024 пикселей – см. Рис. 5.18. Наличие фрагментов с разным масштабом необходимо для того, чтобы нейросетевая модель впоследствии на этапе обучения могла научиться выделять особенности рельефа независимо от пространственного масштаба для повышения робастности и универсальности (т.е. возможности работать с разными масштабами).

[5] https://coast.noaa.gov/dataviewer/

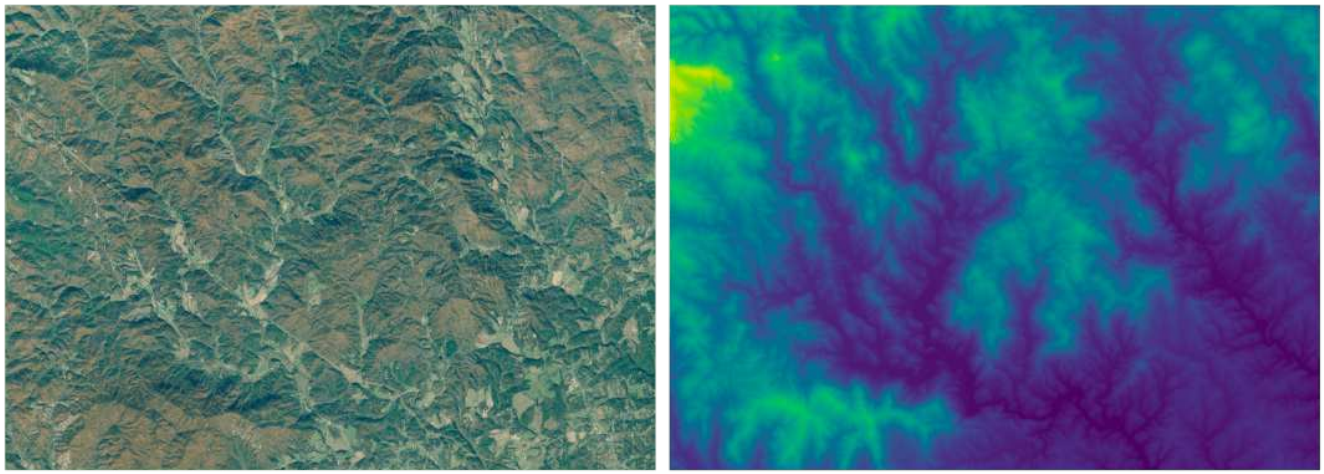


Рисунок 5.17 — Общий вид модели исходный модели рельефа (изображение и карта высот), используемой для создания набора данных.

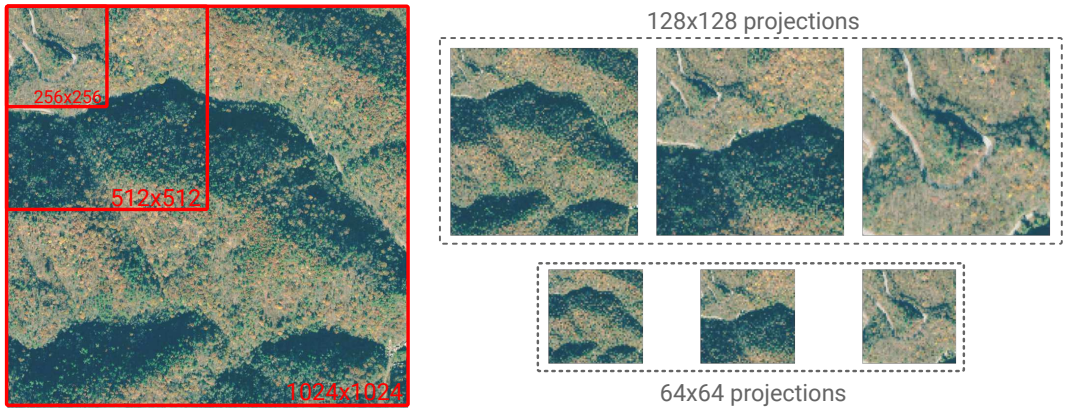


Рисунок 5.18 — Фрагменты изображений ландшафта различного масштаба и их представление в виде изображений стандартного размера для сохранения в наборе данных.

Далее все фрагменты трансформировались с помощью механизма билинейной интерполяции к размерам 64×64 и 128×128. Это обеспечивает совместимость с входными размерами нейросети, используемой для решения задачи PF-G и позитивно сказывается на скорости обучения.

Наконец, каждая карта рельефа была нормализована с тем, чтобы содержать значения в одинаковом интервале – от 0 (минимальная высота) до 1 (максимальная высота). Для этого на отдельной DEM сначала определялось минимальное значение и оно вычиталось из всех элементов DEM (т.о. начальное значение высоты теперь становилось равным 0). Затем определялось максимальное значение высоты и каждый элемент DEM делился на это значение.

$$\begin{aligned} \text{minDem} &= \min_{i,j}(dem(i,j)) \\ dem(i,j) &= dem(i,j) - \text{minDEM} \\ \text{maxDem} &= \max_{i,j}(dem(i,j)) \\ dem(i,j) &= \frac{dem(i,j)}{\text{maxDem}} \end{aligned} \tag{5.12}$$

Как и ранее, для расширения выборки и увеличения разнообразия данных дополнительно использовалась техника аугментации, а именно использовались различные вращения имеющихся фрагментов.

После этапа нормализции и аугментации, для каждой полученной карты (пары RGB-DEM) было сгенерировано 10 различных задач поиска: сначала производился случайный выбор начального элемента, затем – случайный выбор целевого элемента.

Итоговый набор данных содержит:

– 18 316 карт (пар RGB-DEM).

– 183 160 задач поиска.

Этот набор разделен на обучающую, валидационную и тестовую выборки в пропорции 8:1:1.

Таким образом, в результате работы созданы обширные наборы данных (датасеты), насчитывающие сотни тысяч различных задач PG-G и PF-RGB. Эти датасеты опубликованы в открытом доступе[6], что позволяет другим исследователям как воспроизводить результаты данного исследования, так и использовать имеющиеся данные для разработки собственных решений.

5.3 Экспериментальные исследования

5.3.1 Экспериментальное исследование методов решения задачи PF-G

Исследуемые алгоритмы Предложенные в работе гибридные методы решения задачи PF-G – `WA*+CF`, `FS+PPM` и `GBFS+PPM` были программно реализованы

[6] https://github.com/AIRI-Institute/TransPath

на языках программирования C++ (алгоритмы поиска) и Python (нейросетевая аппроксимация эвристических функций)[7] и было проведено их сравнительное экспериментальное исследование. В качестве альтернативных методов для сравнения были выбраны, как классические методы планирования: A* и WA*, так и методы, концептуально схожие с предложенными, т.е. использующие современные нейросетевые модели для предсказания различных эвристик с их последующей интеграцией в алгоритмы поиска.

В качестве основной альтернативы для сравнения использовался один из наиболее современных обучаемых планировщиков NeuralA*, предложенный в [40]. Этот метод опирается на предсказание для каждой вершины входного ГРД дополнительной стоимости перехода в эту вершину: чем менее вершина релевантна конкретной задаче планирования, тем выше эта стоимость. Иначе говоря, NeuralA* предсказывает дополнительный штраф для вершин ГРД с тем, чтобы осуществлять инвертированную фокусировку поиска. Второй важной особенностью этого планировщика является то, что сам алгоритм поиска в нём реализован в виде совокупности матричных операций. За счет этого становится возможным считать операции поиска частью общей нейросетевой архитектуры и осуществлять т.н. сквозное обучение (end-to-end learning). Как было показано авторами в оригинальной публикации, NeuralA* существенно превосходил другие современные обучаемые подходы в задаче поиска пути на ГРД, в том числе методы, описанные в [39; 251]. В целом, можно утверждать, что на момент проведения данного исследования, NeuralA* являлся самым современным и эффективным гибридным методом поиска пути на ГРД (т.н. SOTA, от англ. state of the art), и при этом – обеспечивающих воспроизводимость результатов за счет наличия открытого исходного кода, который и был использован в данной работе. Стоит лишь заметить, что изначально в NeuralA* стоимость перехода между любыми смежными вершинами (как ортогонально-смежными, так и диагонально-смежными) считалась равной 1. Для данной работы в код метода были внесены изменения, позволяющие корректно различать переходы между горизонтально- и вертикально-смежными вершинами (с весом равным 1) и между диагонально-смежными вершинами (с весом равным $\sqrt{2}$).

Дополнительно в качестве альтернативного обучаемого планировщика для проведения экспериментов использовался метод, предложенный в [19]. Он основан на обучении предсказанию значений идеальной эвристической функции

[7] Исходный код доступен по адресу: https://www.github.com/AIRI-Institute/TransPath

и последующему использованию этих значений в алгоритме A*. Этот планировщик далее будет обозначаться как A*+HL.

Для обеспечения корректности сравнения во всех исследуемых гибридных планировщиках: WA*+CF, FS+PPM, GBFS+PPM, NeuralA*, A*+HL, использовалась одна и та же нейросетевая модель (предложенная в работе), которая обучалась с нуля на предлагаемом в работе наборе данных[8].

Два из семи сравниваемых алгоритмов, а именно – FS+PPM и WA*, гарантируют построение ограниченно субоптимальных решений и подразумевают передачу фактора субоптимальности $w > 1$ на вход алгоритму. Значение w в экспериментах варьировалось в диапазоне $\{1.1, 1.25, 1.5, 2, 3, 4, 5, 10\}$. При $w = 2$ алгоритм WA* демонстрировал наилучший баланс "скорость работы - качество решения", поэтому для облегчения восприятия далее будут указаны результаты FS+PPM и WA* для $w = 2$, а более детальные результаты для различных w будут приведены дополнительно.

Детали обучения Для обеспечения корректного и эффективного функционирования исследуемых гибридных методов решения задачи PF-G, – WA*+CF, FS+PPM, GBFS+PPM, NeuralA*, A*+HL, – необходим этап обучения. Все методы в рамках этого этапа использовали одинаковый набор данных (описанный ранее) и одинаковые вычислительные мощности – 1 GPU-вычислитель NVidia A100 80Gb. Обучение производилось с размером пакета данных (batch size) равным 512, число эпох обучения равнялось 35. При обучении использовался алгоритм оптимизации Adam [252] и динамически-варьируемый темп обучения (learning rate) с использованием метода OneCycleLR [253] и максимальным значением 4×10^{-4}.

Для обучения предсказания предлагаемых в работе эвристических функций PPM и CFM использовалась L_2 функции потерь. Для обучения предсказания значений идеальной эвристики для метода A*+HL использовалась L_1 функции потерь, т.к. именно она использовалась в оригинальной работе, посвященной данному методу. Для обучения эвристической функции, используемой в алгоритме NeuralA*, применялась авторская функция потерь описанная в работе, посвященной этому методу[9].

[8]Результаты, получаемые с использованием оригинальных моделей для NeuralA*, A*+HL, были гораздо хуже.

[9]Эта функция представляет собой разницу между числом рассмотренных NeuralA* при поиске состояний и числом состояний, составляющих кратчайший путь

Длительность обучения нейросетевой модели для предсказания предлагаемых в работе эвристик и эвристики для алгоритма A*+HL, составила 4 часа (на указанном выше оборудовании). Аналогичный показатель для NeuralA* составил 16 часов. Такая разница в скорости обучения объясняется тем, что в последнем случае нейронная сеть включает в себя дополнительные блоки, ответственные за реализацию логики алгоритма поиска, и это существенно замедляет процесс обучения.

Результаты Отдельный эксперимент состоит в запуске всех исследуемых алгоритмов решения задачи PF-G на отдельном задании из тестовой выборки (насчитывающей всего 64 000 заданий). Отслеживались следующие два основных индикатора эффективности работы – число итераций алгоритма (машинонезависимый индикатор оценивающий скорость работы алгоритма поиска) и стоимость построенного пути (показатель качества отыскиваемых решений). Численные значения этих индикаторов в каждом эксперименте нормировались (т.е. делились) на соответствующие значения для алгоритма A*, который считался эталонным. Затем проводился анализ всей выборки полученных результатов – индикаторы усреднялись (дополнительно вычислялось стандартное отклонение), и вычислялся процент заданий (по всей тестовой выборке), в которых алгоритмом поиска был найден путь минимальной стоимости (т.е. оптимальное решение задачи PF-G).

Результаты экспериментов приведены в Таблице 5.1.

Каждая строка в таблице соответствует исследуемому алгоритму (его название приведено в первом столбце). Столбцы 2-4 соответствуют отслеживаемым индикаторам эффективности:

- Optimal Found Ratio (%) – процент заданий, в которых алгоритм построил оптимальное решение.
- Cost Ratio (%) – нормированная стоимость отыскиваемых решений (чем ближе сверху к 100%, тем лучше).
- Expansions Ratio (%) – нормированное число итераций, совершенных алгоритмом поиска для отыскания решения (нормировка производится на число итераций, затраченных алгоритмом A*). Чем ниже это значение, тем лучше.

В таблице значения до знака ± соответствуют средним значения по тестовой выборке, значения после знака ± показывают стандартное отклонение по выборке.

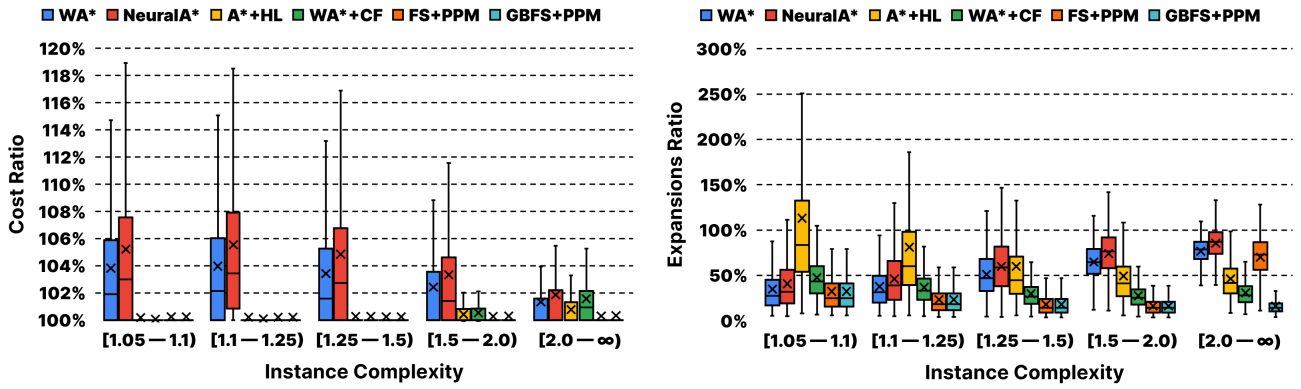
Таблица 5.1 — Результаты экспериментального исследования различных методов решения задачи PF-G.

	Optimal Found Ratio (%) ↑	Cost Ratio (%) ↓	Expansions Ratio (%) ↓
A*	100	100	100
WA*	40.66	103.52 ± 4.85	44.43 ± 25.92
NeuralA*	29.82	104.90 ± 6.56	52.30 ± 30.47
A*+HL	79.11	100.27 ± 0.62	80.50 ± 74.40
WA*+CF	**85.40**	100.25 ± 1.13	36.98 ± 21.18
FS+PPM	82.97	**100.24 ± 0.74**	26.36 ± 21.08
GBFS+PPM	83.02	100.25 ± 0.90	**23.60 ± 18.34**

Анализируя представленные результаты, можно сделать следующие выводы. Во-первых, предлагаемые в работе гибридные методы решения задачи PF-G, – WA*+CF, FS+PPM, GBFS+PPM, – также как и ранее известные подходы NeuralA*, A*+HL, очевидно, обладают способностью к генерализации, т.е. способны эффективно решать задачи, не представленные в обучаемой выборке. Во-вторых, предлагаемые алгоритмы превосходят аналоги по эффективности и достигают наилучших показателей. Так, например, метод WA*+CF является наилучшим по показателю Optimal Found Ratio (число заданий, в которых найдено оптимальное решение), а следом идут GBFS+PPM и FS+PPM. По стоимости решений (длине отыскиваемых путй) наилучший показатель демонстрирует алгоритм FS+PPM, а по числу итераций – GBFS+PPM. В целом, можно утверждать, что последний алгоритм на практике обладает наилучшим соотношением “число итераций – длина отыскиваемых путей”. Так он в среднем сокращает число итераций (относительно A*) в 4 раза (т.е. затрачивает в среднем 23.6% итераций от A*, как видно из последней ячейки таблицы), при этом построенные решения в среднем не превосходят оптимальные более чем на 0.5% стоимости, а в 83% случаев решения являются оптимальными.

Более детально результаты экспериментов представлены на Рис. 5.19.

На приведенном выше рисунке изображены диаграммы размаха для показателей Cost Ratio (относительная стоимость решения) и Expansions Ratio



а) Относительная стоимость решения. б) Относительное число итераций.

Рисунок 5.19 — Относительная стоимость решения и число итераций алгоритма, в зависимости от сложности задачи PF-G.

(относительное число итераций алгоритма) в зависимости от сложности заданий. Здесь сложность определяется как отношение $cost(\pi^*)/h_{octile}(v_{start})$, где в числителе стоит стоимость оптимального пути, а в знаменателе – значение стандартной эвристики, не учитывающей препятствия, для начальной вершины. Чем больше это значение хуже оценивает эвристика длину пути от старта до финиша (очевидно это происходит из-за того, что на пути встречаются препятствия, которые необходимо огибать).

Из рисунка видно, что алгоритмы `WA*` и `NeuralA*` существенно уступают остальным методам по стоимости решения на заданиях не самой высокой сложности и только лишь на заданиях с показателями сложности больше 2 ситуация выравнивается. Тем не менее, очевидно, что методы `FS+PPM` и `GBFS+PPM` демонстрируют наилучшую относительную стоимость во всем диапазоне сложностей.

Что касается числа итераций, то можно выделить следующие наблюдения. Эффективность `WA*` и `NeuralA*` снижается с ростом сложности заданий (относительное число итераций возрастает), в то время как такого тренда для других алгоритмов не наблюдается. Более того, предлагаемый в работе алгоритм `GBFS+PPM` на сложных заданиях демонстрирует результаты лучше, чем на более легких. Отдельно стоит остановиться на результатах `FS+PPM`, в частности на существенном росте числа итераций при переходе к заданиям с показателем сложности больше 2. Этот скачкообразный рост объясняется тем, что фактор субоптимальности, передаваемый на вход алгоритму тоже равняется 2. Если повысить фактор субоптимальности, то наблюдаемого негативного эффекта роста числа итераций не произойдёт, что подтверждается результатами `GBFS+PPM`

(алгоритма, который является вариантом FS+PPM с неограниченным фактором субоптимальности).

В целом, проведенный анализ полученных результатов экспериментальных исследований можно утверждать, что предложенные в работе гибридные методы решения задач PF-G, в особенности метод GBFS+PPM, существенно превосходят аналоги как по качеству отыскиваемых решений, так и по скорости работы, измеренной в числе итераций алгоритма поиска.

Здесь стоит отметить, что с практической зрения важно оценивать время работы алгоритма и в абсолютных величинах (в секундах). Однако такая оценка и сравнение может быть затруднена из-за особенностей реализации алгоритмов, особенно гибридных, использующих нейросетевые модели для предсказания эвристик. Так, например, метод NeuralA* полностью реализован на Python, в то время как предлагаемые в данной работе методы, WA*+CF, FS+PPM, GBFS+PPM, реализованы на C++ (в части алгоритмов поиска) и Python (в части нейросетевой аппроксимации эвристик). Следовательно, сравнивать, например, время работы NeuralA* и GBFS+PPM не вполне корректно. Поэтому далее приведем данные лишь по методам, предлагаемым в работе.

Для пакета данных размером 64 (batch size = 64) и при условии использования типа данных float32 (32-разрядные числа с плавающей точкой) предсказание эвристик занимает примерно 9.5 мс на вычислителе Tesla A100 (и около 40 мс на GTX 1660s). Последующий же поиск путей для этих 64 заданий занимает следующее время: A*+HL – 96 мс, WA*+CF – 60 мс, FS+PPM – 37 мс, GBFS+PPM – 31 мс. Если же решать эти задания классическими алгоритмами поиска (без этапа предсказания эвристик), то временные затраты характеризуются следующим образом: A* – 155 мс, WA* – 77 мс. Таким образом, в режиме пакетной обработки заданий и при использовании современного аппаратного обеспечения (NVidia A100), предлагаемый в работе метод GBFS+PPM превосходит по быстродействию классические аналоги в несколько раз. Заметим, что это соотношение может быть изменено в ту или иную сторону. Например, при увеличении размера пакета входных данных скорость работы предлагаемых методов ещё более повысится, с другой стороны, при использовании маломощных GPU-вычислителей и/или отказа от режима пакетной обработки входных данных (т.е. при последовательном решении задач PF-G) скорость работы, наоборот, понизится.

В завершении приведем ряд примеров решений задач PF-G, построенных исследуемыми алгоритмами – см. Рис. 5.20.

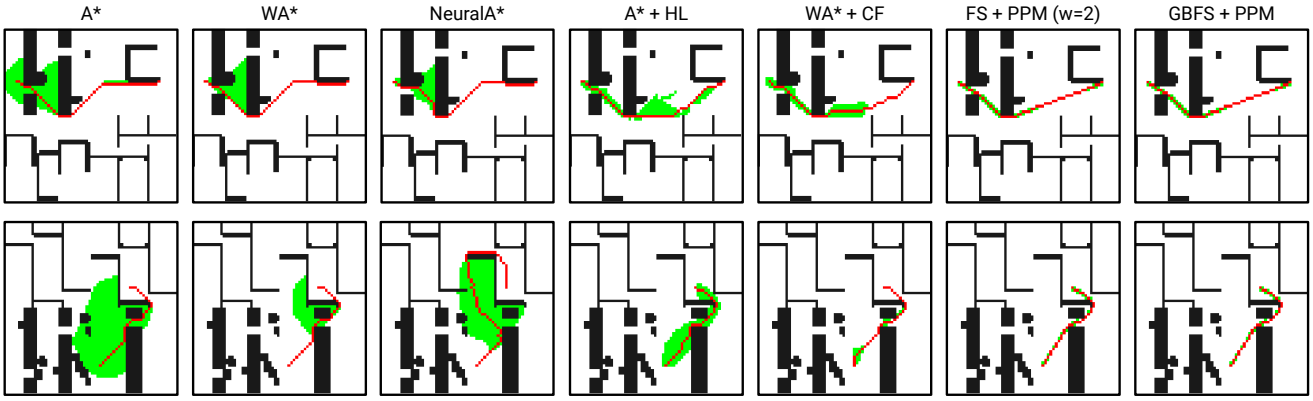


Рисунок 5.20 — Примеры решения задач PF-G различными алгоритмами поиска (как классическими, так и гибридными).

На рисунке в строках изображены различные задачи PF-G, а в столбцах – результаты решения этих задач исследуемыми алгоритмами. Зеленой заливкой показаны области, содержащие вершины рассмотренные алгоритмами (чем меньше зеленых зон, тем лучше). Приведенный рисунок наглядно демонстрирует преимущество предлагаемых в работе подходов, `WA*+CF`, `FS+PPM`, `GBFS+PPM`, перед аналогами.

5.3.2 Экспериментальные исследования методов решения задачи PF-RGB

Для проведения экспериментальных исследований использовалась тестовая выборка заданий, описанная ранее в Разделе 5.2.3, состоящая из 18 316 задач PF-RGB. Каждая задача представлена изображением местности (размером 64 × 64 и 128 × 128) и координатами начального и целевого пикселей. Дополнительно для каждой задачи известна, но недоступна планировщику, информация о рельефе местности (DEM канал). Эта информация используется для количественной оценки того, насколько хорошо исследуемые методы справляются с решением задач, путем нормировки наблюдаемых показателей эффективности на таковые для алгоритма поиска, обладающего привилегированной информацией о DEM-канале.

Всего было проведено три серии экспериментов. В первой серии предлагаемое в работе базовое решение задач PF-RGB, т.е. метод GBFS+PPM, сравнивалось с методом NeuralA*. Для этого эксперимента предсказание PPM осуществлялось с помощью базовой нейросетевой модели, описанной в работе. Затем были проведены сравнительные эксперименты с целью установить, как использование более продвинутых нейросетевых архитектур (генеративно-состязательных сетей и латентных диффузионных моделей) влияет на точность предсказания PPM и, как следствие, на эффективность процесса планирования. В последней серии экспериментов исследовался альтернативный подход к решению задачи PF-RGB, A*+DEM, основанный на нейросетевой аппроксимации DEM-канала по доступному изображению, и последующему поиску пути по построенной карте рельефа с помощью классического алгоритма A*. Опишем далее результаты всех упомянутых экспериментов.

Первая серия экспериментов В этих экспериментах, предлагаемый в работе метод решения задач PF-RGB, GBFS+PPM, сравнивался с методом NeuralA*. Для более корректного сравнения использовались только изображения размера 64×64, т.к. эффективность NeuralA* заметно снижалась при переходе к изображениям размера 128×128 (в отличие от GBFS+PPM, что будет продемонстрировано впоследствии).

Как и ранее при решении каждой задачи PF-RGB из тестовой выборки отслеживались следующие показатели: число итераций алгоритма и стоимость построенного решения (в данном случае это – длина пути с учетом перепада высот). Для более наглядного сравнения полученные показатели нормировались на показатели алгоритма A*, которому была доступна для поиска привилегированная информация, а именно – DEM-канал. Агрегированные по всей тестовой выборке результаты представлены в Табл. 5.2.

Таблица 5.2 — Результаты экспериментального исследования методов решения задачи PF-RGB.

	NeuralA*	GBFS+PPM
Cost Ratio (%)	121 ± 20	$\mathbf{106 \pm 10}$
Expansions Ratio (%)	64 ± 43	$\mathbf{44 \pm 32}$

Строки таблицы соответствуют отслеживаемым индикаторам качества работы алгоритмов:

– Cost Ratio (%) – нормированная стоимость отыскиваемых решений (чем ближе сверху к 100%, тем лучше).
– Expansions Ratio (%) – нормированное число итераций, совершенных алгоритмом поиска для отыскания решения (нормировка производится на число итераций, затраченных алгоритмом A* при поиске с привелигиреванной информацией о DEM-канале). Чем ниже это значение, тем лучше.

В таблице значения до знака ± соответствуют средним значения по тестовой выборке, значения после знака ± показывают стандартное отклонение по выборке.

Как видно из представленных результатов, рассматриваемые методы заметно сокращают число итераций алгоритма поиска, при этом предлагаемый в работе метод, GBFS+PPM, превосходит по этому показателю конкурента (см. последнюю ячейку таблицы). Так, среднее число итераций GBFS+PPM составляет 44%, т.е. алгоритм затрачивает в два раза меньше итераций на поиск решения по сравнению с алгоритмом A*, которому доступна информация о DEM-канале. Что касается качества отыскиваемых решений, т.е. стоимости конструируемых путей, то и по этому показателю GBFS+PPM опережает NeuralA*. В среднем первый алгоритм отыскивает решения, стоимость которых превышает стоимость оптимальных решений на 6%, а второй – на 21%. Итак, можно сделать вывод о том, что предлагаемый в работе способ, во-первых, превосходит современные мировые аналоги, предназначенные для решения задач поиска пути по изображению, во-вторых позволяет отыскивать решения высокого качества, уступающие оптимальные решениям по стоимости не более, чем на несколько процентов. Стоит ещё раз подчеркнуть, что на практике, в условиях отсутствия информации о DEM-канале, построение гарантированно оптимальных решений невозможно в принципе, т.к. невозможно использовать классические алгоритмы (эвристического) поиска из-за того, что информация о стоимости переходов между вершинами графа (пикселями на изображении) отсутствует.

Вторая серия экспериментов Основной целью этой серии экспериментов было установить, насколько использование альтернативных нейросетевых архитектур (по сравнению с базовой архитектурой) влияет на качество предсказания PPM и, соответственно, эффективность решения задач PF-RGB в целом. Для

Таблица 5.3 — Результаты экспериментального исследования различных нейросетевых архитектур для решения задачи PF-RGB.

	LDM	LDM-cond	GAN	Autoencoder
Cost Ratio (%)	105 ± 8	**103 ± 5**	105 ± 9	106 ± 9
Expansions Ratio (%)	44.9 ± 78.6	**44.2 ± 67**	54.1 ± 99.4	62.9 ± 92.5
MSE ($\times 10^{-3}$)	1.66 ± 0.14	**1.61 ± 0.13**	1.72 ± 0.14	1.8 ± 0.17

этой серии экспериментов, использовались архитектуры описанные в Разделе 5.2.3:

– Базовая архитектура, которая будет обозначаться как `Autoencoder`.
– Генеративно-состязательная модель, обозначаемая как `GAN`.
– Латентная диффузионная модель, обозначаемая как `LDM`.
– Латентная диффузионная модель, дополнительно обуславливаемая на векторизованное представление координат начального и целевого пикселей на изображении – `LDM-Cond`.

Для этих экспериментов использовались входные изображения из тестовой выборки размером 128 × 128 (т.е. задания были более сложными по сравнению со всеми предыдущими экспериментами). Отслеживались те же индикаторы качества, что и ранее – нормированная стоимость отыскиваемых решений (Cost Ratio) и нормированное число итераций алгоритма поиска (Expansions Ratio). Нормализация проводилась на результаты алгоритма `FS` при использовании эталонных PPM. Так же отдельный интерес представляло качество восстановления PPM по изображению. Для этого отслеживалась средняя квадратичная ошибка предсказания – MSE, – т.е. разница между предсказанной PPM и эталонной (идеальной) PPM, которая может быть построена в условиях наличия информации о DEM-канале. Агрегированные по всей тестовой выборке результаты представлены в Табл. 5.3.

Как и ранее, в таблице до знака ± указаны средние значения, после этого знака – стандартные отклонения.

Как видно из таблицы, применение более продвинутых нейросетевых моделей повышает качество предсказания PPM (см. последнюю строку в таблице). Так, значение среднеквадратичной ошибки предсказания для `GAN` лучше, чем для базовой модели, для `LDM` – лучше, чем `GAN`, а для `LDM-Cond` – лучше, чем для `LDM`. Последняя модель характеризуется наименьшей ошибкой предсказания (как по среднему значению, так и по разбросу значений). Это положитель-

ном образом влияет на, собственно, эффективность решения задач PF-RGB. Так LDM-Cond демонстрирует наилучшие результаты как по стоимости отыскиваемых решений (Cost Ratio), так и по числу итераций алгоритма поиска (Expansions Ratio). Стоит заметить, что последний показатель в этой серии экспериментов заметно выше (особенно по разбросу значений) по сравнению с предыдущей серией. Это можно объяснить тем, что в данных экспериментах использовались изображения заметно б*о*льшего размера, что, вполне ожидаемо, ведет к снижению эффективности поиска.

В целом, наблюдаемые результаты позволяют сделать однозначный вывод о том, что эффективность предлагаемого в работе метода решения задач PF-RGB, GBFS+PPM, может быть повышена за счет применения более продвинутых нейросетевых архитектур, используемых на этапе предсказания вероятностей вхождения элементов изображения в путь (PPM).

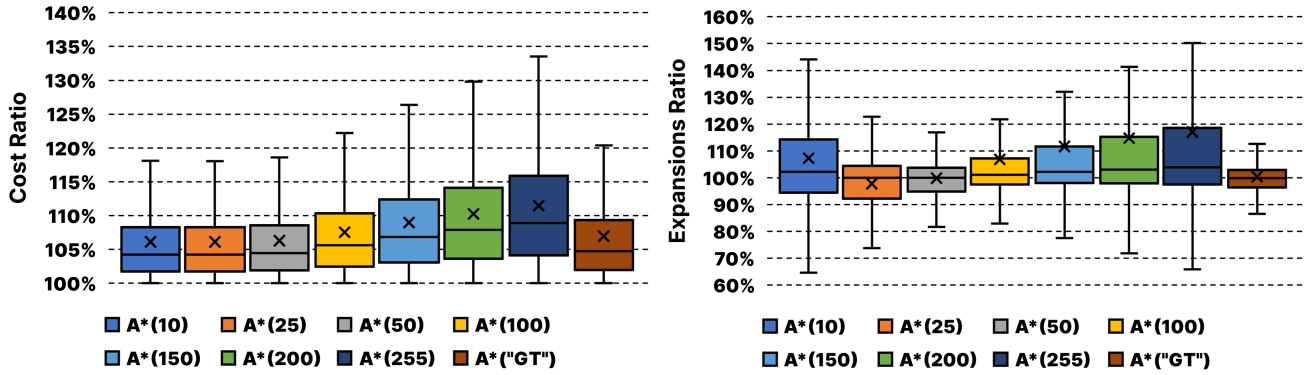
Третья серия экспериментов Целью этой серии экспериментов являлся анализ того, насколько предложенный метод планирования по изображениям, GBFS+PPM, соотносится с более прямолинейным обучаемым подходом, A*+DEM, при котором сначала нейросетевая модель предсказывает по изображению рельеф местности в виде карты высот, т.е. DEM-канал, а потом поиск по построенной модели рельефа осуществляется стандартным алгоритмом эвристического поиска A*.

Одной из особенностей такого подхода, является то, что при нём осуществляется предсказание DEM-канала, содержащего значения от 0 (минимальная высота) до 1 (максимальная высота). Для корректного задания весов переходов между вершинами графа, соответствующего модели рельефа, требуется подбор масштабирующего коэффициента α (см. формулу 5.1). Этот коэффициент необходим для того, чтобы соблюдалась корректная пропорция между слагаемыми функции стоимости, задающими как пройденное расстояние на плоскости, так и перепад высот.

В экспериментах данной серии использовались различные значения коэффициента α из диапазона 10, 25, 50, 100, 150, 200, 255. На графиках ниже метод A*+DEM, использующий коэффициент $\alpha = x$, будет обозначаться для краткости как A*(x). Помимо прочего в экспериментах использовалось эталонное значение коэффициента, соответствующего реальному масштабу и рассчитываемое по формуле $max(DEM) - min(DEM)$, где минимум и максимум берётся по

значениям эталонного ненормированного DEM-канала, а не восстановленного нейросетью. Такой вариант алгоритма будет обозначен далее как A*(GT) (от. англ. ground truth – истинное значение).

Результаты экспериментов представлены на Рис. 5.21.



а) Относительная стоимость решений. б) Относительное число итераций.

Рисунок 5.21 — Относительная стоимость решения и число итераций алгоритма, в зависимости от сложности задачи PF-RGB.

На приведенном выше рисунке изображены диаграммы размаха для показателей Cost Ratio (относительная стоимость решения) и Expansions Ratio (относительное число итераций алгоритма). Для получения этих характеристик результаты работы алгоритма A*(x), в каждом отдельном эксперимента нормируются на соответствующие результаты алгоритма A*, использующего эталонную карту высот для поиска (а не восстановленную нейросетью по изображению).

Можно заметить, что даже с наиболее подходящим коэффициентом масштаба, т.е. при использовании алгоритма A*(GT), не происходит существенного сокращения числа итераций поиска – см. левый столбец на левом графике на рисунке. Т.е. в среднем число итераций поиска такое же, как и при использовании стандартного алгоритма A*, т.е. в среднем относительное число раскрытий составляет 100%. А при неудачном выборе коэффициента α результаты ещё более ухудшаются – см., например, относительное число итераций для A*(10) или A*(100). При этом, как было показано ранее, поиск пути по предсказанной нейросетью карте вероятностей (PPM) требует гораздо меньшее число итераций (см. Табл. 5.3) – в среднем 44%-66% (в зависимости от используемой нейросетевой модели). Следовательно, можно сделать вывод о том, что с вычислительной точки зрения, предлагаемый в работе алгоритм GBFS+PPM заметно превосходит подход A*+DEM.

С точки зрения качества отыскиваемых решений, т.е. стоимости путей разница между GBFS+PPM и A*+DEM не такая значительная, т.к. последний характеризуется построением путей достаточно высокого качества, что может быть объяснено приемлемой точностью восстановления DEM-канала и экстенсивному поиску по нему (т.е. поиску с большим числом итераций, как показано ранее).

Качественный анализ отдельных решений – см. Рис. 5.22, – подтверждает эти наблюдения. На рисунке слева изображено исходное изображение, по которому необходимо осуществить поиск. Затем показано эталонная и восстановленная нейросетью карта высот (DEM-GT и DEM, соответственно). Далее показаны эталонная и предсказанная карта вероятностей вхождения пикселей в путь. Наконец показаны результаты поиска, полученные двумя различными алгоритмами: A*+DEM и GBFS+PPM. Зеленой заливкой показаны области, которые были исследованы алгоритмами поиска. Очевидно, что GBFS+PPM исследует гораздо меньший объем пространства поиска (и тем самым быстрей завершает свою работу) по сравнению A*+DEM. При этом алгоритмы сходятся к очень близким по стоимости решениям

Обобщая результаты этой и предыдущих серий экспериментов, можно сделать однозначный вывод о том, что предлагаемый в работе способ решения задачи PF-RGB с помощью предсказания карты вероятностей вхождения пикселей в путь и последующего запуска алгоритма фокусированного (фокального) поиска на этой карте, во-первых, демонстрирует высокую вычислительную эффективность по числу итераций алгоритма (это число сокращается по сравнению с использование классического алгоритма A*), во-вторых, характеризуется высоким качеством отыскиваемых решений (низкой стоимостью конструируемых путей), в-третьих, заметно превосходит альтернативный подходы – как алгоритм NeuralA*, так и алгоритм A*+DEM.

5.3.3 Дополнительные эксперименты

Помимо основных экспериментов, направленных на эмпирическое исследование предлагаемых в работе методов решения задач PF-G и PF-RGB и их сравнение с современными аналогами, в ходе работы был проведен ряд дополни-

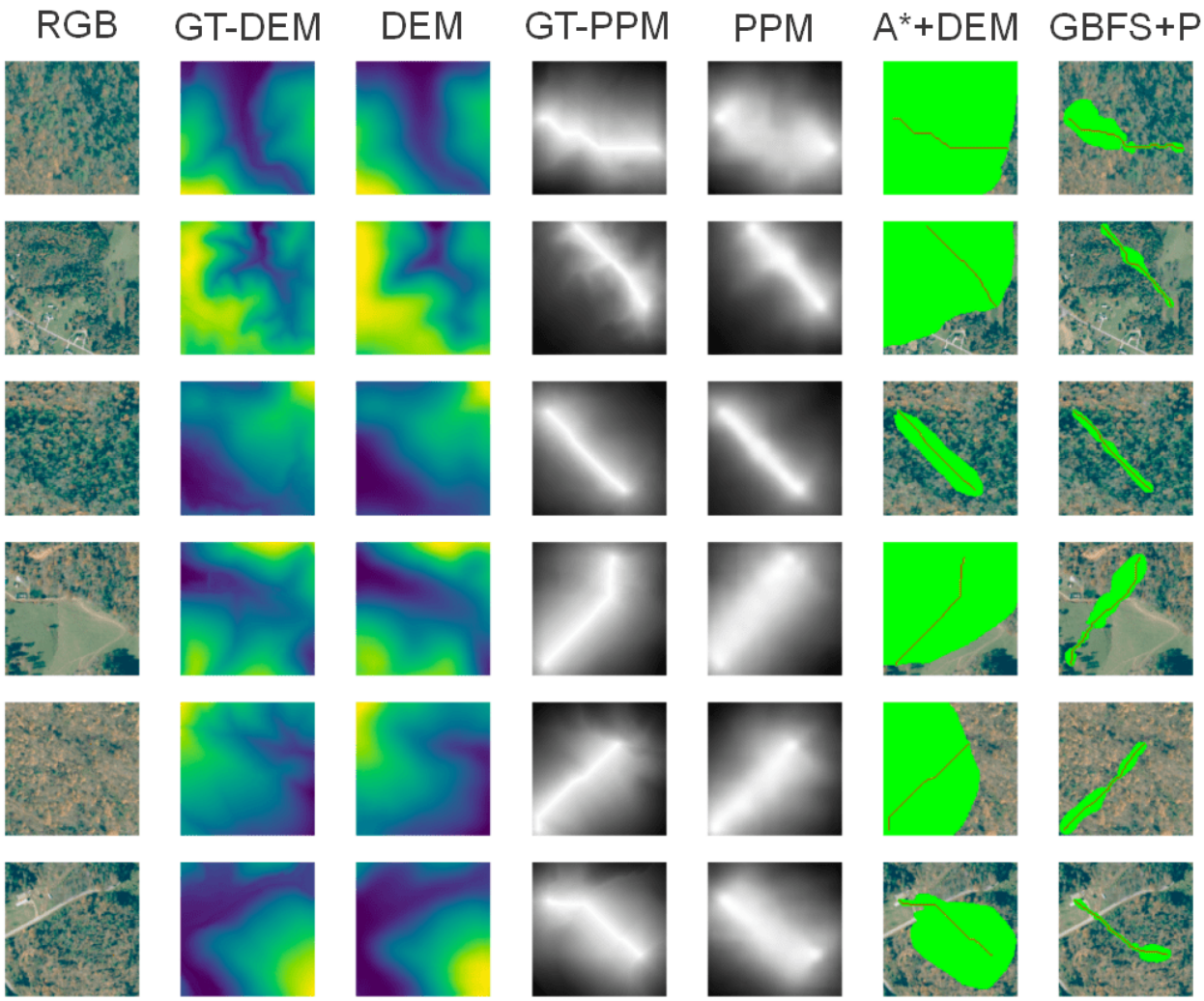


Рисунок 5.22 — Примеры решения задач PF-RGB.

тельных экспериментов, целью которых был более детальных анализ факторов, оказывающих влияние на эффективность предлагаемых решений и исследование возможных вариантов их альтернативного использования. Опишем далее эти эксперименты и их результаты.

Различные способы формирования эталонных PPM В Разделе 5.2.2 при описании способа формирования эталонной PPM (карты вероятности вхождения вершины ГРД в путь), приводился пример, демонстрирующий низкую эффективность подхода, основанного на поиске всех кратчайших путей алгоритмом $\texttt{A}^*$ для включения всех этих путей в PPM. Вместо этого было предложено использовать алгоритм $\texttt{Theta}^*$ для поиска единственного пути и дальнейшей фокусировке PPM вокруг этого пути. Для количественной оценки того, как предложенный способ влияет на эффективность решения задач PF-G был проведен следующий эксперимент.

Была создана дополнительная обучающая выборка эталонных PPM, построенная с помощью алгоритма A*. В этих PPM все вершины ГРД, принадлежащие хотя бы одному кратчайшему пути были отмечены как 1. Далее на этой выборке была обучена нейросетевая модель и повторен эксперимент по решению задач PF-G с помощью алгоритмов FS+PPM и GBFS+PPM. Результаты эксперимента приведены в Табл. 5.4.

Таблица 5.4 — Empirical results for the experiments involving different types of PPMs.

	Optimal Found Ratio (%) ↑	Cost Ratio (%) ↓	Expansions Ratio (%) ↓
FS-Theta*+PPM	82.97	**100.24 ± 0.74**	26.36 ± 21.08
GBFS+Theta*-PPM	83.02	100.25 ± 0.90	**23.60 ± 18.34**
FS-A*+PPM	61.20	103.6 ± 2.45	53.03 ± 28.73
GBFS-A*+PPM	60.06	103.9 ± 2.32	49.93 ± 27.81

Как и ранее в таблице приведены усредненные значения показателей Cost Ratio (относительная стоимость решений) и Expansions Ratio (относительное число итераций поиска), полученные по всей выборке тестовых заданий, и процент заданий, в которых было найдено оптимальное решение. Значения после знака ± показывают стандартное отклонение. Нормировка производится на результаты работы алгоритма A*. Первые две строки соответствуют алгоритмам, использующим PPM, которые построены по предлагаемому в работе способу с опорой на алгоритм Theta*. Последние две строки – алгоритмам, использующим PPM, которые построены с помощью A*.

Очевидно, что использование PPM, которые были построены с помощью алгоритма Theta*, приводит к более эффективному решению задач PF-G: стоимость отыскиваемых путей ниже, в то время как число итераций существенно ниже (до двух раз по сравнению с алгоритмами, которые используют нейросети, обученные на PPM, построенных с помощью A*). Эти результаты подтверждают целесообразность предлагаемого в работе способа построения эталонных PPM, в которых выделятся единственный путь, а не несколько (все) кратчайшие пути.

Ранее в Разделе 5.2.2 описывалось, что к эталонным PPM, предназначенным для решения задачи PF-G применялись дополнительные методы пост-обработки: возведение *pp*-значений в степень и фильтрация (обнуление) значений, не превышающих определенный порог. Обе эти техники приводят

к эффекту повышения контрастности эталонной PPM, который был проиллюстрирован ранее на Рис. 5.13.

Для количественной оценки эффекта от предлагаемой пост-обработки эталонных PPM были проведены отдельные эксперименты, в рамках которых нейросетевая модель обучалась на различных эталонных PPM и далее использовалась в алгоритме `GBFS+PPM` для решения задач PF-G. Результаты этих экспериментов приведены в Табл 5.5.

Таблица 5.5 — Влияние различных вариантов пост-обработки PPM (возведение *pp*-значений в степень, и отбрасывание значений ниже определенного порога) на эффективность последующего решения задач PF-G.

	Cost Ratio (%) ↓		Expansion Ratio (%) ↓	
clipping value	power 1	power 10	power 1	power 10
0	101 ± 1	100.3 ± 0.4	68 ± 35	28 ± 25
0.3	101 ± 1	100.3 ± 0.4	71 ± 40	30 ± 22
0.6	102 ± 1.2	100.4 ± 0.6	75 ± 37	35 ± 21
0.9	101 ± 0.6	101 ± 1.2	79 ± 41	32 ± 22
0.95	107 ± 5	**100.2 ± 0.9**	57 ± 23	**23 ± 18**

Каждая ячейка в таблице соответствует числу итераций алгоритма `GBFS+PPM` (нормированному, как обычно, на число итераций алгоритма `A`*) либо стоимости решения (аналогично нормированной). Результаты однозначно свидетельствуют о том, что предлагаемые в работе техники фокусировки PPM (возведение *pp*-значений в степень и отбрасывание значений ниже определенного порога) дают ощутимый прирост в эффективности решения задач PF-G. При это основной прирост наблюдается от возведения в степень, в то время как техника фильтрации, оказывает чуть меньшее влияние. Наибольший же прирост достигается в случае комбинации этих техник.

Заметим, что в отличие от формирования PPM для последующего решения задач PF-G, при формировании эталонных PPM для задач PF-RGB, указанные выше техники фокусировки PPM не повышают эффективность поиска – см. Табл. 5.6, построенную в результате проведения эксперимента, аналогичного вышеописанному, но для задач поиска пути по изображениям.

Из приведенных результатов кажется, что техника отбрасывания значений ниже определенного порога дает прирост эффективности – см., например, как снижается число итераций при повышении порога (clipping value) в случае,

Таблица 5.6 — Влияние различных вариантов пост-обработки PPM (возведение *pp*-значений в степень, и отбрасывание значений ниже определенного порога) на эффективность последующего решения задач PF-RGB.

	Cost Ratio (%) ↓		Expansion Ratio (%) ↓	
clipping value	power 1	power 10	power 1	power 10
0	106 ± 10	136 ± 39	44 ± 32	21 ± 24
0.3	109 ± 14	130 ± 44	49 ± 34	16 ± 11
0.6	110 ± 12	135 ± 31	47 ± 31	20 ± 19
0.9	151 ± 49	141 ± 39	18 ± 15	15 ± 14
0.95	146 ± 53	133 ± 42	21 ± 19	25 ± 31

когда *pp*-значений не возводятся в степень (power 1) – с 44% до 21%. Однако стоит обратить внимание как при этом ухудшаются показатели качества отыскиваемых решений (Cost Ratio) – стоимость построенных путей существенно возрастает: с 106% до 146%. Фактически, пути вырождаются в прямые линии, соединяющие начальные и целевые пиксели, которые никак не учитывают перепады высот. Такой же эффект дает техника возведения в степень при отсутствии клиппирования (отбрасывания значений ниже порога). В целом, наиболее выгодный баланс между скоростью работы алгоритма (число итераций) и качеством решений (стоимость пути) достигается, когда никакая пост-обработка PPM не используется.

Предоставление строгого объяснения, почему методы возведения в степень и отсечения не являются полезными для задач поиска пути на изображениям, PF-RGB, в то время как они эффективны для поиска пути на ГРД, т.е. при решении задач PF-G, затруднено из-за принципиальных различий между этими задачами. Так, при поиске пути на ГРД, стоимости переходов для любой пары ортогонально/диагонально смежных вершин одинаковы, поэтому *pp*-значения не должны учитывать разницу в этих переходах, в то время как в поиске пути на изображениях стоимости переходов различны из-за различий в высотах. Это, возможно, объясняет, почему возведение PPM в степень не является полезным для поиска пути на изображениях – оно косвенно искажает стоимости переходов между пикселями и усложняет задачу неявного восстановления этих стоимостей в форме PPM.

Далее, при поиске пути на ГРД предсказание *pp*-значения для всех вершин равносильно предсказанию степень близости к оптимальному пути для любой

из них, что может быть нерелеватно задаче, собственно, формирования пути. Таким образом, отсечение низких *pp*-значений повышает качество предсказаний за счет того, что нейросетевая модель "концентрируется" на наиболее важных вершинах. С другой стороны, при поиске пути на изображениях все ячейки потенциально проходимы, а низкие *pp*-значения представляют области, которых следует избегать из-за высокой сложности прохождения (большого перепада высот). Отсечение этих значений удаляло бы ценную информацию, затрудняя для нейронной сети обучение предсказанию степени проходимости вершины, что влияет на качество пути. Следовательно, возможно, что в поиске пути на ГРД наиболее важной задачей является понимание того, где расположен путь, в то время как в поиске пути на изображениях необходимо принимать взвешенное решение относительно каждого отдельного пикселя – с высокой вероятностью его следует избегать.

Озвученные аргументы не являются строгими и носят гипотетический характер. Тем не менее, результаты экспериментов дают неоспоримые доказательства того, что, как возведение *pp*-значений в степень, так и их отсечение, негативно влияют на эффективность алгоритма `GBFS+PPM` при решении задач PF-RGB (при том что при решении задач PF-G картина обратная).

Решение задач вне исходной выборки Предложенные в работе методы решения задач PF-G и PF-RGB предполагают наличие выборки заданий, часть их которых используется для обучения нейросетвой модели предсказанию эвристик, использование которых затем повышает эффективность решения оставшейся части задач. При этом возникает вопрос, как эффективно предлагаемый подход будет справляться с решением задач, которые существенно отличаются от представленных в обучающей (и тестовой) выборке. В англоязычной литературе по машинному обучению такие задачи носят название out-of-distribution (OOD), т.е. буквально "задачи вне (исходного) распределения (задач)".

Для проверки того, как предлагаемые алгоритмы способы справлять с OOD-задачами был проведен эксперимент по решению задач PF-G на ГРД, отличных от тех, что составляли исходную выборку. Для этого эксперимента использовались три карты из известной в области планирования траектории коллекции заданий MovingAI [219]: `Berlin_1` (городская карта), `maze512-32-0`

(лабиринт) и `BigGameHunters` (карта из компьютерной игры в жанре “стратегия реального времени”) – см. Рис. 5.23.

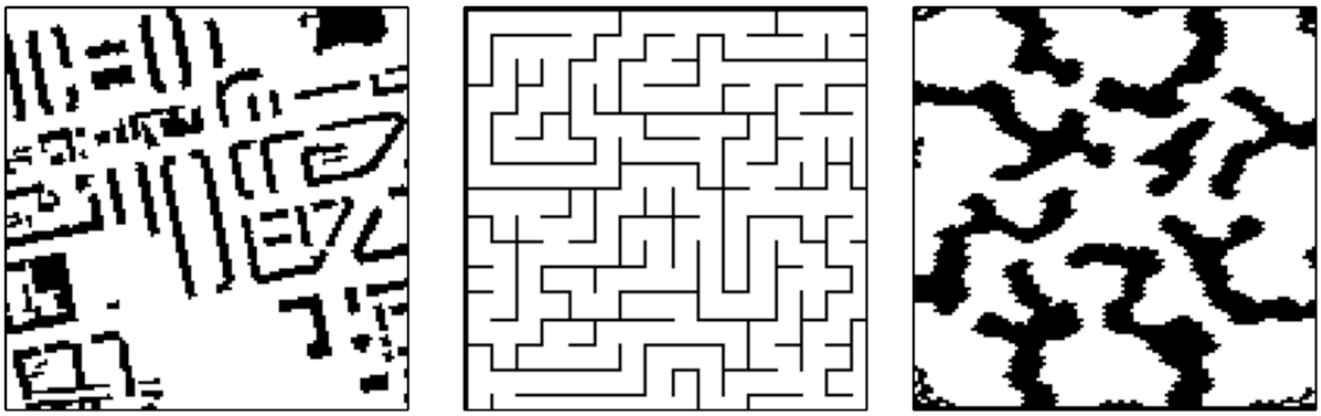


Рисунок 5.23 — Графы регулярной декомпозиции, используемые для дополнительного экспериментального исследования алгоритмов решения задач PF-G.

Каждая из этих карт была приведена к двум размерам, а именно: 64×64 и 128×128. Далее для каждой карты было сгенерировано случайным образом 1 000 заданий (пар старт-финиш). На полученной коллекции были протестированы предлагаемые в работе алгоритмы (и их аналоги) по той же методологии, что и основные эксперименты. Полученные результаты представлены в Табл. 5.7.

Таблица 5.7 — Результаты экспериментального исследования алгоритмов решения задач PF-G на задачах из OOD-выборки.

	Optimal Found Ratio (%) ↑	Cost Ratio (%) ↓	Expansions Ratio (%) ↓
A*	100	100	100
WA*	8.13	104.31 ± 4.76	57.52 ± 30.72
NeuralA*	3.24	107.10 ± 6.77	63.08 ± 34.63
A*+HL	**29.02**	**101.90 ± 2.72**	148.94 ± 136.95
WA*+CF	10.61	106.10 ± 5.59	63.64 ± 36.31
FS+PPM	18.66	105.62 ± 5.61	55.06 ± 39.57
GBFS+PPM	18.59	106.12 ± 6.54	**54.33 ± 47.24**

Как и следовало ожидать эффективность методов, использующих обучаемые эвристики (как предлагаемых в работе, так и аналогичных), снижается.

Такой вывод можно сделать сравнив приведенные показатели эффективности с результатами основного эксперимента, описанного в Разделе 5.3.1 (см. Табл. 5.1). Тем не менее, предлагаемые в работе подходы, `WA*+CF`, `FS+PPM`, `GBFS+PPM`, превосходят обучаемые аналоги, `A*+HL`, `NeuralA*` по эффективности. Так наименьшее число итераций наблюдается у алгоритма `GBFS+PPM`, как и ранее, а а наилучший баланс между скоростью работы и качество решения, среди обучаемых планировщиков, демонстрирует `FS+PPM`.

Влияние блоков внимания на эффективность работы Основная архитектура нейросетевой модели, используемой для предсказания предлагаемых в работе эвристических функций, содержит блоки внимания, которые расположены между свёрточными блоками энкодера и декодера – см. Рис. 5.14. Для того, чтобы выяснить, насколько наличие этих блоков важно с точки зрения эффективности решения исследуемых задач был проведен дополнительный эксперимент. В этом эксперименте была создана копия исходной нейросетевой модели без блоков внимания, т.е. содержащая лишь свёрточные слои (CNN). Эта модель была обучена по тому же сценарию, что и модель с блоками внимания, и затем было проведено их экспериментальное сравнительное исследование в решении задач PF-G. Это исследование проводилось по той же методологии, что и основные эксперименты, с единственной особенностью – использовались лишь наиболее сложные задания тестовой выборки (задания со сложностью, превышающей 1.5), т.к. именно на таких заданиях вероятнее всего проявится разница между наличием и отсутствием блоков внимания (т.к. по выдвинутой ранее в работе гипотезе, блоки внимания позволяют устанавливать корректные взаимосвязи между препятствиями сложной конфигурации).

Результаты эксперимента для алгоритма `FS+PPM` представлены в Табл. 5.8 (для других подходов, а именно – `GBFS+PPM`, `WA*+CF`, результаты аналогичные).

Таблица 5.8 — Результаты экспериментального исследования алгоритма `FS+PPM`, использующего нейросетевые архитектуры с и без блоков внимания на этапе предсказания PPM.

	w/ Attn	w/o Attn
Optimal Found Ratio (%)	**85.22**	61.74
Average Cost Ratio (%)	**100.31 ± 1.58**	101.12 ± 2.19
Average Expansions Ratio (%)	**16.06 ± 11.57**	19.65 ± 17.03

Как видно из таблицы, использование блоков внимания в архитектуре нейросети (столбец w/ Attn) заметно повышает эффективность планирования по сравнению с архитектурой основанной лишь на сверточных слоях (столбец w/o Attn).

На Рис. 5.24 приведен пример решения одной из задач PF-G, планировщиками, использующими различные нейросетевых архитектуры для предсказания PPM: с и без блоков внимания.

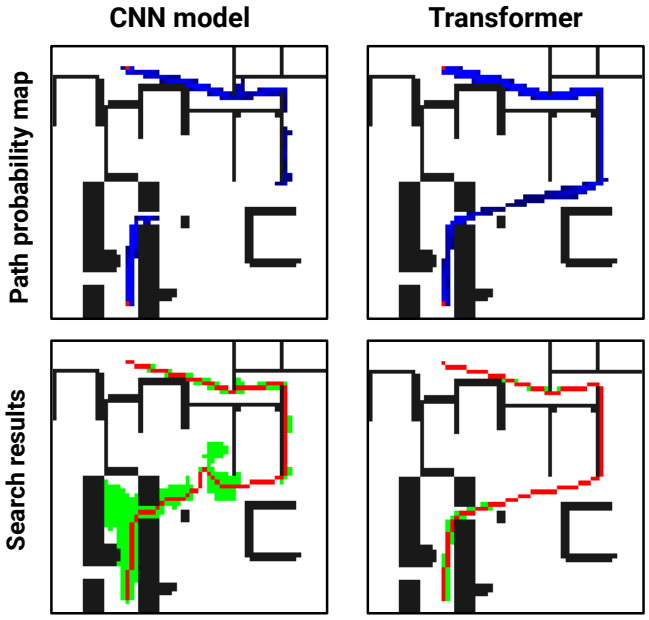


Рисунок 5.24 — Пример, демонстрирующий разницу в решении задачи PF-G при использовании различных нейросетевых архитектур на этапе предсказания PPM.

Сформированная PPM показана на рисунке в верхней части, а в нижней изображен результат решения задачи PF-G: найденный путь изображен красным цветом, а исследованное пространство поиска – зеленым. Рисунок косвенно подтверждает выдвинутую в работе гипотезу о том, что сверточные слои нейросети позволяют выделить особенности изображения, а блоки внимания определить релевантные взаимосвязи между ними. В данном конкретном примере, использование блоков внимание позволяет сформировать PPM, в которой корректно отражено, что для построения искомого пути нужно переместиться из узкого прохода в левой части карты к определенному углу препятствия, расположенного в правой части. При этом при отсутствии блоков внимания (т.е. использования лишь сверточных слоев) формируется PPM, содержащая ”раз-

рыв" из-за чего поиск с использованием такой PPM становится гораздо менее эффективным.

Исключительно нейросетевое планирование Предложенные в работе методы решения задач PF-G и PF-RGB опираются на комбинацию нейросетевого предсказания особых эвристических функций, pp-значений и cf-значений (образующих в совокупности PPM и CFM), с последующей интеграцией этих эвристик в алгоритмы систематического поиска на графе. Возникает естественный вопрос – можно ли использовать предсказанные PPM (CFM) более простым и быстрым способом для конструирования итогового решения, т.е. искать "напрямую" по PPM (CFM).

Для ответа на этот вопрос был проведен эксперимент со следующим решателем. На первом этапе как и ранее предсказывалась карта вероятности вхождения вершины в путь – PPM. Далее, осуществлялся простой жадный поиск по этой PPM, т.е. на каждой итерации в путь добавлялась вершина а) смежная с текущей, б) не добавленная в путь ранее, в) имеющая наиболее высокое pp-значение. Алгоритм начинает обработку с начальной вершины и завершается либо при достижении целевой вершины, либо когда на очередной итерации невозможно выбрать элемент, который еще не был добавлен в путь. Результаты этого эксперимента представлены в Таблице 5.9.

Таблица 5.9 — Результаты экспериментального исследования алгоритма решения задач PF-G, опирающегося исключительно на построенную PPM.

Success Rate (%)	Cost Ratio (%)	Expansions Ratio (%)
87.80	143.07±72.86	27.59±34.34

Первый столбец Таблицы 5.9 показывает процент успешно решенных задач из тестовой выборки. Как можно заметить, 12.2% экземпляров остались нерешенными. Более того, как указано во втором столбце, стоимость полученных путей значительно выше по сравнению как с оптимальными путями, так и с путями, полученными с помощью `FS+PPM` или `GBFS+PPM` (эти планировщики находят пути со средней относительной стоимостью 100.25% – см. Табл. 5.1). Последний столбец показывает, сколько итераций (в среднем) потребовалось предложенному жадному алгоритму для восстановления путей по сравнению

с количеством итераций, используемых A*. Безусловно, жадный поиск характеризуется гораздо меньшим числом итераций по сравнению с A* и в целом соответствует по производительности FS+PPM или GBFS+PPM, для которых значений этого показателя составляло 26% и 23% соответственно (см. Табл. 5.1).

В целом можно заключить, что рассмотренный подход, который опирается на примитивный алгоритм поиска по построенным нейросетевой моделью эвристикам, значительно менее эффективен, чем предложенные в работе методы. Это подтверждает, что эффективность последних обуславливается как использование предложенных в работе эвристических функций (в частности – PPM), так и предложенными способами интеграцией этих функций с продвинутыми алгоритмами систематического эвристического поиска.

5.4 Выводы по главе

В этой главе рассматривались два типа задач поиска пути – классическая задача поиска пути на (статическом) ГРД, и задача поиска пути по изображению. В обоих случаях предполагалось, что имеется некоторая (достаточно большая) выборка различных экземпляров задач, часть из которых может использоваться для предобработки и извлечения полезной информации из этих данных для последующего использования (решения оставшегося множества задач). Был предложен новый подход, основанный на комбинации известных алгоритмов систематического поиска и оригинальных, т.е. впервые предложенных в работе, эвристических функциях, алгоритм формирования которых автоматически конструируется с помощью методов машинного обучения. В частности для аппроксимации эвристик предлагается использовать глубокое обучение с учителем и современные нейросетевые архитектуры, содержащие блоки внимания (трансформеры). На основе предложенного подхода был предложен ряд гибридных алгоритмов поиска, т.е. использующих одновременно классические алгоритмические компоненты и обучаемые процедуры. Отличительной особенностью разработанных алгоритмов является гарантия полноты, т.е. гарантия того, что для любой решаемой задачи будет сформирован корректный ответ, а если у задачи нет решения, то алгоритм корректно завершится и вернет специальный символ, свидетельствующий об отсутствии решения. Более того, один

из алгоритмов позволяет гарантировать построение решений, качество которых не превосходит качество оптимальных решений (отыскиваемых классическими алгоритмами поиска), более чем в заданное число раз. Были разработаны авторские коллекции (выборки) данных для обучения и проведения экспериментов. Было проведено обширное экспериментальное исследование, показавшее высокую эффективность предлагаемых методов при решении рассматриваемых задач и их превосходство над имеющимися современными мировыми аналогами по различным показателям эффективности, таким как качество решений (длина отыскиваемых путей) и скорость работы (число итераций алгоритма поиска). Были проведены дополнительные экспериментальные исследования, с помощью которых была подтверждена обоснованность ряда технический решений, использованных при разработке предлагаемых алгоритмов.

Заключение

Диссертация посвящена исследованию и разработке эффективных методов и алгоритмов поиска пути (совокупности путей) на графах особой структуры, вложенных в метрическое пространство, в том числе методов, использующих современные нейросетевые модели для аппроксимации информативных эвристических функций, использование которых позволяет существенно сократить перебор и повысить вычислительную эффективность поиска, при этом сохранить гарантии отыскания корректного и, в ряде случаев, ограниченно суб-оптимального решения. Указанные методы и алгоритмы представляют практический интерес, т.к. применимы для решения множества прикладных робототехнических задач, где требуется автоматическое планирование траектории для отдельных роботов или построение совокупности неконфликтных путей для групп роботов, функционирующих в общей среде.

Основные результаты работы состоят в следующем:

1. Разработан новый метод, и алгоритм его реализующий, поиска пути на динамическом графе регулярной декомпозиции, допускающий переходы между произвольными вершинами, что позволяет сократить длину пути и время достижения цели. Метод основан на принципе безопасно-интервального планирования и использует оригинальный подход к обращению направления поиска для сокращения вариантов перебора в ходе поиска. Предложенный метод применим для планирования траектории мобильного агента (робота) в среде с динамическими препятствиями. При этом, в отличие от аналогов, он позволяет агенту перемещаться в произвольном направлении, что повышает качество решений. Проведено теоретическое исследование алгоритма: установлены его свойства, доказана гарантия нахождения пути при его существовании (или корректное завершение в противном случае), а также оптимальность решения по критерию времени достижения целевой позиции.
2. Предложен метода и алгоритм поиска пути на динамическом графе регулярной декомпозиции, использующий эвристическую процедуру изменения родительского узла в дереве поиска. Это позволило существенно сократить пространство поиска, при незначительном (на

практике – менее 3%) ухудшении качества решения. Проведено теоретическое исследование алгоритма и доказаны следующие свойства: гарантия нахождения пути при его наличии в исходной топологии графа и то, что стоимость найденного пути не превышает стоимости оптимального пути в исходной топологии.

3. Разработан ряд новых вычислительно-эффективных субоптимальных методов поиска пути на динамическом графе регулярной декомпозиции. Методы основаны на применении решетки переходов и регулируемого коэффициента субоптимальности. Проведены их экспериментальные исследования.
4. Предложен новый метод и алгоритм построения совокупности неконфликтных траекторий на графе регулярной декомпозиции, основанный на принципе конфликтно-ориентированного поиска. Предложены техники сокращения пространства поиска, повышающие вычислительную эффективность без потери гарантий оптимальности.
5. Разработан новый алгоритм приоритизированного многоагентного планирования. В отличие от аналогов, метод позволяет агентам выполнять действия произвольной продолжительности и перемещаться между любыми вершинами графа, что значительно улучшает качество решений (сокращается суммарная длина траекторий). Предложены оригинальные эвристики для повышения практической эффективности. Проведено эмпирическое исследование алгоритма, в том числе на реальных роботах.
6. Предложено семейство алгоритмов поиска на графе регулярной декомпозиции, допускающих перемещение между произвольными вершинами и учитывающих геометрические ограничения, в частности – ограничение на максимальный угол поворота между смежными прямолинейными сегментами траектории. Таким образом, становится возможен косвенный учет кинематических ограничений реальных мобильных агентов (роботов) на этапе планирования маршрута. Исследованы теоретические свойства разработанных алгоритмов, сформулированы и доказаны гарантии нахождения решений в заданном классе решений. Проведено обширное эмпирическое исследование методов.

7. Предложены новые типы эвристических функций для поиска пути на графе регулярной декомпозиции, учитывающие специфику отдельного экземпляра задачи и эффективно аппроксимируемые современными нейросетевыми моделями. Комбинация этих функций с классическими алгоритмами эвристического поиска существенно повышает их практическую эффективность, что подтверждено многочисленными вычислительными экспериментами.
8. Разработанный подход к решению задачи поиска пути, сочетающий обучаемые эвристики и классические алгоритмы, успешно применен для планирования траектории по изображению (спутниковому снимку местности со сложным рельефом). Сравнительное исследование показало существенное преимущество предлагаемого метода над существующими мировыми аналогами.

Список литературы

1. *Ghosh S. K.*, *Mount D. M.* An output-sensitive algorithm for computing visibility graphs [Текст] // SIAM Journal on Computing. — 1991. — Т. 20, № 5. — С. 888—910.
2. *Takahashi O.*, *Schilling R. J.* Motion planning in a plane using generalized Voronoi diagrams [Текст] // IEEE Transactions on robotics and automation. — 1989. — Т. 5, № 2. — С. 143—150.
3. *Kavraki L.*, *Svestka P.*, *Latombe J.-C.*, *Overmars M.* Probabilistic roadmaps for path planning in high-dimensional configuration spaces [Текст] // IEEE Transactions on Robotics and Automation. — 1996. — Т. 12, № 4. — С. 566—580.
4. *Yap P.* Grid-based path-finding [Текст] // Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence. — 2002. — С. 44—55.
5. *Яковлев К. С.*, *Баскин Е. С.* Графовые модели в задаче планирования траектории на плоскости [Текст] // Искусственный интеллект и принятие решений. — 2013. — № 1. — С. 5—12.
6. *Rivera N.*, *Hernández C.*, *Hormazábal N.*, *Baier J. A.* The 2^ k Neighborhoods for Grid Path Planning [Текст] // Journal of Artificial Intelligence Research. — 2020. — Т. 67. — С. 81—113.
7. *Nash A.*, *Daniel K.*, *Koenig S.*, *Felner A.* Theta*: Any-Angle Path Planning on Grids [Текст] // Proceedings of The 22nd AAAI Conference on Artificial Intelligence (AAAI 2007). — 2007. — С. 1177—1183.
8. *Dijkstra E. W.* A note on two problems in connexion with graphs [Текст] // Numerische mathematik. — 1959. — Т. 1, № 1. — С. 269—271.
9. *Bellman R.* On a routing problem [Текст] // Quarterly of applied mathematics. — 1958. — Т. 16, № 1. — С. 87—90.
10. *Ford L. R. J.*, *Fulkerson D. R.* Flows in Networks [Текст]. — Princeton, NJ : Princeton University Press, 1962.

11. *Hart P. E.*, *Nilsson N. J.*, *Raphael B.* A formal basis for the heuristic determination of minimum cost paths [Текст] // IEEE transactions on Systems Science and Cybernetics. — 1968. — Т. 4, № 2. — С. 100—107.

12. *Pearl J.*, *Kim J. H.* Studies in semi-admissible heuristics [Текст] // IEEE transactions on pattern analysis and machine intelligence. — 1982. — № 4. — С. 392—399.

13. *Pearl J.* Heuristics: intelligent search strategies for computer problem solving [Текст]. — Addison-Wesley Longman Publishing Co., Inc., 1984.

14. *Pohl I.* Heuristic search viewed as path finding in a graph [Текст] // Artificial intelligence. — 1970. — Т. 1, № 3/4. — С. 193—204.

15. *Harabor D.*, *Grastien A.* Online graph pruning for pathfinding on grid maps [Текст] // Proceedings of the 25th AAAI conference on artificial intelligence (AAAI 2011). — 2011. — С. 1114—1119.

16. *Botea A.*, *Müller M.*, *Schaeffer J.* Near optimal hierarchical path-finding. [Текст] // Journal of Game Development. — 2004. — Т. 1, № 1. — С. 1—30.

17. *Likhachev M.*, *Stentz A.* R* Search [Текст] // Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008). — AAAI Press, 2008. — С. 344—350. — URL: http://www.aaai.org/Library/AAAI/2008/aaai08-054.php.

18. *Яковлев К. С.* HGA*: эффективный алгоритм планирования траектории на плоскости [Текст] // Искусственный интеллект и принятие решений. — 2010. — № 2. — С. 16—25.

19. *Takahashi T.*, *Sun H.*, *Tian D.*, *Wang Y.* Learning heuristic functions for mobile robot path planning using deep neural networks [Текст] // Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS 2019). — 2019. — С. 764—772.

20. *Skrynnik A.*, *Andreychuk A.*, *Yakovlev K.*, *Panov A.* Pathfinding in stochastic environments: learning vs planning [Текст] // PeerJ Computer Science. — 2022. — Т. 8. — e1056.

21. *Koenig S.*, *Likhachev M.* D* lite [Текст] // Proceedings of the 18th AAI Conference on Artificial Intelligence (AAAI 2002). — 2002. — С. 476—483.

22. *Otte M.*, *Frazzoli E.* RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning [Текст] // The International Journal of Robotics Research. — 2016. — Т. 35, № 7. — С. 797—822.

23. *Angulo B.*, *Panov A.*, *Yakovlev K.* Policy optimization to learn adaptive motion primitives in path planning with dynamic obstacles [Текст] // IEEE Robotics and Automation Letters. — 2022. — Т. 8, № 2. — С. 824—831.

24. *Zhao W.*, *Zhang Y.*, *Xie Z.* EPPE: An Efficient Progressive Policy Enhancement framework of deep reinforcement learning in path planning [Текст] // Neurocomputing. — 2024. — Т. 596. — С. 127958.

25. *Chen L.*, *Wu P.*, *Chitta K.*, *Jaeger B.*, *Geiger A.*, *Li H.* End-to-end autonomous driving: Challenges and frontiers [Текст] // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 2024.

26. *Stern R.*, *Sturtevant N. R.*, *Felner A.*, *Koenig S.*, *Ma H.*, *Walker T. T.*, *Li J.*, *Atzmon D.*, *Cohen L.*, *Kumar T. S.* [и др.]. Multi-agent pathfinding: Definitions, variants, and benchmarks [Текст] // Proceedings of the 12th Annual Symposium on Combinatorial Search (SoCS 2019). — 2019. — С. 151—158.

27. *Damani M.*, *Luo Z.*, *Wenzel E.*, *Sartoretti G.* PRIMAL _2: Pathfinding Via Reinforcement and Imitation Multi-Agent Learning – Lifelong [Текст] // IEEE Robotics and Automation Letters. — 2021. — Т. 6, № 2. — С. 2666—2673.

28. *Ma Z.*, *Luo Y.*, *Ma H.* Distributed heuristic multi-agent path finding with communication [Текст] // 2021 IEEE International Conference on Robotics and Automation (ICRA 2021). — IEEE. 2021. — С. 8699—8705.

29. *Andreychuk A.*, *Yakovlev K.*, *Panov A.*, *Skrynnik A.* MAPF-GPT: Imitation learning for multi-agent pathfinding at scale [Текст] // Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2025). — 2025. — С. 23126—23134.

30. *Van Den Berg J.*, *Guy S.*, *Lin M.*, *Manocha D.* Reciprocal n-body collision avoidance [Текст] // Robotics research. — 2011. — С. 3—19.

31. *Dergachev S.*, *Yakovlev K.* Distributed multi-agent navigation based on reciprocal collision avoidance and locally confined multi-agent path finding [Текст] // Proceedings of the 17th International Conference on Automation Science and Engineering (CASE 2021). — IEEE. 2021. — С. 1489—1494.

32. *Van Nieuwstadt M. J.*, *Murray R. M.* Real-time trajectory generation for differentially flat systems [Текст] // International Journal of Robust and Nonlinear Control. — 1998. — Т. 8, № 11. — С. 995—1020.

33. *Григоренко Н. Л.*, *Анисимов А. В.*, *Лукьянова Л. Н.* Построение терминального управления для системы второго порядка при наличии фазовых ограничений [Текст] // Труды Института математики и механики УрО РАН. — 2014. — Т. 20, № 4. — С. 97—105.

34. *Белинская Ю.*, *Четвериков В.* Метод накрытий для терминального управления и орбитальная декомпозиция систем [Текст] // Дифференциальные уравнения. — 2018. — Т. 54, № 4. — С. 502—502.

35. *Chen M.*, *Herbert S. L.*, *Vashishtha M. S.*, *Bansal S.*, *Tomlin C. J.* Decomposition of reachable sets and tubes for a class of nonlinear systems [Текст] // IEEE Transactions on Automatic Control. — 2018. — Т. 63, № 11. — С. 3675—3688.

36. *Karaman S.*, *Frazzoli E.* Sampling-based algorithms for optimal motion planning [Текст] // The international journal of robotics research. — 2011. — Т. 30, № 7. — С. 846—894.

37. *Sakcak B.*, *Bascetta L.*, *Ferretti G.*, *Prandini M.* Sampling-based optimal kinodynamic planning with motion primitives [Текст] // Autonomous Robots. — 2019. — Т. 43, № 7. — С. 1715—1732.

38. *Tamar A.*, *Wu Y.*, *Thomas G.*, *Levine S.*, *Abbeel P.* Value iteration networks [Текст] //. — 2016.

39. *Bhardwaj M.*, *Choudhury S.*, *Scherer S.* Learning heuristic search via imitation [Текст] // Proceedings of the 1st Conference on Robot Learning (CoRL 2017). — 2017. — С. 271—280.

40. *Yonetani R.*, *Taniai T.*, *Barekatain M.*, *Nishimura M.*, *Kanezaki A.* Path planning using neural A* search [Текст] // Proceedings of the 38th International Conference on Machine Learning (ICML 2021). — PMLR. 2021. — С. 12029—12039.

41. *Stentz A.* The focussed D* algorithm for real-time replanning [Текст] // Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995). — 1995. — С. 1662—1669.

42. *Koenig S., Likhachev M., Furcy D.* Lifelong planning A* [Текст] // Artificial Intelligence. — 2004. — Т. 155, № 1. — С. 93—146.

43. *Likhachev M., Gordon G. J., Thrun S.* ARA* : Anytime A* with Provable Bounds on Sub-Optimality [Текст] // Advances in Neural Information Processing Systems 16 (NeurIPS 2003) / под ред. S. Thrun, L. K. Saul, B. Schölkopf. — MIT Press, 2003. — С. 767—774. — URL: http://papers.nips.cc/paper/2382-ara-anytime-a-with-provable-bounds-on-sub-optimality.pdf.

44. *Holte R. C., Felner A., Sharon G., Sturtevant N. R., Chen J.* MM: A bidirectional search algorithm that is guaranteed to meet in the middle [Текст] // Artificial Intelligence. — 2017. — Т. 252. — С. 232—266.

45. *Thayer J. T., Ruml W.* Bounded suboptimal search: A direct approach using inadmissible estimates [Текст] // Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI) 2011). — 2011. — С. 674—679.

46. *Surynek P.* An optimization variant of multi-robot path planning is intractable [Текст] // Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010). — 2010. — С. 1261—1263.

47. *Standley T. S.* Finding optimal solutions to cooperative pathfinding problems [Текст] // Proceedings of The 24th AAAI Conference on Artificial Intelligence (AAAI 2010). — 2010. — С. 173—178.

48. *Sharon G., Stern R., Felner A., Sturtevant. N. R.* Conflict-based search for optimal multiagent path finding [Текст] // Artificial Intelligence Journal. — 2015. — Т. 218. — С. 40—66.

49. *Wagner G., Choset H.* Subdimensional expansion for multirobot path planning [Текст] // Artificial intelligence. — 2015. — Т. 219. — С. 1—24.

50. *Sharon G., Stern R., Goldenberg M., Felner A.* The increasing cost tree search for optimal multi-agent pathfinding [Текст] // Artificial intelligence. — 2013. — Т. 195. — С. 470—495.

51. *Ерусалимский Я.*, *Скороходов В. А.* Графы с вентильной достижимостью. Марковские процессы и потоки в сетях [Текст] // Известия высших учебных заведений. Северо-Кавказский регион. Естественные науки. — 2003. — № 3. — С. 1—6.

52. *Скороходов В. А.*, *Ерусалимский Я. М.* Локальное управление потоками в ресурсных сетях с малым ресурсом [Текст] // Актуальные проблемы прикладной математики, информатики и механики. — 2022. — С. 1671—1677.

53. *Кузнецов О. П.* Однородные ресурсные сети I. Полные графы [Текст] // Автоматика и телемеханика. — 2009. — № 11. — С. 136—147.

54. *Жилякова Л. Ю.* Несимметричные ресурсные сети. I. Процессы стабилизации при малых ресурсах [Текст] // Автоматика и телемеханика. — 2011. — № 4. — С. 133—143.

55. *Жилякова Л. Ю.* Управление предельными состояниями в поглощающих ресурсных сетях [Текст] // Проблемы управления. — 2013. — № 3. — С. 51—59.

56. *Агаев Р. П.*, *Чеботарев П. Ю.* Матрица максимальных исходящих лесов орграфа и ее применения [Текст] // Автоматика и телемеханика. — 2000. — № 9. — С. 15—43.

57. *Агаев Р. П.* Об исследовании и применении лапласовских спектров орграфов кольцевой структуры [Текст] // Автоматика и телемеханика. — 2008. — № 2. — С. 3—16.

58. *Мартынов А. В.*, *Курейчик В. М.* Гибридный алгоритм решения задачи коммивояжера [Текст] // Известия Южного федерального университета. Технические науки. — 2015. — 4 (165). — С. 36—44.

59. *Курейчик В. М.*, *Мартынов А. В.* Об алгоритмах решения задачи коммивояжера с временными ограничениями [Текст] // Информатика, вычислительная техника и инженерное образование. — 2014. — № 1. — С. 1—13.

60. *Ченцов А. Г.*, *Ченцов П. А.* Маршрутизация в условиях ограничений: задача о посещении мегаполисов [Текст] // Автоматика и телемеханика. — 2016. — № 11. — С. 96—117.

61. *Коротаева Л. Н.*, *Ченцов А. Г.* Об одном обобщении задачи коммивояжера “на узкие места” [Текст] // Журнал вычислительной математики и математической физики. — 1995. — Т. 35, № 7. — С. 1067—1076.

62. *Пузынина С. А.* Совершенные раскраски вершин графа G(Z^2) в три цвета [Текст] // Дискретный анализ и исследование операций. — 2005. — Т. 12, № 1. — С. 37—54.

63. *Райгородский А. М.*, *Шабанов Д. А.* Задача Эрдеша–Хайнала о раскрасках гиперграфов, ее обобщения и смежные проблемы [Текст] // Успехи математических наук. — 2011. — Т. 66, 5 (401. — С. 109—182.

64. *Шабанов Д. А.* О существовании полноцветных раскрасок для равномерных гиперграфов [Текст] // Математический сборник. — 2010. — Т. 201, № 4. — С. 137—160.

65. *Осипов Г. С.* Лекции по искусственному интеллекту [Текст]. — URSS, 2009.

66. *Ghallab M.*, *Nau D.*, *Traverso P.* Automated Planning: theory and practice [Текст]. — Elsevier, 2004.

67. *Fikes R. E.*, *Nilsson N. J.* STRIPS: A new approach to the application of theorem proving to problem solving [Текст] // Artificial intelligence. — 1971. — Т. 2, № 3/4. — С. 189—208.

68. *Sacerdoti E. D.* Planning in a hierarchy of abstraction spaces [Текст] // Artificial intelligence. — 1974. — Т. 5, № 2. — С. 115—135.

69. *Sacerdoti E. D.* The nonlinear nature of plans [Текст] // Proceedings of the 4th International Joint Conference on Artificial intelligence (IJCAI 1975). — 1975. — С. 206—214.

70. *Chapman D.* Planning for conjunctive goals [Текст] // Artificial intelligence. — 1987. — Т. 32, № 3. — С. 333—377.

71. *Bylander T.* The computational complexity of propositional STRIPS planning [Текст] // Artificial Intelligence. — 1994. — Т. 69, № 1/2. — С. 165—204.

72. *Weld D. S.* An introduction to least commitment planning [Текст] // AI magazine. — 1994. — Т. 15, № 4. — С. 27—27.

73. *Kambhampati S.*, *Knoblock C. A.*, *Yang Q.* Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning [Текст] // Artificial Intelligence. — 1995. — Т. 76, № 1/2. — С. 167—238.

74. *Thiébaux S.*, *Hoffmann J.*, *Nebel B.* In defense of PDDL axioms [Текст] // Artificial Intelligence. — 2005. — Т. 168, № 1/2. — С. 38—69.

75. *Blum A. L.*, *Furst M. L.* Fast planning through planning graph analysis [Текст] // Artificial intelligence. — 1997. — Т. 90, № 1/2. — С. 281—300.

76. *Bonet B.*, *Geffner H.* Planning as heuristic search [Текст] // Artificial Intelligence. — 2001. — Т. 129, № 1/2. — С. 5—33.

77. *Hoffmann J.*, *Nebel B.* The FF planning system: Fast plan generation through heuristic search [Текст] // Journal of Artificial Intelligence Research. — 2001. — Т. 14. — С. 253—302.

78. *Silver D.*, *Huang A.*, *Maddison C. J.*, *Guez A.*, *Sifre L.*, *Van Den Driessche G.*, *Schrittwieser J.*, *Antonoglou I.*, *Panneershelvam V.*, *Lanctot M.* [и др.]. Mastering the game of Go with deep neural networks and tree search [Текст] // Nature. — 2016. — Т. 529, № 7587. — С. 484—489.

79. *Yao S.*, *Yu D.*, *Zhao J.*, *Shafran I.*, *Griffiths T.*, *Cao Y.*, *Narasimhan K.* Tree of thoughts: Deliberate problem solving with large language models [Текст] // Advances in neural information processing systems. — 2023. — Т. 36. — С. 11809—11822.

80. *Hao S.*, *Gu Y.*, *Ma H.*, *Hong J.*, *Wang Z.*, *Wang D.*, *Hu Z.* Reasoning with Language Model is Planning with World Model [Текст] // Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023). — 2023. — С. 8154—8173.

81. *Besta M.*, *Blach N.*, *Kubicek A.*, *Gerstenberger R.*, *Podstawski M.*, *Gianinazzi L.*, *Gajda J.*, *Lehmann T.*, *Niewiadomski H.*, *Nyczyk P.* [и др.]. Graph of thoughts: Solving elaborate problems with large language models [Текст] // Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI 2024). — 2024. — С. 17682—17690.

82. *Hernández C.*, *Yeoh W.*, *Baier J. A.*, *Zhang H.*, *Suazo L.*, *Koenig S.*, *Salzman O.* Simple and efficient bi-objective search algorithms via fast dominance checks [Текст] // Artificial intelligence. — 2023. — Т. 314. — С. 103807.

83. *Heiden E.*, *Palmieri L.*, *Bruns L.*, *Arras K. O.*, *Sukhatme G. S.*, *Koenig S.* Bench-MR: A motion planning benchmark for wheeled mobile robots [Текст] // IEEE Robotics and Automation Letters. — 2021. — Т. 6, № 3. — С. 4536—4543.

84. *Андрейчук А.*, *Яковлев К.* Планирование траектории на плоскости с учетом размера агента (мобильного робота, беспилотного транспортного средства) [Текст] // Четвертый Всероссийский научно-практический семинар “Беспилотные транспортные средства с элементами искусственного интеллекта” (БТС-ИИ-2017). — 2017. — С. 107—117.

85. *Дергачев С.*, *Яковлев К.* Об одном вопросе реализации алгоритма планирования траектории A [Текст] // Труды Пятого Всероссийского научно-практического семинара "Беспилотные транспортные средства с элементами искусственного интеллекта"(БТС-ИИ-2019). — 2019. — С. 66—76.

86. *Asai M.*, *Fukunaga A.* Tie-breaking strategies for cost-optimal best first search [Текст] // Journal of Artificial Intelligence Research. — 2017. — Т. 58. — С. 67—121.

87. *Яковлев К. С.*, *Петров А. В.*, *Хитьков В. В.* Программный комплекс навигации и управления беспилотными транспортными средствами [Текст] // Информационные технологии и вычислительные системы. — 2013. — № 3. — С. 72—83.

88. *Яковлев К.*, *Хитьков В.*, *Логинов М.*, *Петров А.* Система навигации группы БЛА на основе маркеров [Текст] // Робототехника и техническая кибернетика. — 2014. — № 4. — С. 44—48.

89. *Zong W.*, *Zhang C.*, *Wang Z.*, *Zhu J.*, *Chen Q.* Architecture design and implementation of an autonomous vehicle [Текст] // IEEE access. — 2018. — Т. 6. — С. 21956—21970.

90. *Осипов Г.*, *Тихомиров И.*, *Хачумов В.*, *Яковлев К.* Интеллектуальные системы управления автономными транспортными средствами: стандарты, проекты, реализации [Текст] // Авиакосмическое приборостроение. — 2009. — № 6. — С. 34—43.

91. *Макаров Д. А.*, *Панов А. И.*, *Яковлев К. С.* Архитектура многоуровневой интеллектуальной системы управления беспилотными летательными аппаратами [Текст] // Искусственный интеллект и принятие решений. — 2015. — № 3. — С. 18—33.

92. *Emel'yanov S.*, *Makarov D.*, *Panov A. I.*, *Yakovlev K.* Multilayer cognitive architecture for UAV control [Текст] // Cognitive Systems Research. — 2016. — Т. 39. — С. 58—72.

93. *Bojarski M.*, *Del Testa D.*, *Dworakowski D.*, *Firner B.*, *Flepp B.*, *Goyal P.*, *Jackel L. D.*, *Monfort M.*, *Muller U.*, *Zhang J.* [и др.]. End to end learning for self-driving cars [Текст]. — 2016.

94. *Zhu Y.*, *Mottaghi R.*, *Kolve E.*, *Lim J. J.*, *Gupta A.*, *Fei-Fei L.*, *Farhadi A.* Target-driven visual navigation in indoor scenes using deep reinforcement learning [Текст] // Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA 2017). — 2017. — С. 3357—3364.

95. *Gupta S.*, *Davidson J.*, *Levine S.*, *Sukthankar R.*, *Malik J.* Cognitive mapping and planning for visual navigation [Текст] // Proceedings of the 2017 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2017). — 2017. — С. 2616—2625.

96. *Radford A.*, *Kim J. W.*, *Hallacy C.*, *Ramesh A.*, *Goh G.*, *Agarwal S.*, *Sastry G.*, *Askell A.*, *Mishkin P.*, *Clark J.*, *Krueger G.*, *Sutskever I.* Learning transferable visual models from natural language supervision [Текст] // Proceedings of the 38th International Conference on Machine Learning (ICML 2021). — 2021. — С. 8748—8763.

97. *Ichter B.*, *Brohan A.*, *Chebotar Y.*, *Finn C.*, *Hausman K.*, *Herzog A.*, *Ho D.*, *Ibarz J.*, *Irpan A.*, *Jang E.*, *Julian R.*, *Kalashnikov D.*, *Levine S.*, *Lu Y.*, *Parada C.*, *Rao K.*, *Sermanet P.*, *Toshev A. T.*, *Vanhoucke V.*, *Xia F.*, *Xiao T.*, *Xu P.*, *Yan M.*, *Brown N.*, *Ahn M.*, *Cortes O.*, *Sievers N.*, *Tan C.*, *Xu S.*, *Reyes D.*, *Rettinghouse J.*, *Quiambao J.*, *Pastor P.*, *Luu L.*, *Lee K.-H.*, *Kuang Y.*, *Jesmonth S.*, *Joshi N. J.*, *Jeffrey K.*, *Ruano R. J.*, *Hsu J.*, *Gopalakrishnan K.*, *David B.*, *Zeng A.*, *Fu C. K.* Do As I Can, Not As I Say: Grounding Language in Robotic Affordances [Текст] // Proceedings of the 6th Conference on Robot Learning (CORL 2023). — 2023. — С. 287—318.

98. *Zitkovich B.*, *Yu T.*, *Xu S.*, *Xu P.*, *Xiao T.*, *Xia F.*, *Wu J.*, *Wohlhart P.*, *Welker S.*, *Wahid A.* [и др.]. Rt-2: Vision-language-action models transfer web knowledge to robotic control [Текст] // Proceedings of the 6th Conference on Robot Learning (CORL 2023). — 2023. — С. 2165—2183.

99. *Karkus P.*, *Ivanovic B.*, *Mannor S.*, *Pavone M.* Diffstack: A differentiable and modular control stack for autonomous vehicles [Текст] // Proceedings of the 6th Conference on Robot Learning (CORL 2023). — 2023. — С. 2170—2180.

100. *LaValle S. M.*, *Kuffner Jr J. J.* Randomized kinodynamic planning [Текст] // The international journal of robotics research. — 2001. — Т. 20, № 5. — С. 378—400.

101. *Kleinbort M.*, *Solovey K.*, *Littlefield Z.*, *Bekris K. E.*, *Halperin D.* Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation [Текст] // IEEE Robotics and Automation Letters. — 2019. — Т. 4, № 2. — С. i—vii.

102. *Kuffner J. J.*, *LaValle S. M.* RRT-connect: An efficient approach to single-query path planning [Текст] // Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA 2020). Т. 2. — IEEE. 2000. — С. 995—1001.

103. *Sánchez G.*, *Latombe J.-C.* A single-query bi-directional probabilistic roadmap planner with lazy collision checking [Текст] // Robotics Research: The Tenth International Symposium. — Springer, 2003. — С. 403—417.

104. *Yershova A.*, *Jaillet L.*, *Siméon T.*, *LaValle S. M.* Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain [Текст] // Proceedings of the 2005 IEEE international conference on robotics and automation (ICRA 2025). — IEEE, 2005. — С. 3856—3861.

105. *Solovey K.*, *Janson L.*, *Schmerling E.*, *Frazzoli E.*, *Pavone M.* Revisiting the asymptotic optimality of RRT* [Текст] // Proceedings of the 2020 IEEE international conference on robotics and automation (ICRA 2020). — IEEE, 2020. — С. 2189—2195.

106. *Gammell J. D.*, *Srinivasa S. S.*, *Barfoot T. D.* Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic [Текст] // Proceedings of the 2014 IEEE/RSJ

international conference on intelligent robots and systems (IROS 2014). — IEEE, 2014. — С. 2997—3004.

107. *Arslan O.*, *Tsiotras P.* Use of relaxation methods in sampling-based algorithms for optimal motion planning [Текст] // Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA 2013). — IEEE, 2013. — С. 2421—2428.

108. *Gammell J. D.*, *Barfoot T. D.*, *Srinivasa S. S.* Batch informed trees (BIT*): Informed asymptotically optimal anytime search [Текст] // The International Journal of Robotics Research. — 2020. — Т. 39, № 5. — С. 543—567.

109. *A. I.* Nonlinear Control Systems [Текст]. — Berlin: Springer, 1995. — 549 p.

110. *Khalil H.* Nonlinear Systems (3rd ed.). [Текст]. — Upper Saddle River, 2002.

111. *Utkin V. I.* Sliding mode control design principles and applications to electric drives [Текст] // IEEE transactions on industrial electronics. — 2002. — Т. 40, № 1. — С. 23—36.

112. *Пропой А.* Применение методов линейного программирования для синтеза импульсных автоматических систем [Текст] // Автоматика и телемеханика. — 1963. — Т. 24, № 7. — С. 912—920.

113. *Garcia C. E.*, *Prett D. M.*, *Morari M.* Model predictive control: Theory and practice – A survey [Текст] // Automatica. — 1989. — Т. 25, № 3. — С. 335—348.

114. *Fliess M.*, *Lévine J.*, *Martin P.*, *Rouchon P.* Flatness and defect of non-linear systems: introductory theory and examples [Текст] // International journal of control. — 1995. — Т. 61, № 6. — С. 1327—1361.

115. *Bascetta L.*, *Arrieta I. M.*, *Prandini M.* Flat-RRT*: A sampling-based optimal trajectory planner for differentially flat vehicles with constrained dynamics [Текст] // IFAC-PapersOnLine. — 2017. — Т. 50, № 1. — С. 6965—6970.

116. *Sahoo S. R.*, *Chiddarwar S. S.* Mobile robot control using bond graph and flatness based approach [Текст] // Procedia computer science. — 2018. — Т. 133. — С. 213—221.

117. *Четвериков В. Н.* Плоскостность динамически линеаризуемых систем [Текст] // Дифференциальные уравнения. — 2004. — Т. 40, № 12. — С. 1665—1674.

118. *Белинская Ю.*, *Четвериков В.* Метод накрытий для терминального управления с учетом ограничений [Текст] // Дифференциальные уравнения. — 2014. — Т. 50, № 12. — С. 1629—1629.

119. *Belinskaya Y. S.*, *Chetverikov V.* Covering method for point-to-point control of constrained flat systems [Текст] // IFAC-PapersOnLine. — 2015. — Т. 48, № 11. — С. 924—929.

120. *Pivtoraiko M.*, *Kelly A.* Generating near minimal spanning control sets for constrained motion planning in discrete state spaces [Текст] // Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005). — IEEE. 2005. — С. 3231—3237.

121. *Wang B.*, *Gong J.*, *Chen H.* Motion primitives representation, extraction and connection for automated vehicle motion planning applications [Текст] // IEEE Transactions on Intelligent Transportation Systems. — 2019. — Т. 21, № 9. — С. 3931—3945.

122. *Головин В. А.*, *Яковлев К. С.* Примитивы движения робота в задаче планирования траектории с кинематическими ограничениями [Текст] // Информатика и автоматизация. — 2023. — Т. 22, № 6. — С. 1354—1386.

123. *Dolgov D.*, *Thrun S.*, *Montemerlo M.*, *Diebel J.* Path planning for autonomous vehicles in unknown semi-structured environments [Текст] // The international journal of robotics research. — 2010. — Т. 29, № 5. — С. 485—501.

124. *Likhachev M.*, *Ferguson D.* Planning long dynamically feasible maneuvers for autonomous vehicles [Текст] // The International Journal of Robotics Research. — 2009. — Т. 28, № 8. — С. 933—945.

125. *Webb D. J.*, *Van Den Berg J.* Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics [Текст] // Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA 2013). — IEEE. 2013. — С. 5054—5061.

126. *Bianco C. G. L.*, *Piazzi A.*, *Romano M.* Smooth motion generation for unicycle mobile robots via dynamic path inversion [Текст] // IEEE Transactions on Robotics. — 2004. — Т. 20, № 5. — С. 884—891.

127. *Palmieri L.*, *Arras K. O.* A novel RRT extend function for efficient and smooth mobile robot motion planning [Текст] // Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014). — IEEE. 2014. — С. 205—211.

128. *Dubins L. E.* On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents [Текст] // American Journal of mathematics. — 1957. — Т. 79, № 3. — С. 497—516.

129. *Bui X.-N.*, *Boissonnat J.-D.*, *Soueres P.*, *Laumond J.-P.* Shortest path synthesis for Dubins non-holonomic robot [Текст] // Proceedings of the 1994 IEEE International Conference on Robotics and Automation (ICRA 1994). — 1994. — С. 2—7.

130. *Reeds J.*, *Shepp L.* Optimal paths for a car that goes both forwards and backwards [Текст] // Pacific journal of mathematics. — 1990. — Т. 145, № 2. — С. 367—393.

131. *Chiang H.-T. L.*, *Hsu J.*, *Fiser M.*, *Tapia L.*, *Faust A.* RL-RRT: Kinodynamic motion planning via learning reachability estimators from RL policies [Текст] // IEEE Robotics and Automation Letters. — 2019. — Т. 4, № 4. — С. 4298—4305.

132. *Geraerts R.*, *Overmars M. H.* Creating high-quality paths for motion planning [Текст] // The international journal of robotics research. — 2007. — Т. 26, № 8. — С. 845—863.

133. *Luna R.*, *Şucan I. A.*, *Moll M.*, *Kavraki L. E.* Anytime solution optimization for sampling-based motion planning [Текст] // Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA 2013). — 2013. — С. 5068—5074.

134. *Heiden E.*, *Palmieri L.*, *Koenig S.*, *Arras K. O.*, *Sukhatme G. S.* Gradient-informed path smoothing for wheeled mobile robots [Текст] // Proceedings of the 2018 IEEE international conference on robotics and automation (ICRA 2018). — 2018. — С. 1710—1717.

135. *Ali Z. A.*, *Angulo B.*, *Golovin V.*, *Yakovlev K.* Empirical Evaluation of Theta*-RRT and GRIPS Algorithms [Текст] // Proceedings of the 2021 International Siberian Conference on Control and Communications (SIBCON 2021). — 2021. — С. 1—6.

136. *Fiorini P.*, *Shiller Z.* Motion planning in dynamic environments using velocity obstacles [Текст] // The International Journal of Robotics Research. — 1998. — Т. 17, № 7. — С. 760—772.

137. *Ziegler J.*, *Bender P.*, *Dang T.*, *Stiller C.* Trajectory planning for Bertha – A local, continuous method [Текст] // Proceedings of the 2014 IEEE intelligent vehicles symposium (IV 2014). — 2014. — С. 450—457.

138. *Liu J.*, *Jayakumar P.*, *Stein J. L.*, *Ersal T.* Combined speed and steering control in high-speed autonomous ground vehicles for obstacle avoidance using model predictive control [Текст] // IEEE Transactions on Vehicular Technology. — 2017. — Т. 66, № 10. — С. 8746—8763.

139. *Corno M.*, *Gimondi A.*, *Panzani G.*, *Roselli F.*, *Alessandretti A.*, *Savaresi S. M.* A non-optimization-based dynamic path planning for autonomous obstacle avoidance [Текст] // IEEE Transactions on Control Systems Technology. — 2022. — Т. 31, № 2. — С. 722—734.

140. *Liu S. K. M. L. Y.*, *Furcy D.* Incremental Heuristic Search in Artificial Intelligence [Текст] // AI Magazine. — 2004. — Т. 25, № 2. — С. 99—112.

141. *Sun X.*, *Koenig S.* The Fringe-Saving A* Search Algorithm – A Feasibility Study [Текст] // Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007). — 2007. — С. 2391—2397.

142. *Trovato K. I.*, *Dorst L.* Differential a [Текст] // IEEE Transactions on Knowledge and Data Engineering. — 2002. — Т. 14, № 6. — С. 1218—1229.

143. *Sun X.*, *Koenig S.*, *Yeoh W.* Generalized adaptive A [Текст] // Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008). — 2008. — С. 469—476.

144. *Hernandez C.*, *Asin R.*, *Baier J.* Reusing previously found A* paths for fast goal-directed navigation in dynamic terrain [Текст] // Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 2015). — 2015. — С. 1158—1164.

145. *Ferguson D.*, *Kalra N.*, *Stentz A.* Replanning with RRTs [Текст] // Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA 2006). — IEEE. 2006. — С. 1243—1248.

146. *Zucker M.*, *Kuffner J.*, *Branicky M.* Multipartite RRTs for rapid replanning in dynamic environments [Текст] // Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA 2007). — IEEE. 2007. — С. 1603—1609.

147. *Cui B.*, *Cui R.*, *Yan W.*, *Wang Y.*, *Zhang S.* RT-RRT: Reverse tree guided real-time path planning/replanning in unpredictable dynamic environments [Текст] // Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2024). — 2024. — С. 5380—5387.

148. *Silver D.* Cooperative pathfinding [Текст] // Proceedings of The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2005). — 2005. — С. 117—122.

149. *Yoshizumi T.*, *Miura T.*, *Ishida T.* A* with Partial Expansion for Large Branching Factor Problems [Текст] // Proceedings of The 14th AAAI Conference on Artificial Intelligence (AAAI 2000). — 2000. — С. 923—929.

150. *Goldenberg M.*, *Felner A.*, *Stern R.*, *Sharon G.*, *Sturtevant N.*, *Holte R. C.*, *Schaeffer J.* Enhanced partial expansion A* [Текст] // Journal of Artificial Intelligence Research. — 2014. — Т. 50. — С. 141—187.

151. *Phillips M.*, *Likhachev M.* SIPP: Safe interval path planning for dynamic environments [Текст] // Proceedings of The 2011 IEEE International Conference on Robotics and Automation (ICRA 2011). — 2011. — С. 5628—5635.

152. *Narayanan V.*, *Phillips M.*, *Likhachev M.* Anytime safe interval path planning for dynamic environments [Текст] // Proceedings of The 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2012). — 2012. — С. 4708—4715.

153. *Ren Z.*, *Rathinam S.*, *Likhachev M.*, *Choset H.* Multi-objective safe-interval path planning with dynamic obstacles [Текст] // IEEE Robotics and Automation Letters. — 2022. — Т. 7, № 3. — С. 8154—8161.

154. *Ali Z. A.*, *Yakovlev K.* Safe interval path planning with kinodynamic constraints [Текст] // Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI 2023). — 2023. — С. 12330—12337.

155. *Thomas D. W.*, *Ruml W.*, *Shimony S. E.* Real-time Safe Interval Path Planning [Текст] // Proceedings of the 17th International Symposium Combinatorial Search (SoCS 2024). Т. 17. — 2024. — С. 161—169.

156. *Sintov A.*, *Shapiro A.* Time-based RRT algorithm for rendezvous planning of two dynamic systems [Текст] // Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA) 2014. — IEEE. 2014. — С. 6745—6750.

157. *Grothe F.*, *Hartmann V. N.*, *Orthey A.*, *Toussaint M.* ST-RRT*: Asymptotically-optimal bidirectional motion planning through space-time [Текст] // Proceedings of the 2022 International Conference on Robotics and Automation (ICRA 2022). — IEEE. 2022. — С. 3314—3320.

158. *Wilson T. S.*, *Thomason W.*, *Kingston Z.*, *Gammell J. D.* AORRTC: Almost-Surely Asymptotically Optimal Planning with RRT-Connect [Текст] // IEEE Robotics and Automation Letters. — 2025.

159. *Sim J.*, *Kim J.*, *Nam C.* Safe interval RRT* for scalable multi-robot path planning in continuous space [Текст] // arXiv preprint arXiv:2404.01752. — 2024.

160. *Kerimov N.*, *Onegin A.*, *Yakovlev K.* Safe Interval Randomized Path Planning For Manipulators [Текст] // Proceedings of the 35th International Conference on Automated Planning and Scheduling (ICAPS 2025). — 2025. — С. 213—217.

161. *Aine S.*, *Swaminathan S.*, *Narayanan V.*, *Hwang V.*, *Likhachev M.* Multi-heuristic A* [Текст] // The International Journal of Robotics Research. — 2016. — Т. 35, № 1—3. — С. 224—243.

162. *Li J.*, *Tinka A.*, *Kiesel S.*, *Durham J. W.*, *Kumar T. S.*, *Koenig S.* Lifelong multi-agent path finding in large-scale warehouses [Текст] // Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021). — 2021. — С. 11272—11281.

163. *Azadeh K.*, *De Koster R.*, *Roy D.* Robotized and automated warehouse systems: Review and recent developments [Текст] // Transportation Science. — 2019. — Т. 53, № 4. — С. 917—945.

164. *Ma H.*, *Li J.*, *Kumar T.*, *Koenig S.* Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks [Текст] // Proceedings of The 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2017). — International Foundation for Autonomous Agents, Multiagent Systems. 2017. — С. 837—845.

165. *Morris R.*, *Pasareanu C. S.*, *Luckow K.*, *Malik W.*, *Ma H.*, *Kumar T. S.*, *Koenig S.* Planning, scheduling and monitoring for airport surface operations [Текст] // Proceedings of the Planning for Hybrid Systems Workshop at the 30th AAAI Conference on Artificial Intelligence (AAAI 2016). — 2016. — С. 608—614.

166. *Schwarting W.*, *Alonso-Mora J.*, *Rus D.* Planning and decision-making for autonomous vehicles [Текст] // Annual Review of Control, Robotics, and Autonomous Systems. — 2018. — Т. 1, № 1. — С. 187—210.

167. *Yu J.*, *LaValle S. M.* Optimal multi-robot path planning on graphs: Structure and computational complexity [Текст] // arXiv preprint arXiv:1507.03289. — 2015.

168. *Kornhauser D.*, *Miller G.*, *Spirakis P.* Coordinating Pebble Motion On Graphs, The Diameter Of Permutation Groups, And Applications [Текст] // Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS) 1984. — 1984. — С. 241—250.

169. *Nebel B.* On the computational complexity of multi-agent pathfinding on directed graphs [Текст] // Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS 2020). — 2020. — С. 212—216.

170. *Felner A.*, *Stern R.*, *Shimony S.*, *Boyarski E.*, *Goldenberg M.*, *Sharon G.*, *Sturtevant N.*, *Wagner G.*, *Surynek P.* Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges [Текст] // Proceedings of the 10th International Symposium on Combinatorial Search (SoCS 2017). — 2017. — С. 29—37.

171. *Yu J.*, *LaValle S. M.* Multi-agent path planning and network flow [Текст] // Algorithmic Foundations of Robotics X: Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics. — Springer. 2013. — С. 157—173.

172. *Ali Z. A.*, *Yakovlev K.* Improved Anonymous Multi-Agent Path Finding Algorithm [Текст] // Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI 2024). — 2024. — С. 17291—17298.

173. *Surynek P.*, *Felner A.*, *Stern R.*, *Boyarski E.* Efficient SAT approach to multi-agent path finding under the sum of costs objective [Текст] // Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016). — IOS Press. 2016. — С. 810—818.

174. *Barták R.*, *Švancara J.* On SAT-Based Approaches for Multi-Agent Path Finding with the Sum-of-Costs Objective [Текст] // Proceedings of the 12th Annual Symposium on Combinatorial Search (SOCS 2019). — 2019. — С. 10—17.

175. *Yu J.*, *LaValle S. M.* Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics [Текст] // IEEE Transactions on Robotics. — 2016. — Т. 32, № 5. — С. 1163—1177.

176. *Lam E.*, *Le Bodic P.*, *Harabor D.*, *Stuckey P. J.* Branch-and-cut-and-price for multi-agent path finding [Текст] // Computers & Operations Research. — 2022. — Т. 144. — С. 105809.

177. *Walker T.*, *Sturtevant N. R.*, *Felner A.* Extended Increasing Cost Tree Search for Non-Unit Cost Domains [Текст] // Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018). — 2018. — С. 534—540. — URL: https://www.ijcai.org/proceedings/2018/0074.pdf.

178. *Wagner G.*, *Choset H.* M*: A complete multirobot path planning algorithm with performance bounds [Текст] // Proceedings of The 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011). — 2011. — С. 3260—3267.

179. *Sharon G.*, *Stern R.*, *Felner A.*, *Sturtevant N.* Conflict-Based Search For Optimal Multi-Agent Path Finding [Текст] // Proceedings of the AAAI Conference on Artificial Intelligence. Т. 26. — 2012. — С. 563—569.

180. *Boyarski E.*, *Felner A.*, *Stern R.*, *Sharon G.*, *Betzalel O.*, *Tolpin D.*, *Shimony E.* ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding [Текст] // Proceedings of The 24th International Joint Conference on Artificial Intelligence (IJCAI 2015). — 2015. — С. 740—746.

181. *Li J.*, *Harabor D.*, *Stuckey P. J.*, *Felner A.*, *Ma H.*, *Koenig S.* Disjoint splitting for multi-agent path finding with conflict-based search [Текст] // Proceedings of The 29th International Conference on Automated Planning and Scheduling (ICAPS 2019). — 2019. — С. 279—283.

182. *Felner A.*, *Li J.*, *Boyarski E.*, *Ma H.*, *Cohen L.*, *Kumar T. S.*, *Koenig S.* Adding heuristics to conflict-based search for multi-agent path finding [Текст] // Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018). — 2018. — С. 83—87.

183. *Li J.*, *Harabor D.*, *Stuckey P. J.*, *Ma H.*, *Koenig S.* Symmetry-breaking constraints for grid-based multi-agent path finding [Текст] // Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019). Т. 33. — 2019. — С. 6087—6095.

184. *Li J.*, *Gange G.*, *Harabor D.*, *Stuckey P. J.*, *Ma H.*, *Koenig S.* New techniques for pairwise symmetry breaking in multi-agent path finding [Текст] // Proceedings of the International Conference on Automated Planning and Scheduling. Т. 30. — 2020. — С. 193—201.

185. *Barer M.*, *Sharon G.*, *Stern R.*, *Felner A.* Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem [Текст] // Proceedings of The 7th Annual Symposium on Combinatorial Search (SoCS 2014). — 2014. — С. 19—27.

186. *Li J.*, *Ruml W.*, *Koenig S.* EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding [Текст] // Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021). — 2021. — С. 12353—12362.

187. *Surynek P.* A novel approach to path planning for multiple robots in bi-connected graphs [Текст] // Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009). — IEEE. 2009. — С. 3613—3619.

188. *Luna R. J.*, *Bekris K. E.* Push and swap: Fast cooperative path-finding with completeness guarantees [Текст] // Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011). — 2011. — С. 294—300.

189. *De Wilde B.*, *Ter Mors A. W.*, *Witteveen C.* Push and rotate: a complete multi-agent pathfinding algorithm [Текст] // Journal of Artificial Intelligence Research. — 2014. — Т. 51. — С. 443—492.

190. *Okumura K.*, *Machida M.*, *Défago X.*, *Tamura Y.* Priority inheritance with backtracking for iterative multi-agent path finding [Текст] // Artificial Intelligence. — 2022. — Т. 310. — С. 103752.

191. *Erdmann M.*, *Lozano-Perez T.* On multiple moving objects [Текст] // Algorithmica. — 1987. — Т. 2, № 1. — С. 477—521.

192. *Cap M.*, *Vokrinek J.*, *Kleiner A.* Complete Decentralized Method for On-Line Multi-Robot Trajectory Planning in Well-formed Infrastructures [Текст] // Proceedings of The 25th International Conference on Automated Planning and Scheduling (ICAPS 2015). — 2015. — С. 324—332.

193. *Cap M.*, *Novak P.*, *Kleiner A.*, *Selecky M.* Prioritized planning algorithms for trajectory coordination of multiple mobile robots [Текст] // IEEE Transactions on Automation Science and Engineering. — 2015. — Т. 12, № 3. — С. 835—849.

194. *Bennewitz M.*, *Burgard W.*, *Thrun S.* Optimizing schedules for prioritized path planning of multi-robot systems [Текст] // Proceedings of The 2001 IEEE International Conference on Robotics and Automation (ICRA 2001). Т. 1. — 2001. — С. 271—276.

195. *Bennewitz M.*, *Burgard W.*, *Thrun S.* Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots [Текст] // Robotics and autonomous systems. — 2002. — Т. 41, № 2. — С. 89—99.

196. *Ma H.*, *Harabor D.*, *Stuckey P. J.*, *Li J.*, *Koenig S.* Searching with Consistent Prioritization for Multi-Agent Path Finding [Текст] // Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019). — 2019. — С. 7643—7650.

197. *Okumura K.* LaCAM: Search-based algorithm for quick multi-agent pathfinding [Текст] // Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI 2023). — 2023. — С. 11655—11662.

198. *Okumura K.* Improving LaCAM for scalable eventually optimal multi-agent pathfinding [Текст] // Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI 2023). — 2023. — С. 243—251.

199. *Okumura K.* Engineering LaCAM*: Towards Real-time, Large-scale, and Near-optimal Multi-agent Pathfinding [Текст] // Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024). — 2024. — С. 1501—1509.

200. *Li J.*, *Chen Z.*, *Harabor D.*, *Stuckey P. J.*, *Koenig S.* Anytime Multi-Agent Path Finding via Large Neighborhood Search [Текст] // Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI) 2021). — 2021. — С. 4127—4135.

201. *Li J.*, *Chen Z.*, *Zheng Y.*, *Chan S.-H.*, *Harabor D.*, *Stuckey P. J.*, *Ma H.*, *Koenig S.* Scalable Rail Planning and Replanning: Winning the 2020 Flatland Challenge [Текст] // Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS 2021). — 2021. — С. 477—485.

202. *Li J.*, *Chen Z.*, *Harabor D.*, *Stuckey P. J.*, *Koenig S.* MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search [Текст] // Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022). — 2022. — С. 10256—10265.

203. *Phan T.*, *Huang T.*, *Dilkina B.*, *Koenig S.* Adaptive anytime multi-agent path finding using bandit-based large neighborhood search [Текст] // Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI 2024). — 2024. — С. 17514—17522.

204. *Kaduri O.*, *Boyarski E.*, *Stern R.* Algorithm selection for optimal multi-agent pathfinding [Текст] // Proceedings of the 30th International conference on automated planning and scheduling (ICAPS 2020). — 2020. — С. 161—165.

205. *Ren J.*, *Sathiyanarayanan V.*, *Ewing E.*, *Senbaslar B.*, *Ayanian N.* MAPFAST: A Deep Algorithm Selector for Multi Agent Path Finding using Shortest Path Embeddings [Текст] // Proceedings of the 20th International

Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2021). — 2021. — С. 1055—1063.

206. *Huang T.*, *Dilkina B.*, *Koenig S.* Learning Node-Selection Strategies in Bounded Suboptimal Conflict-Based Search for Multi-Agent Path Finding [Текст] // International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS). — 2021.

207. *Huang T.*, *Li J.*, *Koenig S.*, *Dilkina B.* Anytime multi-agent path finding via machine learning-guided large neighborhood search [Текст] // Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022). — 2022. — С. 9368—9376.

208. *Sartoretti G.*, *Kerr J.*, *Shi Y.*, *Wagner G.*, *Kumar T. S.*, *Koenig S.*, *Choset H.* Primal: Pathfinding via reinforcement and imitation multi-agent learning [Текст] // IEEE Robotics and Automation Letters. — 2019. — Т. 4, № 3. — С. 2378—2385.

209. *Ma Z.*, *Luo Y.*, *Pan J.* Learning selective communication for multi-agent path finding [Текст] // IEEE Robotics and Automation Letters. — 2021. — Т. 7, № 2. — С. 1455—1462.

210. *Li W.*, *Chen H.*, *Jin B.*, *Tan W.*, *Zha H.*, *Wang X.* Multi-agent path finding with prioritized communication learning [Текст] // 2022 International Conference on Robotics and Automation (ICRA 2022). — IEEE. 2022. — С. 10695—10701.

211. *Wang Y.*, *Xiang B.*, *Huang S.*, *Sartoretti G.* SCRIMP: Scalable Communication for Reinforcement-and Imitation-Learning-Based Multi-Agent Pathfinding [Текст] // Proceedings of The 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2023). — 2023. — С. 9301—9308.

212. *Skrynnik A.*, *Andreychuk A.*, *Nesterova M.*, *Yakovlev K.*, *Panov A.* Learn to Follow: Decentralized Lifelong Multi-agent Pathfinding via Planning and Learning [Текст] // Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI 2024). — 2024. — in Press.

213. *Skrynnik A.*, *Andreychuk A.*, *Yakovlev K.*, *Panov A.* Decentralized Monte Carlo Tree Search for Partially Observable Multi-agent Pathfinding [Текст] // Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI 2024). — 2024. — in Press.

214. *Skrynnik A.*, *Andreychuk A.*, *Yakovlev K.*, *Panov A. I.* When to Switch: Planning and Learning for Partially Observable Multi-Agent Pathfinding [Текст] // IEEE Transactions on Neural Networks and Learning Systems. — 2023.

215. *Боковой А. В.*, *Муравьев К. Ф.*, *Яковлев К. С.* Система одновременного картирования, локализации и исследования неизвестной местности по видеопотоку [Текст] // Информационные технологии и вычислительные системы. — 2020. — № 2. — С. 51—61.

216. *Миронов К.*, *Юдин Д.*, *Алхаддад М.*, *Макаров Д.*, *Пушкарев Д.*, *Линок С.*, *Белкин И.*, *Криштопик А.*, *Головин В.*, *Яковлев К.*, *Панов А.* STRL-ROBOTICS: интеллектуальное управление поведением робототехнической платформы в человеко-ориентированной среде [Текст] // Искусственный интелект и принятие решений. — 2023. — № 2. — С. 45—63.

217. *Bresenham J. E.* Algorithm for computer control of a digital plotter [Текст] // IBM Systems journal. — 1965. — Т. 4, № 1. — С. 25—30.

218. *Rivera N.*, *Hernández C.*, *Baier J.* Grid Pathfinding On the 2k Neighborhoods [Текст] // Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017). — 2017. — С. 891—897.

219. *Sturtevant N. R.* Benchmarks for Grid-Based Pathfinding [Текст] // IEEE Transactions on Computational Intelligence and AI in Games. — 2012. — Т. 4, № 2. — С. 144—148.

220. *Wu X.* An efficient antialiasing technique [Текст] // Proceedings of The 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1991). — 1991. — С. 143—152.

221. *Walker T. T.*, *Sturtevant N. R.*, *Felner A.* Generalized and sub-optimal bipartite constraints for conflict-based search [Текст] // Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020). — 2020. — С. 7277—7284.

222. *Andreychuk A.* Multi-agent path finding with kinematic constraints via conflict based search [Текст] // Lecture Notes in Artificial Intelligence. Т. 12412. — Springer, Cham, 2020. — С. 29—45.

223. *Walker T. T.*, *Sturtevant N. R.* Collision detection for agents in multi-agent pathfinding [Текст]. — 2019.

224. *Andreychuk A.*, *Yakovlev K.*, *Surynek P.*, *Atzmon D.*, *Stern R.* Multi-agent pathfinding with continuous time [Текст] // Artificial Intelligence. — 2022. — С. 103662.

225. *Li A.*, *Chen Z.*, *Harabor D.*, *Vered M.* Revisiting Conflict Based Search with Continuous-Time [Текст] // arXiv preprint arXiv:2501.07744. — 2025.

226. *Combrink A.*, *Roselli S. F.*, *Fabian M.* Optimal Multi-agent Path Finding in Continuous Time [Текст] // arXiv preprint arXiv:2508.16410. — 2025.

227. *Andreychuk A.*, *Yakovlev K.*, *Boyarski E.*, *Stern R.* Improving continuous-time conflict based search [Текст] // Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021). — 2021. — С. 11220—11227.

228. *Cohen L.*, *Greco M.*, *Ma H.*, *Hernández C.*, *Felner A.*, *Kumar T. S.*, *Koenig S.* Anytime Focal Search with Applications. [Текст] // Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018). — 2018. — С. 1434—1441.

229. *Андрейчук А. А.* Эффективный поиск ограниченно-субоптимальных решений задачи многоагентного планирования [Текст] // Искусственный интеллект и принятие решений. — 2022. — № 1. — С. 57—70.

230. *Van Den Berg J. P.*, *Overmars M. H.* Prioritized motion planning for multiple robots [Текст] // Proceedings of The 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005). — IEEE. 2005. — С. 430—435.

231. *Ali Z. A.*, *Yakovlev K.* Prioritized SIPP for multi-agent path finding with kinematic constraints [Текст] // Procedding of the 6th International Conference on Interactive Collaborative Robotics (ICR 2021). — Springer. 2021. — С. 1—13.

232. *Yakovlev K.*, *Andreychuk A.*, *Vorobyev V.* Prioritized multi-agent path finding for differential drive robots [Текст] // Proceedings of the 2019 European Conference on Mobile Robots (ECMR 2019). — IEEE. 2019. — С. 1—6.

233. *Яковлев К. С.*, *Макаров Д. А.*, *Баскин Е. С.* Метод автоматического планирования траектории беспилотного летательного аппарата в условиях ограничений на динамику полета [Текст] // Искусственный интеллект и принятие решений. — 2014. — № 4. — С. 3—17.

234. *Munoz P.*, *Rodriguez-Moreno M.* Improving efficiency in any-angle path-planning algorithms [Текст] // Proceedings of the 6th IEEE International Conference Intelligent Systems (IS 2012). — 2012. — С. 213—218.

235. *Kim H.*, *Kim D.*, *Shin J.-U.*, *Kim H.*, *Myung H.* Angular rate-constrained path planning algorithm for unmanned surface vehicles [Текст] // Ocean Engineering. — 2014. — Т. 84. — С. 37—44.

236. *Yakovlev K.*, *Baskin E.*, *Hramoin I.* Grid-based angle-constrained path planning [Текст] // Proceedings of the 38th German Conference on Artificial Intelligence (Künstliche Intelligenz, KI 2015). — 2015. — С. 208—221.

237. *Pitteway M. L.* Algorithms of conic generation [Текст] // Fundamental Algorithms for Computer Graphics. — Springer, 1985. — С. 219—237.

238. *Bennett J.* OpenStreetMap [Текст]. — Packt Publishing Ltd, 2010.

239. *He K.*, *Zhang X.*, *Ren S.*, *Sun J.* Deep residual learning for image recognition [Текст] // Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016). — 2016. — С. 770—778.

240. *Vaswani A.*, *Shazeer N.*, *Parmar N.*, *Uszkoreit J.*, *Jones L.*, *Gomez A. N.*, *Kaiser Ł.*, *Polosukhin I.* Attention is all you need [Текст] // Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017). Т. 30. — 2017.

241. *Dosovitskiy A.*, *Beyer L.*, *Kolesnikov A.*, *Weissenborn D.*, *Zhai X.*, *Unterthiner T.*, *Dehghani M.*, *Minderer M.*, *Heigold G.*, *Gelly S.*, *Uszkoreit J.*, *Houlsby N.* An image is worth 16x16 words: Transformers for image recognition at scale [Текст] // Proceedings of the 9th International Conference on Learning Representations (ICLR 2021). — 2021.

242. *Han K.*, *Wang Y.*, *Chen H.*, *Chen X.*, *Guo J.*, *Liu Z.*, *Tang Y.*, *Xiao A.*, *Xu C.*, *Xu Y.* [и др.]. A survey on vision transformer [Текст] // IEEE transactions on pattern analysis and machine intelligence. — 2022. — Т. 45, № 1. — С. 87—110.

243. *Pang Y.*, *Lin J.*, *Qin T.*, *Chen Z.* Image-to-image translation: Methods and applications [Текст] // IEEE Transactions on Multimedia. — 2021. — Т. 24. — С. 3859—3881.

244. *Creswell A.*, *White T.*, *Dumoulin V.*, *Arulkumaran K.*, *Sengupta B.*, *Bharath A. A.* Generative adversarial networks: An overview [Текст] // IEEE signal processing magazine. — 2018. — Т. 35, № 1. — С. 53—65.

245. *Gui J.*, *Sun Z.*, *Wen Y.*, *Tao D.*, *Ye J.* A review on generative adversarial networks: Algorithms, theory, and applications [Текст] // IEEE transactions on knowledge and data engineering. — 2021. — Т. 35, № 4. — С. 3313—3332.

246. *Croitoru F.-A.*, *Hondru V.*, *Ionescu R. T.*, *Shah M.* Diffusion models in vision: A survey [Текст] // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 2023. — Т. 45, № 9. — С. 10850—10869.

247. *Yang L.*, *Zhang Z.*, *Song Y.*, *Hong S.*, *Xu R.*, *Zhao Y.*, *Zhang W.*, *Cui B.*, *Yang M.-H.* Diffusion models: A comprehensive survey of methods and applications [Текст] // ACM computing surveys. — 2023. — Т. 56, № 4. — С. 1—39.

248. *Goodfellow I. J.*, *Pouget-Abadie J.*, *Mirza M.*, *Xu B.*, *Warde-Farley D.*, *Ozair S.*, *Courville A.*, *Bengio Y.* Generative adversarial nets [Текст] // Proceedings of the 28th Conference on Neural Information Processing Systems (NIPS 2014). — 2014.

249. *Karras T.*, *Aittala M.*, *Laine S.*, *Härkönen E.*, *Hellsten J.*, *Lehtinen J.*, *Aila T.* Alias-free generative adversarial networks [Текст] // Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021). — 2021.

250. *Rombach R.*, *Blattmann A.*, *Lorenz D.*, *Esser P.*, *Ommer B.* High-resolution image synthesis with latent diffusion models [Текст] // Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR 2022). — 2022. — С. 10684—10695.

251. *Pogančić M. V.*, *Paulus A.*, *Musil V.*, *Martius G.*, *Rolinek M.* Differentiation of blackbox combinatorial solvers [Текст] // Proceedings of the 8th International Conference on Learning Representations (ICLR 2020). — 2020.

252. *Kingma D. P.*, *Ba J.* Adam: A Method for Stochastic Optimization [Текст] // Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015). — 2015.

253. *Smith L. N.*, *Topin N.* Super-convergence: Very fast training of neural networks using large learning rates [Текст] // Artificial intelligence and machine learning for multi-domain operations applications. T. 11006. — SPIE. 2019. — С. 369—386.

Список рисунков

Список таблиц