Федеральное государственное автономное образовательное учреждение высшего образования «Российский университет дружбы народов имени Патриса Лумумбы»

На правах рукописи

Андрейчук Антон Андреевич

Методы конфликтно-ориентированного поиска для планирования совокупности безопасных траекторий мобильных агентов с учетом возможности совершения действий произвольной продолжительности

Специальность 1.2.3 Теоретическая информатика, кибернетика

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
кандидат физико-математических наук
Яковлев Константин Сергеевич

Москва — 2023

Оглавление

Введение

Актуальность темы исследования. Согласно отчету компании China Post Group [1] объем пересылаемых почтовых отправлений только в Китае по итогам 2021 года превысил 100 млрд единиц. Розничные и технологические гиганты вкладывают значительные средства в интеллектуальные склады, чтобы своевременно и с минимальными затратами обрабатывать возросший спрос на логистику электронной коммерции. Одним из наиболее ярких примеров автоматизированных складов являются сортировочные центры компании Amazon [2]. Аналогичные системы развиваются и другими гигантами в сфере электронной коммерции, такими как, например, Alibaba [3].

Концепция интеллектуальных складов не ограничивается индустрией электронной коммерции. Другие отрасли также занимаются их разработкой и внедрением, чтобы справиться с растущим спросом на логистические и транспортные услуги. Автономные склады считаются одним из ключевых компонентов, необходимых для перехода к так называемой индустрии 4.0. Объем рынка складской робототехники по итогам 2020 года оценивался в 14,7 млрд $ с перспективами роста до 53 млрд $ к 2030 году [4].

Одной из задач, возникающей при разработке интеллектуальных систем для управления группой роботов, является задача согласованного перемещения. Для того, чтобы роботы-грузчики могли эффективно выполнять задачи связанные с перемещением товаров, их сортировкой и компоновкой, необходимо иметь возможность автоматически строить для них согласованные траектории перемещения, которые позволяют с одной стороны выполнять поставленные роботам задачи как можно эффективнее, с другой – избегать столкновений между роботами, следующими вдоль построенных траекторий.

Аналогичные задачи, связанные с согласованными перемещениями, возникают и в других областях – интеллектуальных транспортных системах, при мониторинге территорий, ликвидации чрезвычайных происшествий. Не смотря на очевидные различия между примерами, в которых возникают задачи согласованного перемещения, принцип работы подходов к решению этих задач идентичен. В компьютерных науках, в частности в искусственном интеллекте, их называют задачами многоагентного планирования (англ. multi-agent pathfidning). Даже в случае использования ряда допущений, таких как дискретное представ-

ление пространства поиска в виде графа специального вида и дискретизации времени, поиск оптимального решения задачи многоагентного планирования относится к классу NP-полных задач. На сегодняшний день существует ряд алгоритмов, гарантирующих, что найденное ими решение является оптимальным. Однако большинство из них опирается на допущение о том, что все действия агентов имеют одинаковую продолжительность. Это допущение ограничивает набор возможных действий для агентов, и приводит к тому, что в действительности могут существовать решения, обладающие лучшим качеством за счет учета возможности совершения большего набора действий.

Ввиду вышеизложенного в диссертацию вошли исследования трех основных направлений: 1) исследование и разработка алгоритма решения задачи многоагентного планирования, который с одной стороны учитывает возможность совершения агентами действий произвольной продолжительности, с другой – гарантирует, что найденное им решение является оптимальным; 2) исследование и разработка модификаций алгоритма, позволяющих повысить его вычислительную эффективность, сохранив при этом гарантию нахождения оптимальных решений; 3) исследование и разработка модификаций алгоритма, позволяющих повысить его вычислительную эффективность за счет отказа от поиска оптимальных решений и переходу к поиску субоптимальных решений.

Степень разработанности. Для анализа текущего состояния области был проведен обзор существующих постановок задачи многоагентного планирования, а также методов и алгоритмов их решения. Основными отечественными учеными, занимающимися исследованиями и внесшими существенный вклад в область планирования, в том числе многоагентного, являются М.Л. Цетлин, В.И. Варшавский, Д.А. Поспелов, Г.С. Осипов, В.Г. Конюший, Б.С. Юдинцев, Ю.И. Нечаев, В.Э. Карпов, Л.Ю. Жилякова. Основными зарубежными исследователями, внесшими вклад в создание методов решения задачи многоагентного планирования, являются S. Koenig, M. Likhachev, R. Stern, A. Felner, N. Sturtevant, D. Silver, D. Harabor, H. Ma, W. Ruml, A. Botea, H. Choset, R. Bartak, P. Surynek.

Целью диссертационной работы является разработка и исследование методов и алгоритмов решения задачи многоагентного планирования траекторий с учетом возможности совершения агентами действий произвольной продолжительности.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. Разработать алгоритм планирования совокупности неконфликтных траекторий, опирающийся на принцип конфликтно-ориентированного поиска и допускающий возможность совершения действий произвольной продолжительности. Провести исследование теоретических свойств разработанного алгоритма, в частности, доказать утверждение об оптимальности решений, отыскиваемых алгоритмом.
2. Разработать модификации алгоритма, позволяющие повысить его вычислительную эффективность и сохраняющие при этом теоретические гарантии нахождения оптимальных решений. Провести исследование теоретических свойств разработанных модификаций.
3. Разработать модификации алгоритма, позволяющие повысить его вычислительную эффективность за счет перехода к поиску ограниченно-субоптимальных решений с возможностью настройки коэффициента субоптимальности. Провести исследование теоретических свойств разработанных модификаций.

Научная новизна:

1. Для возможности осуществления действий произвольной продолжительности был предложен оригинальный способ описания конфликтов, были введены интервальные ограничения, а также предложен алгоритм планирования индивидуальных траекторий, способный учитывать подобные ограничения.
2. Предложенные в работе модификации, такие как приоритизация конфликтов, непересекающееся разделение и эвристики верхнего уровня позволяют повысить вычислительную эффективность разработанного алгоритма и при этом сохранить свойство оптимальности.
3. Предложенные в работе модификации, комбинирующие подход конфликтно ориентированного поиска с субоптимальными алгоритмами планирования, отыскивают ограниченно субоптимальные решения и имеют регулируемый параметр, отвечающий за фактор субоптимальности, что позволяет найти требуемый баланс между качеством отыскиваемых решений и вычислительными ресурсами, затрачиваемыми на их поиск.

Методы исследования. В диссертационной работе применяются методы теории графов, линейного программирования, вычислительной геометрии, исследования операций.

Основные положения, выносимые на защиту:

1. Разработанный алгоритм планирования совокупности неконфликтных траекторий мобильных агентов с учетом возможности совершения действий произвольной продолжительности. Сформулированная и доказанная теорема об оптимальности решений, отыскиваемых предложенным алгоритмом.
2. Предложенные в работе модификации алгоритма планирования, позволяющие существенно повысить его вычислительную эффективность и при этом сохраняющие свойство оптимальности. Сформулированная и доказанная теорема об оптимальности решений, отыскиваемых алгоритмом, использующим предложенные модификации.
3. Предложенные в работе модификации алгоритма многоагентного планирования, отыскивающие субоптимальные решения, позволяющие выбрать требуемый баланс между вычислительной эффективностью алгоритма и качеством отыскиваемых решений. Сформулированные и доказанные утверждения об ограниченной субоптимальности решений, отыскиваемых предложенными субоптимальными алгоритмами.

Обоснованность и достоверность результатов следует из корректного и строгого применения методов дискретной математики и математической логики при проведении исследования, в частности, при доказательстве теоретических свойств предложенных алгоритмов. Достоверность результатов дополнительно подтверждена результатами численных экспериментов, проведенных на данных, полученных из открытых источников, широко используемых в научном сообществе в области многоагентного планирования. Обоснованность и достоверность полученных в ходе исследования результатов также подтверждается их апробацией на ведущих научных конференциях и семинарах в области автоматического планирования.

Теоретическая и практическая значимость работы обусловлена тем, что разработанный алгоритм многоагентного планирования обладает уникальным набором свойств, включающим в себя учет возможности совершения действий произвольной продолжительности, а также гарантию оптимальности отыскиваемых решений. Свойство оптимальности было теоретически доказано. Благодаря

учету возможности совершения агентами действий произвольной продолжительности алгоритм может находить решения, качество которых лучше, чем у существующих аналогов, а свойство оптимальности гарантирует, что отыскиваемые алгоритмом решения не могут быть улучшены (при тех же допущениях и входных данных).

Часть представленных результатов была получена в рамках работ по гранту РНФ №16-11-00048 «Создание теории, методов и моделей децентрализованного управления поведением коллективов когнитивных робототехнических систем в недетерминированной среде», а также в рамках проекта «5-100» повышения конкурентоспособности ведущих российских университетов среди ведущих мировых научно-образовательных центров.

Соответствие паспорту специальности. Диссертация выполнена в соответствии с паспортом научной специальности 1.2.3 «Теоретическая информатика, кибернетика». В соответствии с п.9 «Математическая теория исследования операций» в работе рассматривается задача многоагентного планирования и исследуются методы поиска её оптимальных решений. В соответствии с п.29 «Теоретические основы программирования, создания программных систем для новых информационных технологий» проведена разработка, реализация, теоретическое и экспериментальное исследование алгоритма, решающего задачу многоагентного планирования с возможностью совершения действий произвольной продолжительности, который может быть применен при разработке роботизированных интеллектуальных систем, в которых возникает задача согласованного перемещения группы роботов в общем рабочем пространстве.

Апробация работы. Результаты работы докладывались на следующих конференциях:

1. The 35th AAAI Conference on Artificial Intelligence, 2-7 February 2021, Online
2. The 28th International Joint Conference on Artificial Intelligence (IJCAI), 10-16 August 2019, Macao, China
3. IJCAI-19 Workshop on Multi-Agent Path Finding, 12 August 2019, Macao, China
4. The 14th International Symposium on Combinatorial Search, 26-30 July 2021, Online

5. Семнадцатая Национальная конференция по искусственному интеллекту с международным участием, КИИ-2019, 21-25 октября 2019 г., Ульяновск, Россия
6. Восемнадцатая Национальная конференция по искусственному интеллекту с международным участием, КИИ-2020, 10-16 октября 2020 г., Москва, Россия

Личный вклад. Автор принимал активное участие в исследованиях, в подготовке и представлении статей и докладов по теме работы. Программная реализация и тестирование алгоритмов, проведение модельных экспериментов, а также обработка полученных результатов производились автором лично. Доказательства теорем, приведенные в тексте диссертации, принадлежат автору и ранее не публиковались. Постановка задачи планирования с возможностью осуществления действий произвольной продолжительности, а также идея использования подхода безопасно-интервального планирования для поиска индивидуальных траекторий принадлежат К.С. Яковлеву. Идея использования интервальных ограничений для устранения конфликтов, а также теоретическое обоснование их согласованности принадлежат R. Stern. Метод расчета эвристики верхнего уровня на основе системы линейных уравнений принадлежит E. Boyarski. Основные результаты и положения, выносимые на защиту, отражают персональный вклад автора.

Публикации. Основные результаты по теме диссертации изложены в 8 печатных изданиях, из которых 2[5],[6] работы изданы в журналах, рекомендованных ВАК, 5[7],[8],[9],[10],[11] опубликованы в изданиях, индексируемых Scopus, в том числе 1[7] статья опубликована в журнале первого квартиля по SJR, 1[12] – в сборнике трудов конференции, индексируемый РИНЦ.

Объем и структура работы. Диссертация состоит из введения, 5 глав, заключения. Полный объём диссертации составляет 127 страниц, включая 28 рисунков и 8 таблиц. Список литературы содержит 102 наименования.

Глава 1. Постановка задачи

Существует большое количество различных практических задач, в которых возникает задача согласованного перемещения группы мобильных агентов. Наиболее яркими примерами применения алгоритмов многоагентного планирования являются системы управления группами мобильных роботов на автономных складах [13](см. Рисунок 1.1), различные транспортные системы [14], управление группами персонажей компьютерных игр [15; 16], ликвидация чрезвычайных происшествий [17], мониторинг территорий [18].

В зависимости от решаемой задачи, различаются и постановки. Однако, во всех них присутствует некоторое общее рабочее пространство, в котором оперирует множество агентов, для которых заданы стартовые и целевые положения. Задача заключается в том, чтобы спланировать для каждого из агентов траекторию, следуя которой, агент достигнет своего целевого положения и при этом не столкнется с другими агентами.

В большинстве случаев рабочее пространстве моделируется с помощью графа специального вида. Вершины графа соответствуют возможным положени-

Рисунок 1.1 — Пример практического применения алгоритмов многагентного планирования. Множество мобильных роботов заняты сортировкой товаров на складе компании Alibaba. Изображение взято из [19]

ям агентов в пространстве, а ребра - возможным перемещениям. При решении задач многоагентного планирования зачастую используют допущение о том, что все действия агентов имеют одинаковую продолжительность, благодаря чему время может быть дискретизовано. Наиболее подходящей структурой графа, в котором выполняется это допущение, является граф регулярной декомпозиции (ГРД) [20], в котором каждая вершина соответствует некоторой области пространства и может быть либо проходима, либо заблокирована в зависимости от наличия препятствия в соответствующей области пространства. Перемещаться агенты могут лишь в четырех ортогональных направлениях, а также совершать действие ожидания, оставаясь в той же вершине графа. Сами агенты фактически представляют собой материальные точки, а все возможные конфликты между ними определяются через их положения на графе в конкретные моменты времени (подробнее см. раздел 1.1). Более того, как правило, действуют допущения о том, что рабочее пространство является полностью наблюдаемым, статическим или, по крайней мере, детерминированным, имеется возможность централизованного управления всеми агентами и они идеально исполняют спланированные траектории.

Однако, существует направление, в котором рассматривается постановка задачи, где отсутствует централизованный планировщик и агенты вынуждены либо действовать независимо, опираясь лишь на собственные локальные наблюдения окружающего пространства [21], либо имеют ограниченные возможности для коммуникации, находясь, например, на определенном расстоянии или при прямой видимости друг друга [22; 23].

Существуют также различия в том, что происходит с агентами, достигшими своих целевых положений. Как правило, считается, что агенты продолжают занимать и блокировать для прохода вершину, соответствующую их целевому положению. Однако, есть работы [24], в которых рассматриваются задача, когда агенты, достигшие своих целевых положений, исчезают. В работах [25; 26] рассматривается, так называемая, life-long постановка задачи, когда агентам, достигшим своих целевых положений, назначаются новые цели. Такая постановка задача наиболее характерна для логистических задач, где постоянно появляются новые задачи по транспортировке грузов. В работах [27; 28] рассматривается постановка задачи, которая допускает, что агенты в процессе исполнения спланированных траекторий могут совершать незапланированные задержки, и для их

учета требуется спланировать траектории таким образом, чтобы агенты не столкнулись даже в случае их совершения.

Как было сказано ранее, при решении задачи многоагентного планирования, агентам заранее заданы их целевые положения. При этом, как правило, у каждого агента есть своё собственное уникальное целевое положение. В работах [29; 30] рассматривается постановка задачи, когда целевые положения заранее не распределены и задача включает в себя необходимость распределить целевые положения между агентами. В работах [31; 32] рассматривается постановка, когда задачей является не достижение всеми агентами своих целевых положений, а последовательное посещение определенных положений на графе.

В работах [33; 34] рассматривается постановка задачи, когда агенты имеют тело и они одновременно занимают несколько вершин графа. В работах [35; 36] учитываются не только размеры и формы агентов, но и кинематические ограничения, накладываемые моделью движения агентов. В работах [37; 38] рассматривается постановка задачи, допускающая возможность совершения действий различной продолжительности. Однако, продолжительность всех действий должна быть кратна шагу дискретизации времени.

В данной работе будет рассматриваться постановка задачи, которая допускает возможность совершения действий произвольной продолжительности. Прежде чем описывать постановку задачи, решаемой непосредственно в работе, стоит подробно описать так называемую классическую постановку задачи многоагентного планирования [39].

1.1 Классическая постановка задачи

Классическая постановка задачи многоагентного планирования задается тройкой $\langle \mathcal{G}, Starts, Goals \rangle$, где:

1. $\mathcal{G} = \langle V,E \rangle$ – неориентированный граф, определяющий пространство, в котором оперируют агенты. V – множество вершин, соответствующие возможным положениям агентов в пространстве. E – множество ребер, задающее возможные перемещения между положениями агентов.
2. $Starts$ – множество стартовых вершин: $Starts = \{v_{s_1}, v_{s_2}, ..., v_{s_k}\}$
3. $Goals$ – множество целевых вершин: $Goals = \{v_{g_1}, v_{g_2}, ..., v_{g_k}\}$

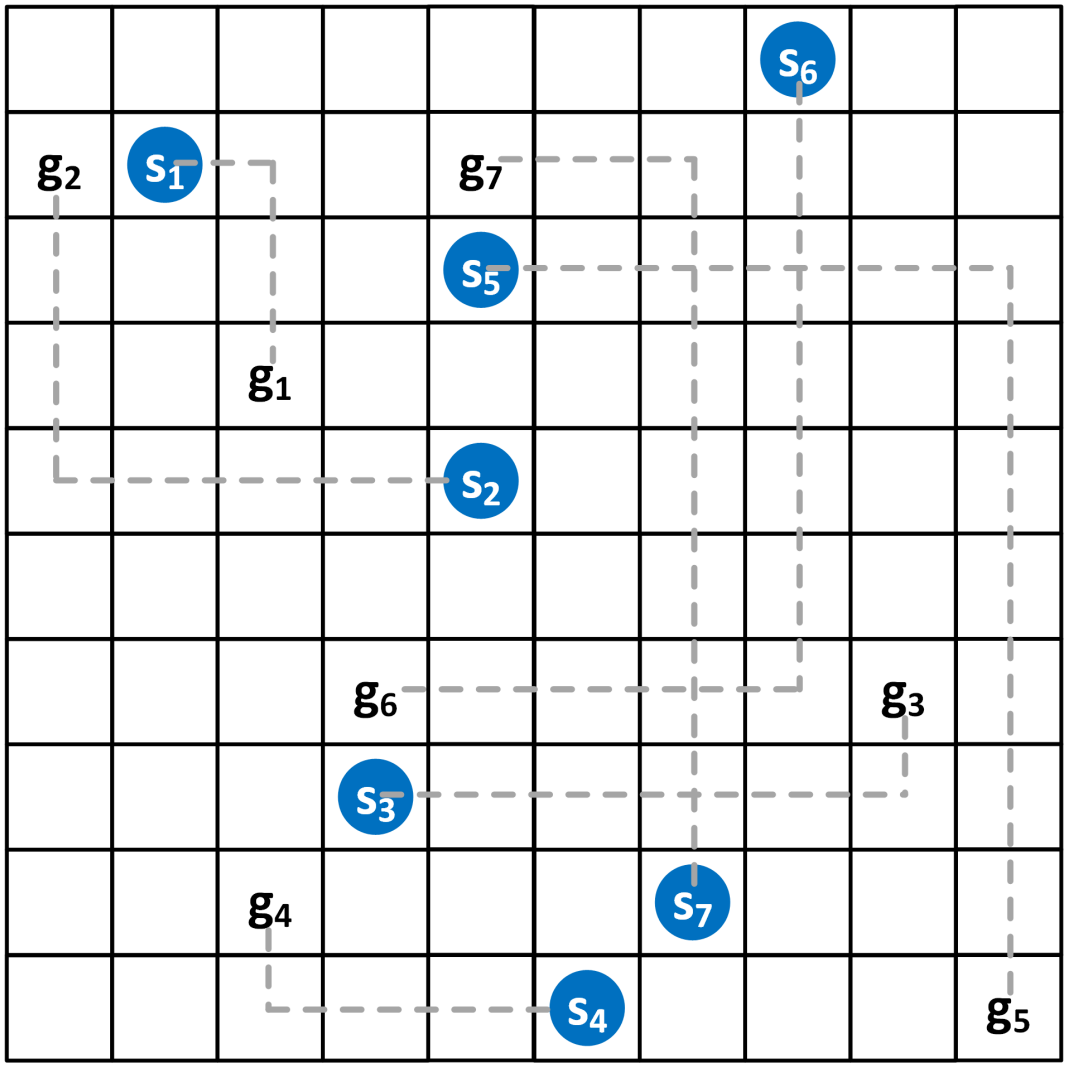


Рисунок 1.2 — Пример задачи многоагентного планирования в классической постановке.

При этом множества стартовых и целевых положений должны удовлетворять следующим критериям:

1. Мощности множеств $Starts$ и $Goals$ должны быть равны: $|Starts| = |Goals|$.
2. Стартовые и целевые положения являются уникальными для каждого агента: $\nexists v_{s_i}, v_{s_j} \in Starts : v_{s_i} = v_{s_j}, i \neq j$, $\nexists v_{g_i}, v_{g_j} \in Goals : v_{g_i} = v_{g_j}, i \neq j$. При этом допускается наличие пересечений между множествами $Starts$ и $Goals$, то есть стартовое положение агента может являться целевым для него же или для какого-либо другого агента.

При решении задачи многоагентного планирования необходимо также учитывать временной аспект. В классической постановке задачи время дискретизировано. В каждый момент времени t каждый агент занимает определенную вершину $v \in V$. Действия агентов задаются с помощью функции $a : V \to V$, такой

что $a(v) = v'$, т.е. в текущий момент времени агент находится в вершине v, а в следующий – в v'. Каждый такт времени агенты совершают одно из возможных действий – либо действие-*ожидание*, т.е. агент продолжает оставаться в том же положении, либо действие–*перемещение*, т.е. агент совершает переход в одну из смежных вершин графа.

Последовательность действий $\pi_i = (a_1, \ldots, a_n)$ является индивидуальной траекторией агента i, если она позволяет перейти из его стартового положения в целевое: $a_n(a_{n-1}(\cdots(a_1(Starts(i)))\cdots)) = Goals(i)$. **Решением** классической задачи многоагентного планирования является совокупность индивидуальных траекторий k агентов: $\Pi = \{\pi_1, \cdots, \pi_k\}$.

Решение задачи многоагентного планирования является действительным, если индивидуальные траектории агентов, его составляющие, не содержат конфликтов. В классической постановке задачи, как правило, рассматриваются два типа конфликтов – а) вершинный; б) реберный. Для формального описания типов конфликтов введем обозначение $\pi_i(t)$ – положение агента i в момент времени t, следующего вдоль траектории π_i.

Первый тип конфликтов возникает тогда, когда два агента находятся в одной и той же вершине графа:

$$\exists t : \pi_i(t) = v \wedge \pi_j(t) = v, v \in V \tag{1.1}$$

Второй тип конфликтов возникает тогда, когда два агент одновременно проходят через одно и то же ребро графа, фактически, меняясь положениями:

$$\exists t : \pi_i(t) = v_i \wedge \pi_j(t) = v_j \wedge \pi_i(t+1) = v_j \wedge \pi_j(t+1) = v_i, v_i, v_j \in V \tag{1.2}$$

В литературе встречаются и другие типы конфликтов, такие как, например, конфликт следования, когда два агента в последовательные моменты времени занимают одну и ту же вершину:

$$\exists t : \pi_i(t) = v \wedge \pi_j(t+1) = v, v \in V \tag{1.3}$$

Разновидностью конфликта следования является циклический конфликт, когда $n > 2$ агентов циклично меняются положениями.

$$\begin{aligned} \exists t :& \pi_i(t) = v_i \wedge \pi_j(t) = v_j \wedge \pi_k(t) = v_k \wedge \\ & \pi_i(t+1) = v_j \wedge \pi_j(t+1) = v_k \wedge \pi_k(t+1) = v_i, v_i, v_j, v_k \in V \end{aligned} \tag{1.4}$$

Примеры всех вышеописанных типов конфликтов показан на Рисунке 1.3.

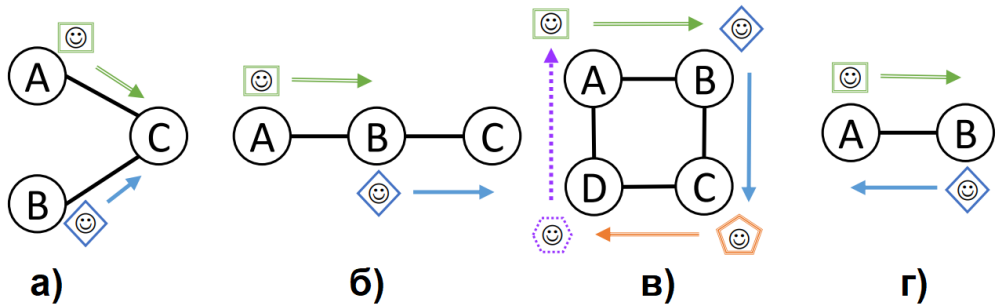


Рисунок 1.3 — Примеры различных типов конфликтов: а) - вершинный, б) - следования, в) циклический, г) - реберный.

В классической постановке задачи все действия имеют одинаковую продолжительность. Благодаря этому, количество действий, составляющих траекторию, определяют её *стоимость*, обозначаемую как $cost(\pi_i)$. Качество решения оценивается по одному из двух следующих критериев:

1. Суммарная стоимость всех траекторий: $cost(\Pi) = \sum_{i=1}^{N} cost(\pi_i)$. Эту оценку можно интерпретировать как совокупные энергозатраты, необходимые для достижения каждым из агентов своего целевого положения. В англоязычной литературе для обозначения этого критерия используется термин flowtime, либо же SOC (аббревиатура sum of costs).
2. Максимальная стоимость среди всех траекторий: $cost(\Pi) = max(cost(\pi_i))$, $i = (1,N)$. Фактически эта оценка соответствует моменту времени, когда все агенты достигнут своих целевых положений. Следовательно, эту оценку можно интерпретировать как время, которое требуется для выполнения поставленной задачи. В англоязычной литературе для обозначения этого критерия используется термин makespan.

Несмотря на то, что классическая постановка задачи не накладывает ограничений на структуру графа, наиболее часто в работах, посвященных методам и алгоритмам решения задачи многоагентного планирования, используется граф специальной структуры – граф регулярной декомпозиции (ГРД) (в англоязычной литературе используется термин $grid$).

На Рисунке 1.2 показан пример задачи многоагентного планирования в классической постановке. Для того, чтобы соблюсти ограничение на одинаковую продолжительность всех действий, агенты могут совершать переходы только в ортогонально-смежные вершины графа. Помимо действий перемещений, также разрешено совершение действия ожидания продолжительностью эквивалентной продолжительности действий перемещений.

1.1.1 Ограничения классической постановки задачи

Одним из наиболее рестриктивных допущений классической постановки задачи многоагентного планирования является допущение о том, что все действия имеют одинаковую продолжительность. Это допущение позволяет дискретизовать время и упростить многие процедуры связанные с процессами планирования и согласования действий агентов. Однако, это допущение приводит к возникновению следующих двух ограничений:

1. **Все действия-перемещения имеют одинаковую продолжительность эквивалентную одному шагу времени.** Это значит, что либо все агенты движутся с одинаковой скоростью и все ребра графа соответствуют переходам одинаковой длины, либо же переходы имеют различную длину и агенты подстраивают свои скорости так, чтобы все перемещения имели одинаковую продолжительность.
2. **Все действия-ожидания имеют одинаковую продолжительность эквивалентную одному шагу времени.** Это значит что агенты не могут ждать произвольное количество времени. Продолжительность действий-ожиданий должна быть кратной дискретному шагу времени.

В данной работе рассматривается и решается постановка задачи лишенная этого допущения. Рассмотрим пример, приведенный на Рисунке 1.4, на котором изображены два решения одной и той же задачи. Дан граф, состоящий из восьми проходимых вершин, а также три агента. Агентам разрешен переход только в ортогонально-смежные вершины.

Верхняя часть рисунка показывает действия агентов, которые они совершают, при условии, что все действия имеют одинаковую продолжительность равную 1 у.е. времени, а нижняя – с возможностью совершения действий произвольной продолжительности. Для избегания столкновения с агентом 1, агент с индексом 3 совершает действие ожидания. В дискретном случае продолжительность этого действия равна 1. Однако, для того чтобы избежать столкновения с агентом 1, достаточно совершить ожидание продолжительностью $\sqrt{2}/2$. В результате того, что агент 3 начал двигаться раньше, он быстрее достиг своего целевого положения и агенту 2 также пришлось ожидать меньшее количество времени пока агент 3 пройдет. В результате, стоимость решения с дискретной продолжительностью действий равна 9, а в случае с произвольной – 8.414.

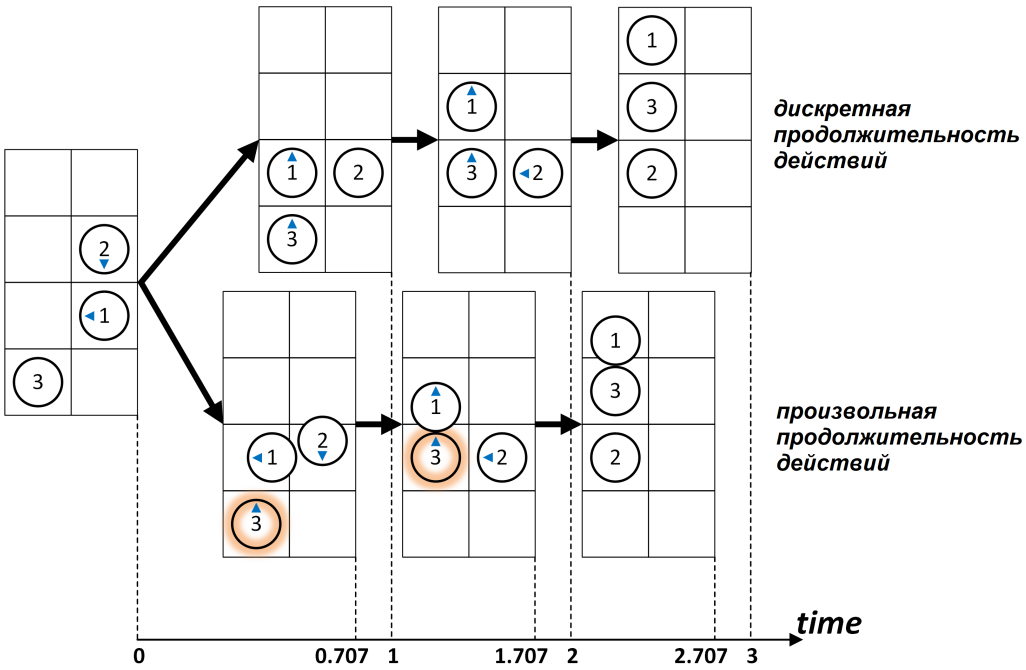


Рисунок 1.4 — Сравнение двух решений, полученных без/с учетом возможности совершения действий произвольной продолжительности.

Стоит отметить, что в случае, когда действия агентов могут иметь произвольную продолжительность, необходимо учитывать размеры и формы агентов, т.к. от этого зависит в том числе продолжительность действия ожидания. В данном примере все агенты представлены в форме диска радиусом $\sqrt{2}/4$. Это значение является максимально возможным, которое позволяет агентам совершать действия без конфликта *следования*, которое не рассматривается в классической постановке задачи многоагентного планирования. Например, если бы агенты имели радиус 0.5, то агент 2 смог бы совершить действие движения вниз только в момент времени $\sqrt{(2)} - 1$, в то время как в дискретной постановке задачи, он может совершить его в момент времени 0, одновременно с действием агента 3.

Данный пример показывает, что даже в случае использования модели графа, в котором все действия перемещения имеют одинаковую продолжительность, возможность совершения действий ожиданий произвольной продолжительности позволяет находить решения, обладающие меньшей стоимостью. Более того, произвольная продолжительность действий позволяет рассматривать графы регулярной декомпозиции более высокой связности, например, с возможностью совершения диагональных действий перемещений, а также использовать графы нерегулярной структуры.

1.2 Планирование с учетом действий произвольной продолжительности

Задача задается набором $\langle G, \mathcal{M}, \mathcal{A}, Starts, Goals, K\rangle$, где $G = \langle V, E\rangle$ – граф, задающий возможные положения агентов, $\mathcal{M}$ – метрическое пространство, в которое вложен граф G, $\mathcal{A} = \{\mathcal{A}_w, \mathcal{A}_m\}$, $\mathcal{A}_w$ – множество действий ожиданий, $\mathcal{A}_m$ – множество действий перемещений, $Starts$, $Goals$ – множества стартовых и целевых положений, K – число агентов.

Каждое действие $a \in \mathcal{A}$ определено парой $\langle a_\varphi, a_D\rangle$, где a_D – продолжительность действия, а a_φ – функция, задающая положение агента в процессе исполнения действия. При этом действия перемещения привязаны к ребрам из множества E: $\forall a \in \mathcal{A} : \exists v, v' \in V : a_\varphi(0) = coord(v), a_\varphi(a_D) = coord(v'), (v,v') \in E$, где $coord(v)$ – координаты вершины v, определяющие её положение в метрическом пространстве $\mathcal{M}$. Действия ожидания также возможно совершать только в положениях соответствующих вершинам из множества V. В зависимости от типа действия a_φ, a_D определены следующим образом:

$$
\begin{aligned}
&\forall a \in A_m : \\
&\quad \forall t \in [0, a_D] : a_\varphi(t) = coord(v) + (coord(v') - coord(v))t/a_D \\
&\quad a_D = ||coord(v) - coord(v')||_2 \\
&\forall a \in A_w : \\
&\quad \forall t \in [0, a_D] : a_\varphi(t) = coord(v) \\
&\quad a_D \in \mathbb{R}^+
\end{aligned}
\tag{1.5}
$$

Выражение 1.5 означает, что агент при исполнении действий перемещений движется с постоянной скоростью, причем такой, что время исполнения действия эквивалентно расстоянию между соответствующими вершинами. При исполнении действий ожиданий агент находится в вершине v на протяжении исполнения всего действия. Стоит отметить, что хотя множество всех действий перемещений $\mathcal{A}$ является конечным, множество всех действий ожиданий является бесконечным, так как допускается, что для действий ожиданий величина a_D может иметь произвольное положительное значение.

Определение 1. Траектория π представляет собой последовательность пар действий и моментов времени $\pi_i = \{(a_1, t_1), \ldots, (a_k, t_k)\}$. При этом продолжитель-

ность траектории π_D и функция π_φ определены следующим образом:

$$\pi_D = \sum_{a \in \pi} a_D \tag{1.6}$$

$$\pi_\varphi(t) = \begin{cases} a_{1\varphi}(t) & t \leqslant a_{1D} \\ \dots & \dots \\ a_{j\varphi}(t - (\pi[:j-1])_D) & (\pi[:j-1])_D < t \leqslant (\pi[:j])_D \\ \dots & \dots \\ a_{n\varphi}(t - (\pi[:n-1])_D) & (\pi[:n-1])_D < t \leqslant (\pi[:n])_D \\ a_{n\varphi}(a_{nD}) & t > (\pi[:n])_D \end{cases} \tag{1.7}$$

, где $\pi[:j]$ – это часть траектории π, состоящая из первых j действий.

Исполнив все действия, составляющие траекторию π_i, агент i перейдет из своего стартового положения $Starts(i)$ в целевое – $Goals(i)$. Выражение 1.7 определяет положение агента в любой момент времени в процессе исполнения заданной траектории. Для этого сперва определяется действие, которое агент совершает в момент времени t, после чего используется функция a_φ соответствующего действия. Последняя строка выражения означает, что после того как агент выполнил все действия, агент продолжает находиться в вершине, которая соответствует его целевому положению.

Находясь в положении вершины v, агент может совершить любое действие, начинающееся из этой вершины. При этом часть этих действий может приводить к конфликтам с другими агентами. Под конфликтом подразумевается ситуация, когда агенты находятся в пространстве-времени настолько близко, что их тела пересекаются. Для определения конфликта введем функцию $InCollision : \{1,\dots,K\} \times \{1,\dots,K\} \times \mathcal{M} \times \mathcal{M} \to \{true, false\}$, где $InCollision(i, j, \pi_{i\varphi}(t), \pi_{j\varphi}(t) = true$ означает, что между агентами i и j, которые в момент времени t находятся в положениях $\pi_{i\varphi}(t)$ и $\pi_{j\varphi}(t)$ соответственно, происходит конфликт. Без ограничения общности будем считать, что каждый агент представляет собой диск радиусом r. Тогда конфликт между агентами происходит в тех случаях, когда расстояние между ними меньше чем сумма их радиусов:

$$InCollision(i, j, \pi_{i\varphi}(t), \pi_{j\varphi}(t)) = \begin{cases} True & ||\pi_{i\varphi}(t) - \pi_{j\varphi}(t)||_2 < 2r \\ False & \text{В остальных случаях} \end{cases} \tag{1.8}$$

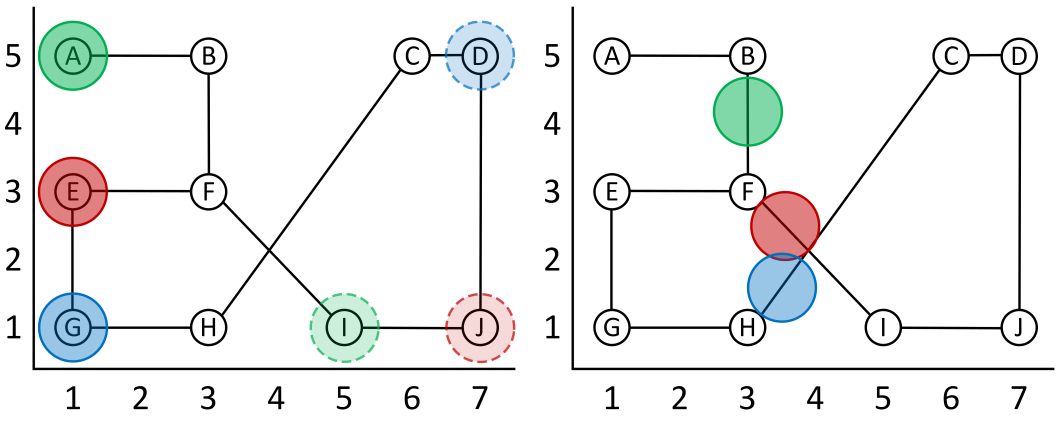


Рисунок 1.5 — Пример задачи с 3 агентами. Слева показан момент времени $t = 0$, когда все агенты находятся в своих стартовых положениях. Круги с пунктирными линиями обозначают целевые положения соответствующих агентов. Справа показаны положения агентов в момент времени $t = 2.8$, в котором происходит конфликт, если все агенты будут двигаться по оптимальным индивидуальным траекториям, спланированным независимо.

На Рисунке 1.5 показан пример задачи с произвольным графом, а также пример конфликта для случая дискообразных агентов.

Для того, чтобы задача могла иметь решение, на множества $Starts$ и $Goals$ накладываются следующие ограничения:

1. Мощности множеств $Starts$ и $Goals$ должны быть эквивалентны: $|Starts| = |Goals|$.
2. Все элементы из множеств $Starts$ и $Goals$ должны принадлежать множеству вершин V: $n \in V, \forall n \in Starts \cup Goals$.
3. Между агентами, находящимися в своих стартовых или целевых положениях, не должно быть конфликтов:

$$\begin{aligned} &InCollision(i,j, coord(Starts(i)), coord(Starts(j)) = False, \\ &InCollision(i,j, coord(Goals(i)), coord(Goals(j)) = False, \\ &\forall i,j = (1,N), i \neq j. \end{aligned} \tag{1.9}$$

Решением задачи является совокупность неконфликтных траекторий $\Pi = \{\pi_1, ..., \pi_K\}$.

Определение 2. Пара траекторий π_i, π_j являются конфликтными, если существует момент времени, когда между агентами происходит конфликт в процессе

исполнения этих траекторий:

$$\exists t \in [0, \max(\pi_{i_D}, \pi_{j_D})] \;\; InCollision\big(i, j, \pi_{i_\varphi}(t), \pi_{j_\varphi}(t)\big) \tag{1.10}$$

Стоимость траектории эквивалента её продолжительности:

$$cost(\pi) = \pi_D \tag{1.11}$$

Качество (стоимость) решения оценивается с помощью суммарной стоимости всех траекторий:

$$cost(\Pi) = \sum_{i=1}^{K} cost(\pi_i) \tag{1.12}$$

Определение 3. Решение Π является оптимальным решением задачи многоагентного планирования, если оно удовлетворяет следующим двум критериям:

1. $\nexists \Pi' : cost(\Pi') < cost(\Pi)$
2. $\forall i, j \in [1, K] : \nexists t \in [0, \max(\pi_{i_D}, \pi_{j_D})] InCollision\big(i, j, \pi_{i_\varphi}(t), \pi_{j_\varphi}(t)\big)$

Таким образом, задача заключается в том, что имея заданный набор $\langle G, \mathcal{M}, \mathcal{A}, Starts, Goals, K \rangle$, необходимо построить совокупность неконфликтных траекторий, обладающую минимальной суммарной стоимостью, т.е. найти оптимальное решение Π.

1.3 Выводы по главе

В данной главе были рассмотрены различные существующие постановки задачи многоагентного планирования. Была описана классическая постановка задачи, которая обладает одним существенным недостатком – допущение об одинаковой продолжительности всех действий. На Рисунке 1.4 было наглядно показано, что даже в случае ограниченного набора действий перемещений, которые имеют одинаковую продолжительность, возможность совершения действий ожиданий произвольной продолжительности может повысить качество отыскиваемых решений. В связи с этим была сформулирована постановка задачи, которая снимает это ограничение и допускает возможность совершения действий произвольной продолжительности.

Глава 2. Анализ современного состояния области и обзор существующих методов

В отечественной литературе задачи, связанные с групповым управлением, коллективным поведением и многоагентными системами, исследуются с конца 60-х годов прошлого века, начиная с работ Цетлина М.Л. [40], Варшавского В.И. [41], Поспелова Д.А. [42], Стефанюка В.Л. [43] и др.

Актуальные на сегодняшний день работы продолжают развитие идей и концепций, предложенных советскими учеными. В работах [44; 45] рассматриваются различные способы кооперации и коммуникации между агентами, а в работе [46] для организации взаимодействия между агентами предлагается применить модели социального поведения. В серии работ [47—49] исследуются вопросы, связанные с устойчивостью, управляемостью и компенсацией возмущений интеллектуальных динамических систем различных классов, в первую тех, что основанны на правилах, а также на основе семантических сетей.

Одной из задач, возникающей в многоагентных системах, является задача распределения целей между агентами. В работе [50] предлагается эвристический алгоритм решения задачи построения маршрутов для множества агентов-коммивояжеров. Подобные подходы применяются в том числе и для решения различных прикладных задач. Так, в работе [51] предлагается графовый метод решения задачи о назначении локомотивов на участке железной дороги, а в работе [52] предлагается использование мультагентного подхода для решения задачи построения расписаний в системе управления железнодорожным движением.

Стоит отметить ряд работ, в которых рассматриваются различные аспекты оперирования интеллектуальных агентов совместно с человеком. В частности, в работе [53] рассматривается задача коллаборативной робототехники, в которой роботы и люди выполняют действия совместно для достижения единой цели, исследуются различные частные случаи задачи, в том числе при различных функциях затрат у разных видов участников. В работе [54] рассматриваются различные стратегии движения роботов в случаях оперирования в скоплениях других аналогичных роботов и людей.

Также активно исследуются задачи, связанные с согласованным перемещением множества агентов. В работе [55] предлагается подход для планирования траекторий множества мобильных роботов. В отличие от классических под-

ходов, использующих эвристические алгоритмы планирования траекторий на графах, предлагаемый в этой работе подход основан на рекуррентной нейронной сети Хопфилда, с помощью которой строится дискретная «нейронная карта», позволяющая осуществлять планирование траекторий агентов. В работе [56] рассматривается задача планирования траекторий для группы мобильных роботов, при этом среда, в которой они оперируют, не является изначально известной. Для её решения предлагается двухфазный подход, в котором на каждом шаге сначала производится выбор локального лидера, а затем осуществляется выбор направления движения, с учетом положений препятствий и других роботов. В работе [57] исследуется аналогичная задача планирования движений группы мобильных роботов, однако, для её решения предлагается использовать подход, основанный на методе потенциальных полей, который позволяет избегать столкновений со статическими препятствиями и между роботами, используя при этом децентрализованный подход к управлению.

Дальнейший обзор посвящен различным методам и алгоритмам решения задачи многоагентного планирования, использующим постановку задачи, схожую с той, что рассматривается в данной работе. Рассматриваются подходы, решающие задачу в детерминированный среде и графовым представлением пространства поиска. При этом имеется один общий центральный планировщик, обладающий информацией о всех агентах и имеющий возможность их координировать.

2.1 Оптимальные методы решения задачи многоагентного планирования

Для поиска оптимального решения задачи многоагентного планирования могут применяться и классические эвристические алгоритмы, применяемые для индивидуального планирования, такие как, например, A* [58]. Алгоритм A* итеративно исследует пространство поиска, рассматривая возможные состояния в порядке возрастания значения $f(s) = g(s) + h(s)$, где $g(s)$ – стоимость известного пути от стартового состояния до состояния s, а $h(s)$ – эвристическая оценка стоимости пути от s до целевого состояния. На каждом шаге алгоритм производит процедуру “раскрытия”, подразумевающую генерацию состояний-потомков и расчет их f-значений. В случае с индивидуальным планированием, число потомков равно числу действий, которые может совершить агент, для

которого осуществляется планирование. Однако в многоагентном случае все агенты рассматриваются как один мета-агент, для которого возможны различные комбинации движений агентов. Итоговое число возможных состояний-потомков достигает значения N^k, где N – число агентов, а k – число возможных действий. Подобный экспоненциальный рост числа возможных состояний делает алгоритм A* крайне неэффективным в задачах многоагентного планирования.

В работе [59] был предложен ряд модификаций, позволяющих повысить эффективность работы алгоритма A*. Первая модификация OD (Operator Decomposition) вводит промежуточные состояния и генерирует их, делая допущение, что агенты движутся последовательно, а не одновременно. Вторая модификация, названная ID (Independence Detection), позволяет разделить все множество агентов на отдельные группы, между которыми нет взаимодействия, и планировать траектории агентов разных групп независимо, снижая тем самым экспоненциальный рост числа генерируемых и рассматриваемых состояний. Таким образом алгоритм рассматривает не все множество агентов как одного большого мета-агента, а создает множество мета-агентов в процессе своей работы. Еще один способ снижения числа генерируемых состояний был предложен в алгоритме EPEA* (Enhanced Partial-Expansion A*) [60]. Суть этого подхода заключается в частичном раскрытии, т.е. генерации лишь части состояний-потомков, у которых f-значение не превышает f-значения текущей раскрываемой вершины.

В работе [61] был предложен алгоритм ICTS (Increasing Cost Tree Search), который также оперирует в совместном пространстве состояний всех агентов, но при этом кардинально отличается по принципу работы от алгоритма A*. Алгоритм ICTS имеет двухуровневую иерархическую структуру. На верхнем уровне алгоритма строится дерево, в котором каждый элемент соответствует суммарной стоимости траекторий всех агентов, а каждый элемент-потомок отличается от элемента-родителя увеличенной на 1 стоимостью траектории одного из агентов. Нижний уровень алгоритма осуществляет планирование в совместном пространстве состояний и проверяет возможность существования неконфликтного решения с заданными стоимостями траекторий агентов. В работе [62] был предложен ряд методик, позволяющих ускорить процесс проверки существования решения.

Алгоритм CBS (Conflict-Based Search), предложенный в работе [63], также использует двухуровневый иерархический подход, однако, не оперирует в совместном пространстве состояний. На верхнем уровне алгоритм оперирует раз-

личными альтернативными решениями, а нижний уровень используется для планирования индивидуальных траекторий. Т.к. планирование траекторий агентов происходит независимо, между ними могут возникать конфликты. Для их устранения алгоритм накладывает ограничения на агентов, запрещая им находиться в определенных вершинах/ребрах графа в определенные моменты времени. Для алгоритма CBS было предложено большое количество различных модификаций, позволяющих повысить его вычислительную эффективность [64—67] или же применить его к различным постановкам задачи многоагнетного планирования [33; 68; 69]. Благодаря тому, что алгоритм CBS не оперирует в совместном пространстве состояний, он значительно лучше масштабируется к задачам с большим числом агентов. Более подробно о принципе работы алгоритма CBS см. раздел 3.1.

Все вышеперечисленные подходы можно отнести к классу методов эвристического поиска. Они решают задачу многоагентного планирования в явном виде. Однако, существует ряд подходов, которые также гарантируют нахождение оптимального решения, однако решают задачу путем её сведения к другим задачам из класса NP-полных.

В работе [70] был предложен подход сведения задачи многоагентного планирования к задаче проверки удовлетворимости булевых формул (англ. Boolean satisfiability problem, SAT). В этой работе для описания каждого агента в каждый момент времени используется отдельная логическая переменная, в связи с чем его эффективность сильно снижается как с ростом числа агентов, так и с ростом размера графа, описывающего пространство, в котором оперируют агенты. В работе [71] был предложен алгоритм SMT-CBS, который также конвертирует задачу многоагентного планирования в задачу проверки удовлетворимости булевых формул, однако, работает более эффективно, используя подход конфликтно-ориентированного поиска, и создавая дополнительные логические переменные лишь в тех случаях, когда между агентами возникают конфликты.

Задачу многоагентного планирования также возможно свести к задаче целочисленного программирования. Подобный подход рассматривался, например, в работе [72]. Однако, как и в случае с подходом сведения к SAT-задаче, для каждого агента в каждый момент времени требуется своя собственная переменная, что негативно сказывается на эффективности поиска решения. Более эффективный алгоритм, использующий принцип сведения к задаче целочисленного программирования был предложен в работе [73] и в дальнейшем был улучшен в работах [74; 75]. Алгоритм BCP (англ. Branch-and-Cut-and-Price) яв-

ляется двухуровневым и действует по аналогии с алгоритмом CBS, осуществляя планирование индивидуальных траекторий на нижнем уровне, и решая задачу целочисленного программирования на верхнем уровне для разрешения конфликтов между агентами.

Как и в случае с подходами, решающими задачу многоагагентного планирования в явном виде, те подходы, что оперируют в совместном пространстве состояний, эффективно решают лишь небольшие задачи, содержащие малое количество агентов. При этом наиболее эффективными являются те подходы, что опираются на подход конфликтно-ориентированного поиска.

2.2 Субоптимальные методы решения задачи многоагентного планирования

На поиск оптимального решения задачи многоагентного планирования может потребоваться чрезвычайно большой объем времени и вычислительных ресурсов. Одним из возможных способов решения этой проблемы является переход от поиска гарантированно оптимальных решений к поиску субоптимальных решений. При этом, стоит отметить наличие группы алгоритмов, которые отыскивают решения близкие к оптимальным и гарантируют, что их качество не хуже качества оптимального решения более чем на заранее заданный фактор субоптимальности w. Эти алгоритмы, по сути, являются субоптимальными версиями оптимальных алгоритмов, описанных выше.

В работах [76—78] были предложены различные субоптимальные модификации алгоритма CBS. Основное отличие заключается в изменении принципа обхода дерева на верхнем уровне алгоритма. Т.к. стоимость оптимального решения заранее неизвестна, для удовлетворения ограничению на фактор субоптимальности используется его нижняя оценка. Для получения этой оценки используется дерево верхнего уровня алгоритма, в котором хранятся различные частичные решения. В работе [76] была также предложена модификация GCBS (Greedy CBS), которая отыскивает субоптимальные решения, неограниченные фактором субоптимальности.

Аналогичные субоптимальные модификации были предложены и для других оптимальных алгоритмов: ICTS [79], eMDD-SAT[80], eSMT-CBS [81].

Существует также ряд подходов, которые эффективно решают задачу многоагентного планирования, но при этом ни имеют никаких гарантий касательно качества отыскиваемых решений. Одним из наиболее эффективных алгоритмов решения задачи многоагентного планирования является приоритизированный подход [82]. Принцип этого подхода заключается в том, что агентам назначаются приоритеты и их траектории планируются последовательно. Для избегания конфликтов между агентами, агенты должны избегать всех других агентов, имеющих более высокий приоритет, т.е. фактически воспринимать их как динамические препятствия.

Несмотря на высокую вычислительную эффективность, алгоритмы, использующие приоритизированный подход, не гарантируют нахождение решения. В зависимости от выбранной последовательности приоритет алгоритм может найти, либо не найти решение. В работах [83; 84] были предложены способы перебора различных последовательностей приоритетов либо для повышения шанса отыскать решение, либо для повышения его качества.

В работе [24] был предложен алгоритм PBS (Priority Based Search), который по аналогии с алгоритмом CBS, использует двухуровневую иерархическую структуру. На верхнем уровне он накладывает ограничения, которые задают приоритет между агентами, а на нижнем - планирует индивидуальные траектории агентов с учетом положений агентов, имеющих более высокий приоритет. Алгоритм PBS способен эффективно перебирать различные последовательности приоритетов и гарантирует нахождение решения тех задач, которые могут быть решены с помощью приоритизированного подхода.

На Рисунке 2.1 показаны различные примеры задач многоагентного планирования. Крайнюю левую задачу невозможно решить с помощью приоритизированного подхода, т.к. любая последовательность приоритетов приведет к тому, что один из агентов окажется заблокированным без возможности достичь своего целевого положения. Центральную задачу приоритизированный подход сможет решить только в том случае, если выберет правильную последовательность приоритетов. Задача, изображенная на рисунке справа, решается приоритизированным подходом вне зависимости от выбранной последовательности приоритетов.

В работе [85] были сформулированы условия для задачи многоагентного планирования, при соблюдении которых, алгоритмы, использующие приоритизированный подход, гарантированно могут найти решение. Суть этих условий сводится к тому, что между стартовым и целевым положением каждого аген-

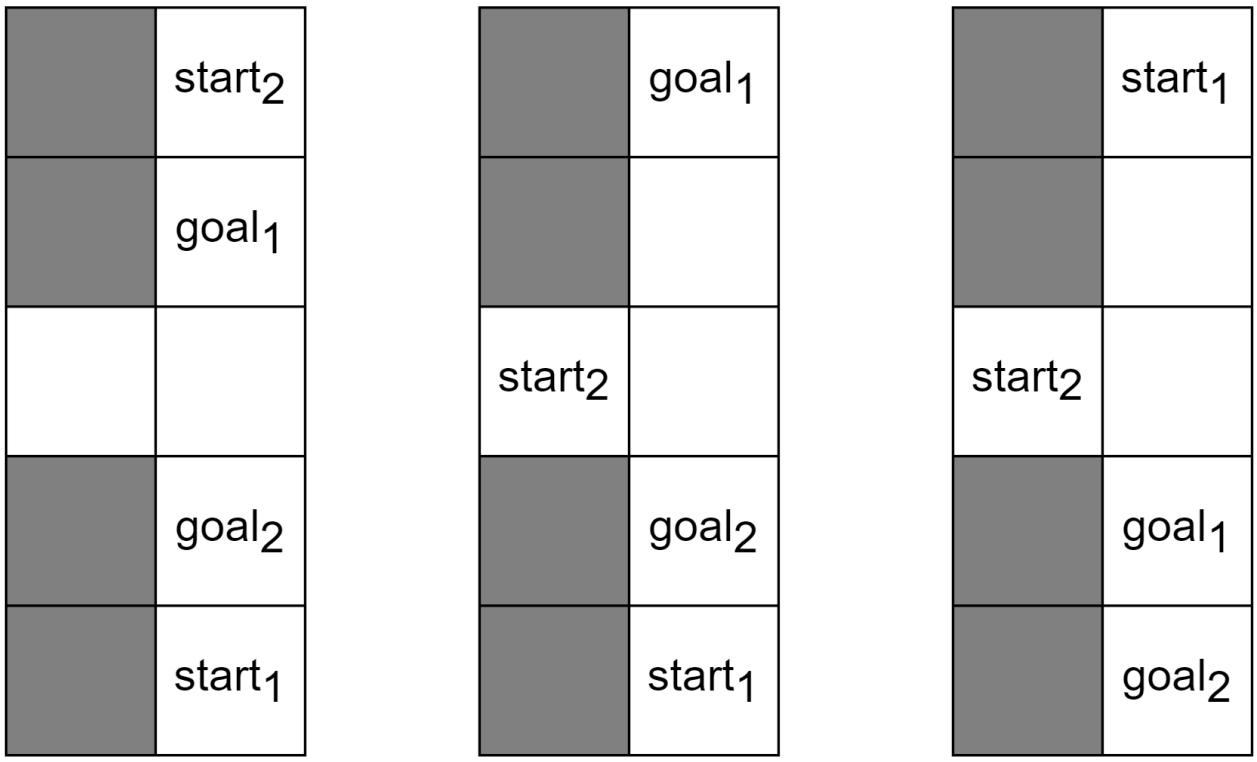


Рисунок 2.1 — Различные примеры заданий на одном и том же графе. Возможность найти решение приоритизированным подходом зависит от расстановки стартовых/целевых положений агентов.

та должна существовать траектория, которая не включает в себя стартовые или целевые положения других агентов. Схожий класс задач был определен в работе [86], в котором был предложен алгоритм MAPP. В этом алгоритме траектории строятся таким образом, чтобы в случае возникновения конфликта, агенты могли разойтись, используя альтернативный путь достижения следующей вершины в траектории.

Существует класс подходов, в которых действия агентов выбираются основываясь на некотором наборе правил. В частности, к ним относятся алгоритмы Push-And-Swap [87] и Push-And-Rotate [88]. Данные алгоритмы гарантируют нахождение решения при условии его существования, а также обладают полиномиальной сложностью. Однако решения, отыскиваемые данными подходами, имеют крайне низкое качество.

В работах [38; 89] были предложены модификации алгоритма CBS, позволяющие оперировать действиями различной продолжительности. Аналогичная модификация для алгоритма ICTS была предложена в работе [37]. Однако, для работы этих алгоритмов требуется дискретизация времени, а все действия имеют продолжительность кратную заданному шагу времени.

2.3 Планирование с учетом действий различной продолжительности

Одной из особенностей большинства вышеупомянутых работ является допущение о том, что все действия имеют одинаковую продолжительность. Однако, как было на примере на Рисунке 1.4, возможность осуществления действий различной продолжительности позволяет повысить качество отыскиваемых решений.

В работе [37] была сформулирована постановка задача, названная $MAPF_R$, которая снимает ограничение на эквивалентную продолжительность всех действий. В постановке задачи $MAPF_R$ любое ребро графа $e = (v,v')$ имеет некоторый положительный вес $w(e) \in \mathbf{R}^+$, который соответствует продолжительности действия, т.е. времени, требуемому для того, чтобы перейти из v в v' по ребру e. Любая вершина $v \in V$ ассоциирована с некоторой точкой в метрическом пространстве – $coord(v)$. В случае, если агент находится в вершине v, это значит, что его положение соответствует $coord(v)$. Когда агент движется по ребру $e = (v, v')$, это значит, что он движется оп прямой с постоянной скоростью из $coord(v)$ в $coord(v')$. При этом агенты занимают некоторый ненулевой объем в этом пространстве. Конфликт между агентами происходит тогда, когда их тела “перекрываются” в некоторый момент времени [37].

Оригинальная постановка задачи $MAPF_R$ вводит возможность совершения действий различной продолжительности, однако, не вводит разграничений между действиями перемещениями и действиями ожиданиями. При этом алгоритм, предложенный в этой работе для решения задачи в постановке $MAPF_R$ - E-ICTS (Extended Increased Cost Tree Search), опирается на то, что действия ожидания имеют фиксированную заранее заданную продолжительность.

Другой вариант постановки задачи, лишенной допущения об одинаковой продолжительности всех действий, был предложен в работе [38]. Эта постановка задачи была названа MAMP (Multi-Agent Motion Planning). В этой постановке задачи у каждого агента есть свой собственный граф $G_i = (V_i,E_i)$. Каждая вершина $v \in V_i$ соответствует состоянию агента i, где состояние может определяться не только координатами в метрическом пространстве, но также, например, ориентацией или скоростью движения агента. Ребро $e = (v,v') \in E_i$ представляет собой действие, которое удовлетворяет кинодинамическим ограничениям агента и позволяет перейти ему из v в v'. При этом пространство, в котором действуют

агенты, описывается с помощью графа регулярной декомпозиции, состоящего из множества вершин $\mathcal{C}$. Каждое состояние $V \in V_i$ агента i ассоциировано с набором вершин из множества $\mathcal{C}$. Таким образом определяется какие вершины блокирует агент, находясь в том или ином состоянии. Каждое ребро $e = (v,v') \in E_i$ также ассоциировано с множеством вершин из $\mathcal{C}$. Для каждой из вершин определен интервал времени, в течении которого агент блокирует эту вершину совершения действие, которому соответствует ребро e. Однако, как и в случае с алгоритмом E-ICTS, предложенный в работе алгоритм опирается на допущение о том, что все действия агентов имеют продолжительность кратную некоторому шагу времени.

2.4 Выводы по главе

Проведенный обзор существующих алгоритмов решения задачи многоагентного планирования и анализ современного состояния области показал, что:

- Существующие алгоритмы многоагентного планирования, гарантирующие нахождение оптимального решения, опираются на допущение о дискретности времени и либо оперируют только действиями одинаковой продолжительности, либо кратными шагу дискретизации.
- Большинство существующих оптимальных алгоритмов оперируют в совместном пространстве состояний, в связи с чем не могут эффективно масштабироваться к задачам, содержащим большое количество агентов. Этого недостатка лишены алгоритмы, использующие подход конфликтно-ориентированного поиска, который планирует траектории агентов независимо, устраняя возможные конфликты с помощью ограничений.

В связи с этим предлагается разработать алгоритм, использующий подход конфликтно-ориентированного поиска, который будет способен осуществлять планирование траекторий с возможностью осуществления действий произвольной продолжительности. Использование подхода конфликтно-ориентированного поиска позволит добиться отыскания гарантированно оптимальных решений, а возможность осуществления действий произвольной продолжительности позволит повысить качество отыскиваемых решений.

Глава 3. Алгоритм конфликтно-ориентированного поиска с действиями произвольной продолжительности

3.1 Подход конфликтно-ориентированного поиска

Предлагаемый в работе алгоритм использует подход конфликтно-ориентированного поиска [90]. Прежде чем описывать особенности разработанного алгоритма, стоит рассмотреть принцип работы этого подхода на примере классической постановки задачи с дискретным временем и возможностью совершения действий только одинаковой продолжительности.

Подход конфликтно-ориентированного поиска является двухуровневым. На верхнем уровне алгоритм оперирует различными частичными решениями, а на нижнем – осуществляет планирование индивидуальных траекторий агентов. Под частичным решением подразумевается совокупность траекторий, удовлетворяющая некоторому набору ограничений, наложенных на агентов, которая при этом может содержать конфликты.

В первую очередь алгоритм создает начальное частичное решение Π_0, состоящее из траекторий агентов, спланированных независимо, т.е. без учета положения других агентов. Для планирования индивидуальных траекторий агентов может использоваться любой алгоритм, гарантирующий нахождение траекторий с минимально возможной стоимостью, например A* [58]. Совокупность траекторий Π_0 обладает минимально возможной стоимостью и по сути является нижней оценкой стоимости решения рассматриваемой задачи:

$$\nexists \Pi' : cost(\Pi') < cost(\Pi_0) \tag{3.1}$$

Если начальное частичное решение не содержит конфликтов, то следуя определению , оно является искомым. Однако, в общем случае начальное частичное решение может содержать конфликты и для их устранения алгоритм накладывает ограничения на агентов. В случае, если это вершинный конфликт, то накладывается ограничение вида $\langle i, v, t \rangle$, которое запрещает агенту i находиться в вершине v в момент времени t. Аналогично, если найден реберный конфликт, то накладывается ограничение вида $\langle i, e, t \rangle$, которое запрещает агенту i совершать

переход по ребру e в момент времени t. Допустим, найден конфликт между агентами i и j, которые пытаются занять вершину v в момент времени t – $\langle i,j,v,t\rangle$. Для устранения этого конфликта алгоритм накладывает ограничение на одного из агентов, которое запрещает ему занимать вершину v в момент времени t. При этом подход конфликтно-ориентированного поиска рассматривает оба альтернативных варианта устранения конфликта, накладывая ограничения на обоих агентов, участвующих в конфликте, и создавая два альтернативных частичных решения. Таким образом, если агент i должен занять вершину v в момент времени t, то он сможет сделать это в том альтернативном частичном решении, где ограничение было наложено на агента j и наоборот, соответственно. При этом, очевидно, не может существовать такого решения, в котором оба агента должны занять вершину v в момент времени t, т.к. это приведет к конфликту.

Планирование индивидуальных траекторий агентов с учетом накладываемых ограничений осуществляется с помощью алгоритма A* [58], у которого в качестве идентификатора состояния используется пара $\langle v,t\rangle$, т.е. одной и той же вершине соответствует множество состояний, различающихся моментами времени. Алгоритм итерационно раскрывает вершины в порядке возрастания значения $f = g + h$, где g – стоимость пути от стартового состояния до текущего, а h – эвристическая оценка стоимости пути до цели. Для осуществления действий ожиданий, при генерации состояний-потомков помимо смежных вершин, состояние генерируется также в текущей раскрываемой вершине с моментом времени $t + 1$. В случае, если генерируемое состояние-потомок нарушает какое-либо из наложенных на агента ограничений, то состояние-потомок не генерируется.

Для возможности оперирования различными частичными решениями на верхнем уровне алгоритма используется так называемое дерево ограничений (от англ. Constraint Tree). Каждая вершина этого дерева N включает в себя три компонента: $N.cons$ – набор ограничений, $N.\Pi$ – частичное решение, т.е. совокупность траекторий, удовлетворяющих всем ограничениям из множества $N.cons$, а также $N.cost$ – стоимость частичного решения, т.е. $cost(N.\Pi)$. На каждом шаге алгоритм выбирает одно из листьев дерева N, обладающее минимальной стоимостью. В случае, если содержащаяся в нем совокупность траекторий не содержит конфликтов – то искомое оптимальное решение найдено. В противном случае выбирается один из конфликтов, который устраняется путем наложения ограничений на агентов, участвующих в этом конфликте, и создания двух новых вершин

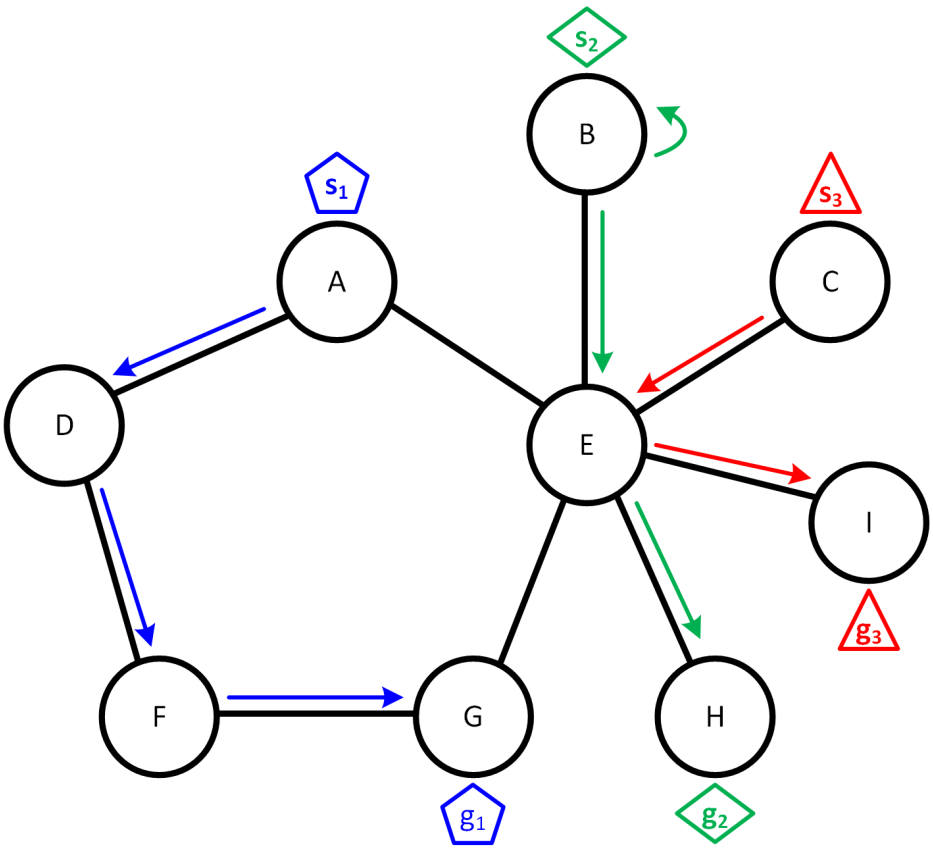


Рисунок 3.1 — Пример задачи многоагентного планирования. s_i – стартовые положения агентов, g_i – целевые. Стрелками обозначено одно из существующих неконфликтных решений этой задачи, обладающее минимально возможной стоимостью.

дерева ограничений. Алгоритм продолжает свою работу до тех пор, пока не будет найдена совокупность траекторий, не содержащая конфликтов.

Рисунок 3.1 показывает пример задачи многоагентного планирования. Имеется граф, состоящий из девяти вершин, а также даны три агента – $green$, $blue$, red. Агент $green$ должен перейти из вершины A, в вершину G, агент $blue$ – из B в I, а агент red – из C в H. На рисунке также показаны траектории агентов, исполнив которые, агенты смогут достичь своих целевых положений не столкнувшись друг с другом. Рассмотрим на этом примере, каким образом подход конфликтно-ориентированного поиска найдет это решение.

В ходе работы основного цикла алгоритм итеративно рассматривает различные частичные решения до тех пор, пока не найдет совокупность траекторий, не содержащую конфликтов. На Рисунке 3.2 показано состояние дерева ограничений на последней итерации работы алгоритма в случае, когда на шаге 4 была выбрана вершина N_4. Далее подробно распишем работу алгоритма на каждой итерации.

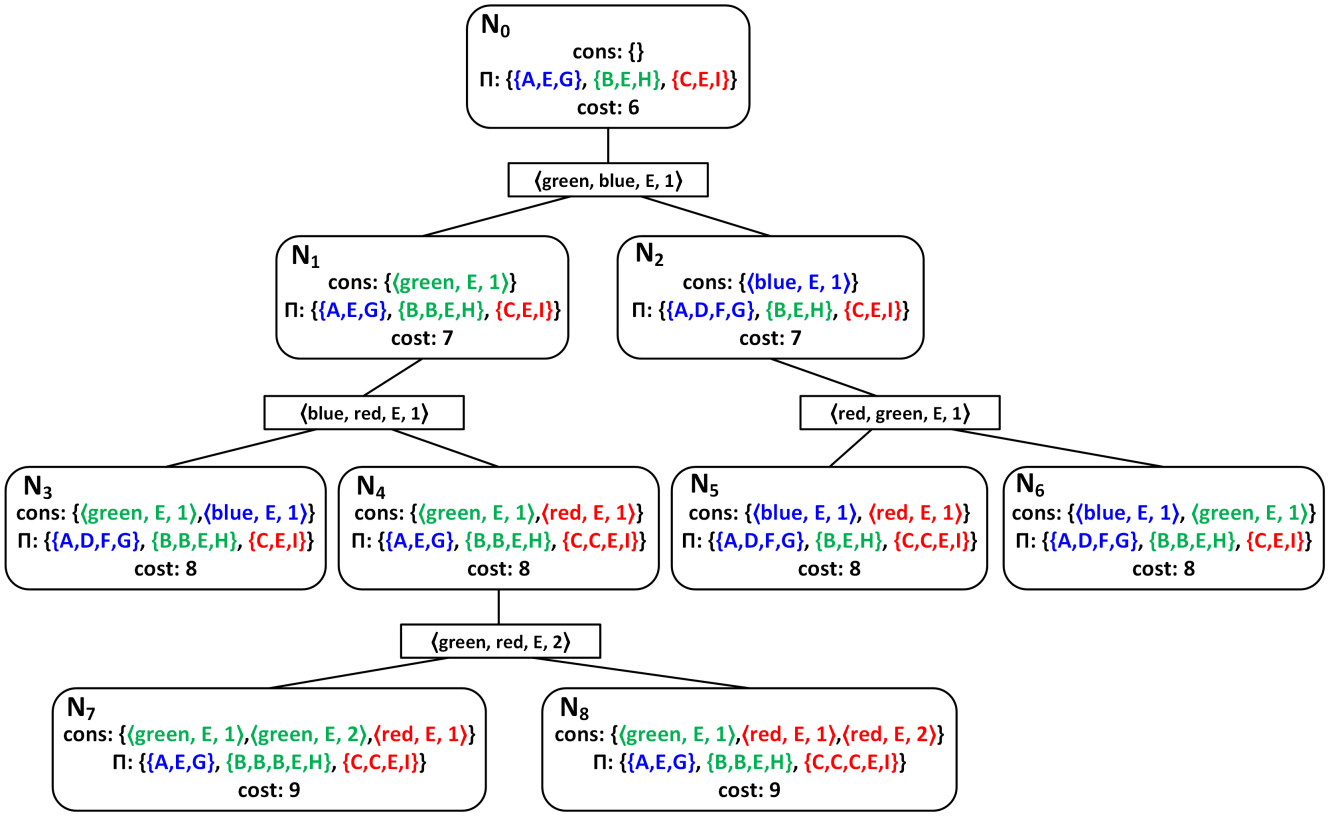


Рисунок 3.2 — Состояние дерева ограничений на последней итерации основного цикла работы алгоритма при решении задачи, показанной на Рисунке 3.1.

На этапе инициализации создается корень дерева ограничений N_0, который содержит начальное частичное решение $N_0.\Pi = \{\{A,E,G\},\{B,E,I\},\{C,E,H\}\}$. Стоимость начального частичного решения: $N_0.cost = 6$. Очевидно, что эта совокупность траекторий имеет конфликты, т.к. все агенты переходят в вершину E в момент времени 1.

На шаге 1 из дерева ограничений извлекается вершина N_0, т.к. она является единственной. Один из конфликтов, содержащихся в $N_0.\Pi$, например, между агентами $green$ и $blue$, выбирается для устранения. В результате создаются две новых вершины дерева ограничений, в которых на одного из агентов, участвующих в конфликте, накладывается ограничение, запрещающее ему находиться в вершине E в момент времени 1. Таким образом, вершина N_1 содержит частичное решение, в котором агент $green$ вынужден совершить действие ожидания для того, чтобы удовлетворить ограничению, а в вершине N_2 агент $blue$ найдет альтернативный путь к своему целевому положению. В обоих случаях стоимость решения выросла до 7. Оба созданных частичных решения устраняют конфликт между агентами $green$ и $blue$, однако, все еще содержат конфликты с агентом red.

На шаге 2 из дерева ограничений извлекается либо вершина N_1, либо N_2. Т.к. они обе обладают одинаковой стоимостью, то возможен выбор любой из этих вершин. Пусть будет выбрана вершина N_1. Совокупность траекторий N_1.П содержит конфликт между агентами $blue$ и red в вершине E в момент времени 1. Аналогичным образом, создается два новых частичных решения, в которых на одного из агентов накладывается ограничение, а его траектория перепланируется с учетом нового ограничения. В результате создаются вершины N_3, N_4, каждая из которых имеет стоимость 8.

На шаге 3 возможен выбор из трех вершин дерева ограничений: N_2, N_3, N_4. Однако, выбрана будет вершина N_2, т.к. она обладает меньшей стоимостью в сравнении с вершинами N_3, N_4. Совокупность траекторий N_2.П содержит конфликт между агентами $blue$ и $green$ в вершине E в момент времени 1. Аналогичным образом, создается два новых альтернативных решения, в которых на одного из агентов накладывается ограничение, а его траектория перепланируется с учетом нового ограничения. В результате создаются вершины N_5, N_6, каждая из которых имеет стоимость 8.

На шаге 4 возможен выбор из четырех вершин дерева ограничений: N_3, N_4, N_5, N_6. Все они обладают одинаковой стоимостью. В случае если алгоритм выберет любую вершину, отличную от N_4, то искомое решение будет найдено, т.к. частичные решения в вершинах N_3, N_5 и N_6 не содержат конфликтов. При этом решения являются оптимальными, т.к. все рассмотренные частичные решения меньшей стоимости содержат в себе по крайней мере один конфликт. В случае выбора вершины N_4 алгоритм совершит дополнительную итерацию и создаст две новых вершины N_7, N_8 со стоимостью 9, в которых будет устранен конфликт, содержащийся в вершине N_4. При этом, следуя заданному принципу выбора вершин из дерева ограничений, алгоритм всегда выберет вершину с минимальной стоимостью, поэтому даже в случае нахождения решения, не содержащего конфликтов со стоимостью 9 или больше, алгоритм сперва извлечет из дерева ограничений одну из оставшихся вершин со стоимостью 8, например N_3.

Алгоритм конфликтно-ориентированного поиска гарантирует, что найденное им решение обладает минимально возможной стоимостью. Алгоритм также гарантирует нахождение решения за конечное число итераций, при условии его существования. Подробное описание свойств этого алгоритма и их доказательства приведены в работе [90].

3.2 Переход к действиям произвольной продолжительности

В отличие от оригинального алгоритма конфликтно-ориентированного поиска, предлагаемый в работе алгоритм способен решать задачу многоагентного планирования в более общей постановке задачи, которая позволяет агентам совершать действия произвольной продолжительности. Разработанный алгоритм, названный Continuous Conflict Based Search (CCBS), обладает следующими ключевыми отличиями:

1. Определение конфликта.
2. Тип накладываемых ограничений.
3. Процесс планирования индивидуальных траекторий с учетом наложенных ограничений.

Далее будут подробно описаны каждый из вышеупомянутых пунктов.

3.2.1 Определение конфликта

Те типы конфликтов, которые рассматриваются в классической постановке задачи, т.е. вершинный и реберный, не способны описать те конфликты, которые могут возникать в рассматриваемой постановке задачи. Рассмотрим пример, представленный на Рисунке 3.3, в котором есть всего два агента. Индивидуальные траектории этих агентов, спланированные независимо, выглядят следующим образом:$\pi_{blue} = \{(A \longrightarrow B, \sqrt{5}), (B \longrightarrow F, \sqrt{11})\}$, $\pi_{green} = \{(D \longrightarrow E, 2.0), (E \longrightarrow C, 2\sqrt{2})\}$. Однако, если оба этих агента начнут одновременно исполнять спланированные траектории, то они столкнутся. При этом нельзя отнести место столкновения к какой-либо одной вершине или ребру графа. В действительности, конфликт происходит между действиями агентов и его наличие зависит от того, в какие моменты времени агенты начнут их совершать. Таким образом, определим конфликт следующим образом:

Определение 4. Набор $\langle i,j,(a_i,t_i),(a_j,t_j)\rangle$ является конфликтом, обозначаемый как $InConflict((a_i,t_i),(a_j,t_j))$, в тех случаях, когда:

$$\exists t \in [t_i,t_i + a_{iD}] \cap [t_j,t_j + a_{jD}] : \ InCollision\Big(i,j,a_{i\varphi}(t - t_i), a_{j\varphi}(t - t_j)\Big) \quad (3.2)$$

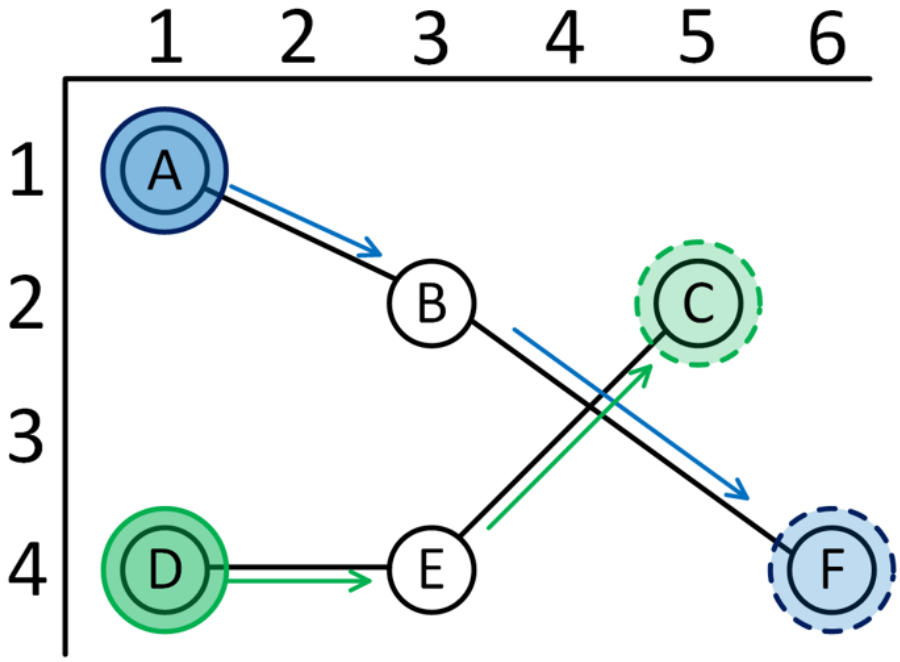


Рисунок 3.3 — Пример задачи с двумя агентами. При одновременном движении между действиями $(B \longrightarrow F, \sqrt{11})$ агента $blue$ и действием $(E \longrightarrow C, 2\sqrt{2})$ агента $green$ произойдет конфликт.

Сложность проверки наличия конфликта между парой действий a_i и a_j зависит от формы агентов i, j, а также от соответствующих функций движения $a_{i\varphi}$,$a_{j\varphi}$. В проведенных модельных экспериментальных исследованиях агенты моделируются дисками радиуса r, а при исполнении действий-перемещений агенты движутся по прямой вдоль ребра графа с постоянной скоростью. В таком случае для определения конфликта возможно использование подхода, описанного в [91], который может за константное время, т.е. за $O(1)$, однозначно определить наличие или отсутствие конфликта между парой действий. В общем случае, когда агенты имеют произвольную форму тел, а также могут двигаться по криволинейным траекториями с изменяющейся скоростью, задача определения конфликта между парой действий является нетривиальной. Для решения такого рода задач могут применяться алгоритмы определения пересечений, применяемые в задачах компьютерной графики [92]. Однако, исследование этих методов не является частью настоящей диссертации.

Стоит отметить, что для наличия конфликта между парами действий, необязательно наличие пересечения ребер, вдоль которых движутся ребра, или инцидентности ребра вершине, в которой один из агентов совершает действие ожидания. Конфликт возможен между непересекающимися и неинцидентными

ребрами в связи с тем, что он происходит с учетом форм и размеров тел агентов. В связи с чем наличие конфликта между агентами проверяется через минимальное возможное расстояние между действиями a_i и a_j. Если оно меньше сумм радиусов агентов ($2r$), то конфликт между такой парой действий возможен. В рассматриваемом примере, как показано на Рисунке 3.3, конфликт происходит между действием $(B \longrightarrow F, \sqrt{11})$ агента $blue$ и действием $(E \longrightarrow C, 2\sqrt{2})$ агента $green$.

Поиск конфликтов

Процедура поиска конфликтов является одной из наиболее затратных операций, которую алгоритмы, использующие подход конфликтно-ориентированного поиска, выполняют на каждой итерации в процессе своей работы. В случае классической постановки задачи с дискретным временем и использования 4х-связного ГРД, процедура поиска конфликтов довольно тривиальна. Для этого используется таблица, в которую записываются положения агентов в каждый дискретный момент времени и если в какой либо из моментов времени два агента попадают в одну и ту же ячейку таблицы, следовательно, между ними происходит вершинный конфликт. Сверяя положения в смежные моменты времени, определяется реберный конфликт.

В рассматриваемой постановке задачи этот подход, очевидно, не может быть применен по причине отсутствия дискретных моментов времени и иного определения самого понятия конфликта. Конфликт в рассматриваемой постановке задачи происходит между действиями, с учетом размеров агентов. По сути, для проверки существования конфликта между парой действий, необходимо найти минимальное расстояние между агентами, которое достигается в процессе исполнения этих действий. Для этого может быть использован подход, предложенный в работе [91]. Этот подход позволяет вычислить минимальное расстояние между парой агентов, имея заданные начальные положения и направления движения:

$$||(a_{i\varphi}(0) + v_i\tau) - (a_{j\varphi}(0) + v_j\tau)|| = r_i + r_j \tag{3.3}$$

, где $a_{k\varphi}(0)$ – начальное положение агента, v_k – вектор скорости движения, r_k – радиус агента, $k = i,j$. Данное выражение может быть преобразовано к квад-

ратичному виду:

$$(x \cdot x)\tau^2 + 2(x \cdot y)\tau + x \cdot y - (r_i + r_j)^2 = 0 \tag{3.4}$$

, где $x = a_{i\varphi}(0) - a_{j\varphi}(0)$, $y = v_i - v_j$. Вычислив корни этого уравнения, можно определить превысит ли сумма радиусов агентов минимальное расстояние между ними и в какой момент времени. Используем символ D для обозначения дискриминанта уравнения, а также τ_1 и τ_2 в качестве корней уравнения. Рассмотрим все возможные варианты решения этого уравнения:

- $D < 0 : \nexists \tau_1, \tau_2$ – действия агентов не имеют конфликта, т.к. расстояние между агентами никогда не становится меньше $2r$. Такая ситуация возможно только в тех случаях, когда либо агенты движутся вдоль параллельных прямых, либо один из агентов совершает действие ожидания и имеет нулевой вектор скорости движения.
- $D = 0 : \tau_1 = \tau_2$ – существует лишь один момент времени, когда расстояние между агентами равно $2r$. Т.е. в процессе исполнения действий агенты сближаются до расстояния $2r$. Этот случай не является конфликтом, т.к. для наличия конфликта расстояние между агентами должно быть строго меньше $2r$.
- $D > 0 : \tau_1 \neq \tau_2$ – существует интервал времени (τ_1, τ_2) в течение которого расстояние между агентами меньше суммы их радиусов. Конфликт происходит в том случае, если этот интервал имеет пересечение с интервалом времени, когда оба агента совершают соответствующие действия: $(\tau_1, \tau_2) \cap [max(t_i, t_j), min(t_i + a_{iD}, t_j + a_{jD}] \neq \varnothing$

Стоит отметить, что для применения этого подхода необходимо, чтобы оба действия начинались одновременно. Поэтому, в тех случаях, когда агенты начинают совершать свои действия в разные моменты времени, то положение агента, начинающего свое действие раньше, смещается до положения, в котором он будет в момент начала совершения действия вторым агентом. Агент i начинает совершать свое действие в момент времени t_i, а агент j – в момент t_j. Допустим, агент i начинает совершать свое действие раньше: $t_i < t_j$. Тогда, для осуществление проверки наличия конфликта между действиями (a_i, t_i) и (a_j, t_j), положение агента i необходимо сместить до $a_{i\varphi}(t_j - t_i)$. Таким образом, за $O(1)$ можно однозначно определить наличие конфликта между парой действий.

Имея процедуру проверки наличия конфликтов между парой действий, можно проверить наличие конфликта между парой траекторий. Для определения

конфликта между парой траекторий π_{blue} и π_{green}, необходимо рассмотреть их в виде последовательности пар действий и моментов времени: $\pi_{blue} = \{(A \longrightarrow B, \sqrt{5}), (B \longrightarrow F, \sqrt{11})\}$, $\pi_{green} = \{(D \longrightarrow E, 2.0), (E \longrightarrow C, 2\sqrt{2})\}$. Сравнивая каждую возможную комбинацию пар действий, необходимо будет проверить $(|\pi_{green}| + |\pi_{blue}|)^2$ пар действий, где $|\pi|$ – количество действий, составляющих траекторию π. Однако, конфликт между агентами возможен только в тех случаях, если соответсвующие действия:

1. пересекаются во времени:

$$t_i \leqslant t_j < t_i + a_{iD} \wedge t_j \leqslant t_i < t_j + a_{jD} \tag{3.5}$$

2. могут пересечься в пространстве:

$$min\_dist(a_i, a_j) < 2r \tag{3.6}$$

Другими словами, если одно действие начинается после того, как другое уже завершилось, то конфликт между такой парой действий невозможен. Конфликт также невозможен, если действия агентов совершаются в разных местах рабочего пространства и поэтому расстояние между агентами в процессе исполнения этих действий не может быть меньше, чем сумма их радиусов. Процедура проверки наличия конфликта между парой траекторий показана в Алгоритме 1.

Для того, чтобы проверять только те пары действий, которые имеют пересечение по времени, процедура использует два счетчика n и m, указывающих на текущее рассматриваемое действие в траектории π_i и π_j соответственно. Изначально, оба счетчика указывают на первые действия в траекториях. Затем, если первое действие агента i заканчивается раньше первого действия агента j, то увеличивается значение счетчика n и проверяется конфликт между вторым действием агента i и первым действием агента j. В противном случае увеличивается значение счетчика m и конфликт проверяется между вторым действием агента j с первым действием агента i. Процесс повторяется и на каждом шаге увеличивается значение счетчика той траектории, чье действие закончилось раньше. Если оба действия закончились одновременно, то увеличивается значение счетчика m. Процесс продолжается до тех пор, пока оба счетчика не достигнут значений эквивалентных количеству действий в траекториях π_iи π_j. Иными словами, процедура работает до тех пор, пока не будут проверены все действия, входящие в траектории π_i и π_j.

Algorithm 1: Процедура проверки пары траекторий на наличие конфликтов

input: π_i, π_j

$n \leftarrow 1$

$m \leftarrow 1$

while $i \leqslant |\pi_1| or j \leqslant |\pi_2|$ **do**

  $a_i, t_i \leftarrow \pi_i(n)$

  $a_j, t_j \leftarrow \pi_j(m)$

  if $checkConflict(a_i, t_i, a_j, t_j)$ **then**

    return $InConflict((a_i, t_i), (a_j, t_j))$

  if $\pi_i[: n+1]_D \geqslant \pi_j[: m+1]_D$ **then**

    $n \leftarrow n + 1$

  else

    $m \leftarrow m + 1$

return "no conflicts"

3.2.2 Интервальные ограничения

Алгоритм CCBS использует подход конфликтно-ориентированного поиска и на каждом шаге выбирает из дерева ограничений один из листьев, обладающих минимальной стоимостью. Обозначим множество всех листьев дерева ограничений как $OPEN$ – список кандидатов-вершин на раскрытие. Аналогичная терминология применяется, например, в классическом алгоритме планирования - A* [58]. Выбрав из списка $OPEN$ решение N минимальной стоимости: $N = argmin_{N.cost} OPEN(N)$, алгоритм осуществляет проверку наличия конфликтов в совокупности траекторий N.П. Если конфликтов не обнаружено, то искомое решение найдено. В противном случае необходимо устранить один из конфликтов, которые есть в рассматриваемом частичном решении. Выбор конфликта производится эвристически и может повлиять на эффективность работы алгоритма. Более подробно о способе ранжирования и выборе конфликта см. раздел 4.1.1. Однако, на сам принцип работы алгоритма и его теоретические свойства, способ выбора конфликта никак не влияет. Будем считать, что алгоритм выбирает первый конфликт, который он смог обнаружить во множестве траекторий N.П.

Пусть найден конфликт $InConflict((a_i,t_i),(a_j,t_j))$. Для его устранения необходимо наложить ограничения на агентов i и j, создав две новых вершины дерева ограничений – N_i и N_j. В отличие от ограничений, которые накладывались на агентов в случае классической постановки задачи, имеющих вид $\langle i, x, t\rangle$, где x – ребро или вершина графа G, в рассматриваемой постановке задачи ограничения предлагается накладывать не на положения, а на действия. Более того, недостаточно запретить агенту совершать действие лишь в один конкретный момент времени t, т.к. он сможет совершить его в момент времени $t + \varepsilon$, что снова приведет к конфликту между той же самой парой действий. Поэтому ограничение предлагается накладывать не на единичные моменты времени, но на интервалы, в течение которых агенты не могут совершать действия a_i и a_j соответственно. Интервал имеет вид $[t, t^u)$, где t^u – это первый момент времени, когда агент может начать совершать соответствующее действие без конфликта с действием другого агента.

Определение 5. Для устранения конфликта $InConflict((a_i,t_i),(a_j,t_j))$ на одного из агентов накладывается ограничение, которое задается набором $\langle k, a_k, [t_k, t_k^u)\rangle$ и запрещает агенту k совершать действие a_k в течение конфликтного интервала $[t_k, t_k^u)$, где t_k^u – первый момент времени, когда агент k может начать совершать действие a_k не создавая конфликта с действием другого агента ($k = i,j$). В случае, если $k = i$, значение t_k^u определено следующим образом:

$$t_i^u = argmin_{t\in[t_i,t_j+a_{jD}]}\{InConflict((a_i,t_i),(a_j,t_j)) = \text{False}\} \tag{3.7}$$

Интервал $[t_i, t_i^u)$ называется *конфликтным*, т.к. если агент i начнет исполнять действие a_i в любой момент времени в течение интервала $[t_i, t_i^u)$, то возникнет конфликт с действием a_j, которое агент j начинает совершать в момент времени t_j. Стоит отметить, что если конфликт происходит с агентом, который уже завершил исполнение своей траектории и находится в своем целевом положении, то проверка $InConflict((a_i,t_i),(a_j,t_j))$ будет всегда выдавать значение $true$. В этом случае значение t_i^u устанавливается равным $+\infty$. Во всех остальных случаях момент времени t_i^u гарантированно существует и является конечным, т.к. в худшем случае конфликт между рассматриваемой парой действий закончится в момент времени $t_j + a_{jD}$, т.е. тогда, когда другой агент закончит выполнять свое действие.

Таким образом для устранения конфликта $InConflict((a_i,t_i),(a_j,t_j))$, содержащегося в совокупности траекторий N.П, алгоритм CCBS создает две новых

вершины дерева ограничений N_i и N_j, в каждую из которых добавляется дополнительное ограничение $\langle i, a_i, [t_i, t_i^u)\rangle$ и $\langle j, a_j, [t_j, t_j^u)\rangle$ соответственно. На примере, изображенном на Рисунке 3.3, начальное частичное решение, содержащееся в корневой вершине дерева ограничений, содержит конфликт $InConflict((B \longrightarrow F, \sqrt{5}), (E \longrightarrow C, 2.0))$, возникающий между агентами $green$ и $blue$. Конфликтный интервал для действия $(B \longrightarrow F)$ агента $green$ равен $[\sqrt{5}, 3.126)$, а для действия $(E \longrightarrow C, 2.0)$ агента $blue$ – $[2.0, 3.467)$.

Расчет интервальных ограничений

Для расчета ограничений, накладываемых на агентов для устранения конфликтов, необходимо вычислять конфликтный интервал, т.е. интервал времени, в течение которого агент не может выполнить соответствующее действие. Процедура расчета интервальных ограничений, как и процедура идентификации конфликтов, зависит от того, какую форму имеют агенты и какой моделью движения они обладают. В рассматриваемой постановке задачи агенты имеют дискообразную форму и движутся с постоянной скоростью. Для рассматриваемого случая может быть использован подход, описанный в [93], который позволяет определить продолжительность конфликтного интервала. Прежде чем описывать этот подход, введем следующие обозначения:

- P_i, P_j – начальные положения агентов i и j соответственно.
- V_i, V_j – направления движений агентов i и j соответственно.
- δ – продолжительность действия ожидания, которое нужно совершить агенту для избегания конфликта.

Для вычисления значения δ необходимо решить следующее уравнение:

$$sqEdgeDist(t, \delta) = At^2 + Bt\delta + C\delta^2 + Dt + E\delta + F, \tag{3.8}$$

где:
$A = (V_i - V_j)^2$
$B = 2(V_i^2 - V_i \cdot V_j)$
$C = V_i^2$
$D = 2(P_j + P_i)(V_j - V_i)$
$E = -2(P_j \cdot V_i + P_i \cdot V_i)$

$F = (V_i - V_j)^2 - (r_i + r_j)^2$

Выражение 3.8 является уравнением конического сечения. В случаях, когда значения A и C являются положительными, это выражение является уравнением эллипса. На Рисунке 3.4(а) показан пример действий агентов, график изменения расстояния между агентами, а также результирующее коническое сечение. Обратите внимание, что горизонтальная линия при $\delta = 0$ пересекает как график расстояния, так и график сечения, в одни и те же моменты времени. Если агенту i добавить задержку, то горизонтальная линия сместится вверх, и наоборот, если добавить задержку агенту j, то линия сместится вниз. Вопрос, который необходимо решить, это размер задержки, который необходимо совершить агенту, чтобы агенты перестали сталкиваться, а минимальное расстояние между ними достигало значения $r_i + r_j$.

Величина требуемой задержки определяется верхним и нижним экстремумами эллипса [94]:

$$delayRange = center_\delta \pm \sqrt{(2BD - 4AE)^2 + 4(4AC - B^2)(D^2 - 4AF)}/2(4AC - B^2), \tag{3.9}$$

где $center_\delta = (BD - 2AE)/(4AC - B^2)$.

Значения концов конфликтных интервалов определяются через выражение:

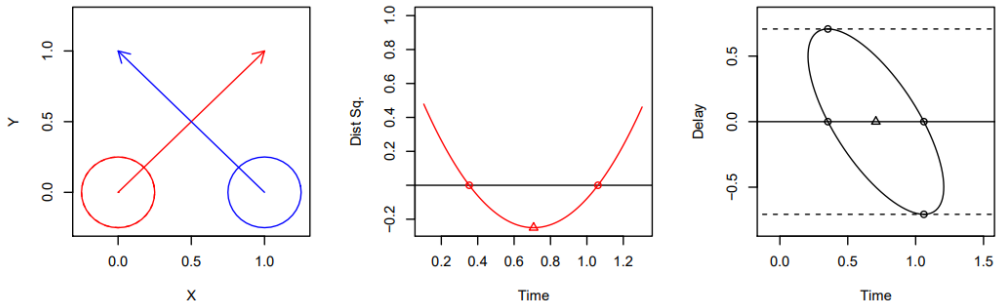
$$collisionTimes = (-B(delayRange) - D)/2A \tag{3.10}$$

Т.к. действия агентов не бесконечны, необходимо также учитывать моменты окончания действий агентов. Когда движения агентов i и j начинаются в t_i и t_j и заканчиваются в t'_i и t'_j соответственно, вычисления конфликтного интервала производятся относительно моментов $t_0 = min(t_i, t_j)$ и $t_{max} = min(t'_i, t'_j)$. В случаях, когда $\delta = t_i - t_j$ выходит за пределы диапазона $delayRange$, рассчитанного по (3.9), конфликта между агентами нет.

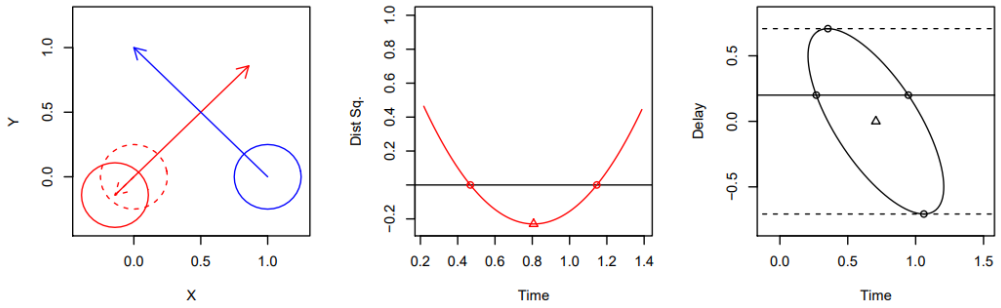
Псевдокод, описывающий процедуру расчета величины требуемой задержки, дан в работе [93].

3.2.3 Планирование индивидуальных траекторий

Для устранения конфликта между агентами недостаточно наложить ограничения. Необходимо спланировать траекторию агента с учетом этих ограничений,



(a)



(b)

Рисунок 3.4 — Пример определения границ конфликтного интервала. Иллюстрация взята из [93]

т.е. найти траекторию, которая бы удовлетворяла всем наложенным на агента ограничениям. При этом траектория должна быть оптимальной, т.е. обладать наименьшей возможной стоимостью. В случае классической постановки задачи, где используется допущение о дискретности времени, возможно применение классического алгоритма A*. Как отмечалось ранее, для возможности осуществления действия ожидания, состояние-потомок генерируется и в текущей раскрываемой вершине v с моментом времени $t + 1$. Однако, подобный подход обладает рядом недостатков. Во-первых, генерация отдельного состояния для каждого момента времени снижает эффективность работы алгоритма. Во-вторых, количество моментов времени даже в дискретном случае является бесконечным. Следовательно, без каких-либо дополнительных модификаций, алгоритм не сможет корректно завершить свою работу в случаях, когда пути между стартовой и целевой вершиной не существует. Возможным решением является введение дополнительного правила, которое меняет идентификатор состояния на вершину v без использования момента времени t в тех случаях, когда генерируемое состояние имеет момент времени t больший чем последний момент времени из всех наложенных на агента ограничений. Фактически происходит переключение режима работы алгоритма

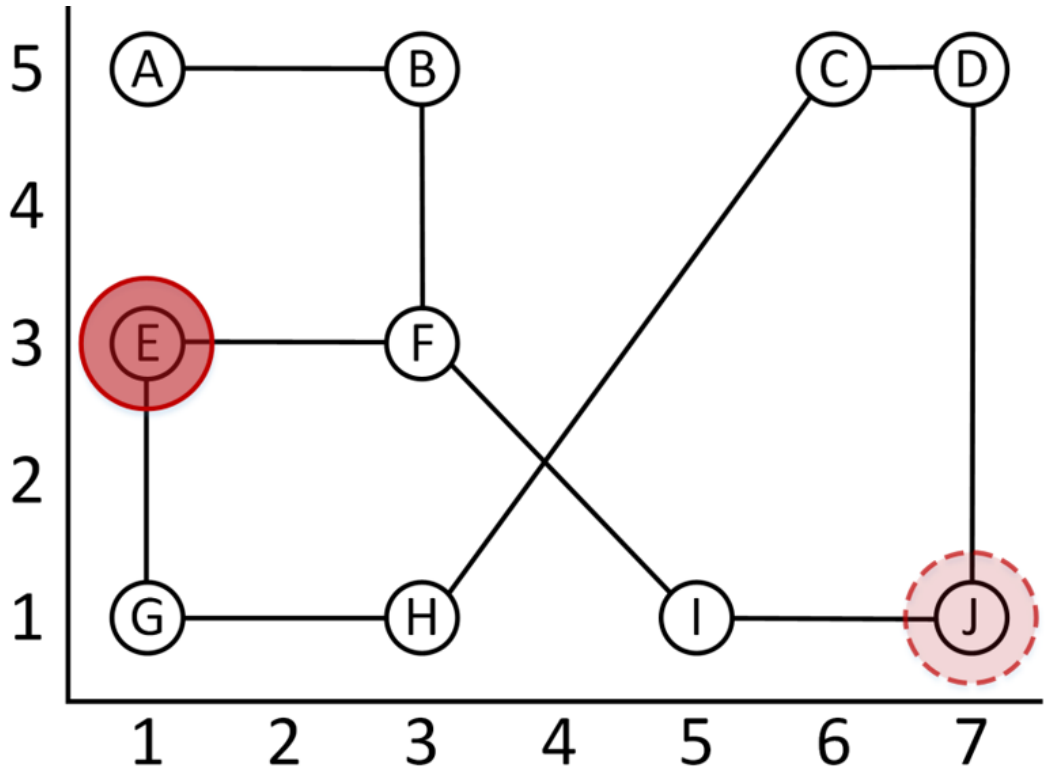


Рисунок 3.5 — Пример задания индивидуального планирования. Агент должен достичь вершину J из вершины E. При этом на него наложены два интервальных ограничения: $\langle red, F \longrightarrow I, [2.0, 3.74)\rangle$, $\langle red, F \longrightarrow F, [3.0, 3.5)\rangle$.

на планирование в статичной среде. Введение такого правила объясняется тем, что совершение действия ожидания, а также повторное посещение одной и той же вершины графа лишены смысла в случае статичной среды. В-третьих, генерация состояния-потомка в той же самой вершине с моментом времени $t + 1$ делает этот подход неприменимым в рассматриваемой постановке задачи, где действия ожидания могут иметь произвольную продолжительность.

Рассмотрим пример задания на Рисунке 3.5. В этом примере агенту требуется достичь вершину J из вершины E. При этом на агента были наложены два ограничения: $\langle red, F \longrightarrow I, [2.0, 3.74)\rangle$, $\langle red, F \longrightarrow F, [3.0, 4.0)\rangle$. Первое ограничение наложено на действие перемещения из вершины F в вершину I в течение интервала $[2.0, 3.74)$, в то время как второй интервал запрещает агенту находиться в вершине F в течение интервала $[3.0, 3.5)$. Если следовать логике стандартного алгоритма A*, где в качестве идентификатора состояния используется только вершина, то на первом шаге алгоритма, раскрывая вершину E, будут сгенерированы состояния-потомки в вершинах G и F соответственно. На следующем шаге, при попытке генерации потомков состояния в вершине F, алгоритм не сможет сгенерировать потомка в вершине I, т.к момент начала совершения действия ($t = 2.0$)

входит в конфликтный интервал. В теории, имея информацию о значении конца конфликтного интервала, можно сместить момент начала совершения действия на конец конфликтного интервала. Однако, совершить действие перемещения из F в I в момент времени 3.74 невозможно, т.к. агенту запрещено ждать в вершине F в интервале $[3.0, 3.5)$. Т.к. никакое дополнительное состояние в вершине E не генерировалось (с действием ожидания), оптимальный путь в этой задаче оригинальным алгоритмом A* фактически не может быть найден. При использовании алгоритма A*, который оперирует в пространстве состояний состоящих из пар вершин и моментов времени, оптимальный путь до вершины J также не может быть найден, т.к. алгоритм не может сгенерировать все возможные действия ожидания. Генерируя действия ожидания с шагом 1, оптимальный путь будет утрачен.

Для решением этой проблемы и возможности планирования индивидуальных траекторий агентов с учетом интервальных ограничений предлагается использование подхода безопасно-интервального планирования (англ. Safe Interval Path Planning, SIPP) [95]. Оригинальный алгоритм SIPP гарантирует нахождение пути минимальной стоимости при условии его существования и был предложен для планирования в средах с динамическими препятствиями. В рассматриваемом случае динамические препятствия отсутствуют, однако присутствуют ограничения, накладываемые верхним уровнем алгоритма CCBS, которые необходимо учитывать. Модификацию алгоритма, которая учитывает интервальные ограничения, накладываемые алгоритмом CCBS, будем называть CSIPP (англ. Constrained Safe Interval Path Planning).

Принцип работы алгоритма SIPP схож с принципом работы алгоритма A*. Алгоритм оперирует двумя списками – $OPEN$ и $CLOSED$. Список $OPEN$ содержит в себе все состояния-кандидаты на раскрытие, а список $CLOSED$ – все уже раскрытые состояния. Изначально список $OPEN$ содержит состояние, соответствующее стартовому положению, а список $CLOSED$ пуст. На каждом шаге алгоритм выбирает из списка $OPEN$ вершину с минимальным f-значением, $f = g + h$, где g – стоимость пути от стартового состояния до текущего, а h – эвристическая оценка стоимости пути от текущего состояния до целевого. Алгоритм итеративно раскрывает состояния, рассматривая состояния в смежных вершинах, до тех пор, пока не будет раскрыто состояние, соответствующее целевому положению. После того, как алгоритм раскрыл состояние, он добавляет его в список $CLOSED$. Соответственно список $CLOSED$ содержит все раскрытые состояния. Более того, если используемая эвристическая функция h обладает

свойством монотонности, то можно утверждать, что все вершины, находящиеся в списке $CLOSED$, обладают минимально возможным g-значением, т.е. до них найден кратчайший путь. Более подробно о свойствах алгоритма см. раздел 29. Свойство монотонности определяется через неравенство треугольника: $\forall v,v,',v'' : h(v,v') \leqslant h(v,v'') + h(v'',v')$, т.е. значение эвристической функции между двумя вершинами не может быть больше чем сумма значений эвристической функции, посчитанных через какую-либо дополнительную промежуточную вершину.

Ключевой особенностью подхода безопасно-интервального планирования является принцип описания состояний. Каждое состояние s определяется парой $\langle cfg, interval \rangle$, где cfg – конфигурация, определяющая положение агента в пространстве, а $interval$ – безопасный интервал, т.е. промежуток времени, в течение которого агент может находиться в соответствующем положении без конфликтов с динамическими препятствиями. В рассматриваемой задаче не учитывается скорость движения или ориентация агента, поэтому конфигурацию определяет только координаты вершины графа, а безопасные интервалы для конфигураций зависят от того, какие ограничения на действия ожидания были наложены верхним уровнем алгоритма CCBS. Пусть даны два ограничения $\langle i, a_1, [t_1, t_1^u) \rangle$ и $\langle i, a_2, [t_2, t_2^u) \rangle$ ($t_1^u < t_2$), запрещающие агенту i ждать в вершине v в течение соответствующих интервалов. Тогда для вершины v существуют следующие три безопасных интервала: $[0; t_1]$, $[t_1^u; t_2]$, $[t_u^2, +\infty)$. В свою очередь ограничения, наложенные на действия-перемещения, учитываются при расчете времени достижения смежных состояний, т.е. g-значений. Пусть дано ограничение $\langle i, a_3, [t_3, t_3^u) \rangle$, запрещающее агенту i начинать совершать действие перемещения из вершины v в v' в течение интервала $[t_3, t_3^u)$. Если агент i достигает вершину v в момент времени $t : t_3 \leqslant t < t_3^u$, то начать совершать действие перемещения к вершине v' он может только в момент t_3^u. Для этого перед действием a_3 агент i совершает действие ожидания $a_{wait} : \langle a_{wait\,D} = t_3^u - t, a_{wait\,\varphi} = v \rangle$.

Рассмотрим принцип работы алгоритма CSIPP на том же примере, представленном на Рисунке 3.5. В каждой вершине в графе, за исключением вершины F, есть состояние с безопасным интервалом $[0, +\infty)$, т.е. агент может находиться в нем неограниченное количество времени. Вершине F соответствуют два состояния – $\langle F, [0, 3] \rangle$ и $\langle F, [3.5, +\infty] \rangle$. Возникновение двух безопасных интервалов в вершине F обусловлено наличием ограничения, наложенного на действия-ожидания в вершине F в интервале $[3, 3.5)$.

На первом шаге работы алгоритма будет раскрыта стартовая вершина E и будут рассмотрены все состояния соседних вершин – $\langle G, [0, +\infty)\rangle$, $\langle F, [0, 3]\rangle$, $\langle F, [3.5, +\infty)\rangle$. Первые два могут быть достигнуты без необходимости совершения дополнительных действий ожиданий, в то время как для достижения третьего состояния алгоритму необходимо предварительно совершить действие-ожидание в вершине E продолжительностью 1.5. Продолжительность действия ожидания определяется через разницу между g-значением текущего состояния, а также моментом начала безопасного интервала состояния в смежной вершине: $a_{iD} = s.interval.begin - g(E)$, где a_i – действие ожидания, $s.interval.begin$ – момент начала безопасного интервала состояния s. При этом также осуществляется проверка на возможность совершения данного действия ожидания. Если оно приводит к выходу за пределы безопасного интервала раскрываемого состояния, то соответствующее состояние в смежной вершине, для достижения которого необходимо совершить это действие-ожидание, достичь невозможно (по крайней мере из текущего раскрываемого состояния). Подобная ситуация возникает на следующем шаге алгоритма, когда раскрываемым состоянием становится $\langle F, [0,3]\rangle$. Смежные вершины содержат состояния $\langle E, [0, +\infty)\rangle$, $\langle B, [0, +\infty)\rangle$, $\langle I, [0, +\infty)\rangle$. Состояние в вершине E не рассматривается, так как оно уже было раскрыто и имеет меньшее g-значение чем текущее раскрываемое состояние. Состояние в вершине B может быть достигнуто в момент времени 4 без совершения каких-либо дополнительных действий ожиданий. При расчете g-значения для состояния в вершине I, алгоритм учитывает ограничение, наложенное на действие $F \longrightarrow I$. Начать двигаться к вершине I в момент времени 2 невозможно, т.к. этот момент времени входит в конфликтный интервал ограничения. Следовательно, первым моментом времени, в который агент может начать совершать действие $F \longrightarrow I$ соответствует моменту окончания конфликтного интервала, т.е. 3.74. Для того чтобы начать совершать действие-перемещение из вершины F в вершину I, агент должен предварительно совершить действие ожидания продолжительностью 1.74, однако, совершить его невозможно, т.к. безопасный интервал текущего раскрываемого состояния заканчивается в момент времени 3.0. Следовательно, состояние $\langle I, [0, +\infty)\rangle$ не может быть достигнуто из состояния $\langle F, [0, 3.0]\rangle$ ввиду наложенных ограничений. Однако, оно может быть достигнуто из состояния $\langle F, [3.5, +\infty]\rangle$, которое агент достигает в момент времени 3.5. Совершив действие ожидания продолжительностью 0.24, агент может начать совершать действие перемещение $F \longrightarrow I$ в момент времени 3.74, не нарушая какие-либо ограничения.

В последующем, на одной из итераций будет раскрыто состояние $\langle I, [0, +\infty)\rangle$, которое является смежным с целевым состоянием $\langle J, [0, +\infty)\rangle$. Алгоритм завершит свою работу на том шаге, когда из списка OPEN для раскрытия будет извлечено состояние $\langle J, [0, +\infty)\rangle$. Итоговая траектория выглядит следующим образом: $\pi = \{\langle E \longrightarrow E, 0\rangle, \langle E \longrightarrow F, 1.5\rangle, \langle F \longrightarrow F, 3.5\rangle, \langle F \longrightarrow I, 3.74\rangle, \langle I \longrightarrow J, 6.57\rangle\}$.

Псевдокод алгоритма планирования индивидуальных траекторий

Алгоритм 2 демонстрирует основную логику работы алгоритма CSIPP. Первые 11 строк описывают процедуру инициализации пространства состояний, в котором алгоритм CSIPP будет осуществлять планирование. Изначально множество состояний пусто (строка 1). Для каждой вершины графа создается состояние с интервалом $[0, +\infty)$ и g-значением равным бесконечности, т.к. на этапе инициализации они неизвестны (строки 2-5). Затем происходит обработка ограничений, наложенных на агента. Все ограничения, наложенные на действия ожидания преобразуются в конфликтные интервалы и приводят к созданию новых состояний в соответствующих вершинах графа (строки 6-9). Список OPEN изначально содержит в себе состояние, соответствующее стартовому положению (строка 10). Функция $GetFirstSafeInterval$ возвращает первый безопасный интервал для соответствующей вершины графа, т.е. тот, что начинается в момент времени 0. Список $CLOSED$ изначально пуст.

После того как пространство состояний, а также списки $OPEN$ и $CLOSED$ были инициализированы, начинается основной цикл работы алгоритма (строки 11-27). Условие цикла в строке 11 проверяет, что список $OPEN$ не пуст, т.к. в противном случае алгоритм не может продолжить работу по причине отсутствия следующего кандидата на раскрытие. Такая ситуация возможна лишь в том случае, если целевое состояние является недостижимым из стартового состояния, т.е. построить траекторию между ними невозможно. Более подробно о свойствах алгоритма CSIPP будет рассказано далее в разделе 29.

Каждая итерация алгоритма начинается с выбора текущего состояния s из списка $OPEN$, имеющего минимальное f-значение (строка 12). Состояние s добавляется в список $CLOSED$ (строка 13) и если оно соответствует целевому состоянию, то искомый путь найден (строки 14-15). Елси же состояние s не явля-

ется целевым, то происходит процедура его раскрытия. Сперва осуществляется поиск всех смежных вершин в графе G (строка 16). Затем происходит перебор вершин-соседей (строки 19-27). Для каждой вершины n из N в множестве состояний $States$ осуществляется поиск всех состояний, соответствующих вершине n (строка 18). Для каждого состояния s' производится проверка его наличия в списке $CLOSED$ (строка 20). Если состояние находится в этом списке, то кратчайший путь до этого состояния уже найден, следовательно, его можно не рассматривать. Для всех остальных состояний производится расчет значения EAT (англ. Earliest Arival Time) с учетом ограничений $Cons$ (строка 22). Если состояние s' является достижимым, то функция $GetEAT$ возвращает некоторое конечное значение. Если это значение меньше чем $g(s')$, то значение $g(s')$ обновляется, на его основе рассчитывается новое f-значение, сохраняется информация о состоянии, через которое это g-значение было получено. Если состояние было достигнуто впервые, то оно добавляется в список $OPEN$. Если же оно уже было ранее достигнуто, но на текущей итерации его g-значение было уменьшено, значит состояние s' содержится в списке $OPEN$ и его необходимо обновить (строка 27). В случае, если состояние s' недостижимо из текущего раскрываемого состояния s, то функция $GetEAT$ возвращает значение $+\infty$. Пример ситуации, в которой такое может произойти показан на Рисунке 3.5, в котором состояние $\langle I, [0, +\infty)\rangle$ не может быть достигнуто из состояния $\langle F, [0{,}3]\rangle$ по причине ограничения, наложенного на действие перемещения $F \longrightarrow I$.

Свойства алгоритма планирования индивидуальных траекторий

Алгоритм CSIPP обладает те ми же свойствами, что и оригинальный алгоритм SIPP. Во-первых, алгоритм гарантирует, что найденное решение обладает минимально возможной стоимостью. Во-вторых, алгоритм гарантирует, что найдет решение если оно существует, а также корректно завершит свою работу в случаях, когда задача не имеет решения. Доказательства этих утверждений для алгоритма SIPP приведены в [95]. Далее будут приведены доказательства наличия этих свойств у алгоритма CSIPP, аналогичные тем, что были приведены в [95] для оригинального алгоритма.

Algorithm 2: Псевдокод алгоритма CSIPP

input: $G = (V,E)$, Start, Goal, Cons

$States \leftarrow \varnothing$
foreach *vertex v in* V **do**
  $s \leftarrow \langle v, [0, +\infty) \rangle$
  $g(s') \leftarrow +\infty$
  $States \leftarrow States \cup \{s\}$
foreach *constraint c in Cons* **do**
  if *c is wait constraint* **then**
    foreach *s in S where* $s.cfg = c.a_{\varphi}(0)$ **do**
      merge $s.interval$ with $c.interval$
$OPEN \leftarrow \{\langle Start, GetFirstSafeInterval(Start) \rangle\}$
$CLOSED \leftarrow \varnothing$
while *OPEN* $\neq \varnothing$ **do**
  $s \leftarrow$ pop s from OPEN with minimal f-value
  $CLOSED \leftarrow CLOSED \cup \{s\}$
  if *s is goal* **then**
    return ReconstructPath(s)
  $N \leftarrow GetNeighbors(s.cfg, G)$
  foreach $n \in N$ **do**
    $S \leftarrow$ states from $States$ corresponding to vertex n
    foreach $s' \in S$ **do**
      if $s' \in CLOSED$ **then**
        continue
      $EAT \leftarrow GetEAT(s, s', Cons)$
      if $EAT < g(s')$ **then**
        $g(s') \leftarrow EAT$
        $f(s') \leftarrow g(s') + h(s', Goal)$
        $parent(s') \leftarrow s$
        insert or update s' into OPEN
return “path not found”

Утверждение 1. Алгоритм CSIPP гарантирует нахождение решения при условии его существования, а также гарантирует корректное завершение работы в случаях отсутствия решения.

Доказательство. Доказательство этого утверждения опирается на принцип работы самого алгоритма. Пусть дано некоторое состояние $s = \langle cfg, interval \rangle$. Это состояние может быть достигнуто в различные моменты времени в течение безопасного интервала состояния s. Пусть даны два момента времени t_0, t_1, такие что: $t_0 \in interval$, $t_1 \in interval$, $t_0 < t_1$. Любое действие, которое можно совершить из состояния s, достигнутое в момент времени t_1, может быть совершено из состояния s, достигнутого в момент времени t_0, т.к., достигнув состояние s в момент времени t_0, агент может совершить действие ожидания до момента t_1. При этом агент всегда может совершить это действие-ожидание, т.к. оба момента времени принадлежат одному и тому же безопасному интервалу. Таким образом, достигая состояние в минимально возможное время, на этапе раскрытия алгоритм генерирует все возможные состояния-потомки, которые могут быть достигнуты из текущего раскрываемого состояния. Поэтому, имея всего один момент времени на каждый безопасный интервал, алгоритм не теряет какие-либо состояния-потомки. Таким образом, алгоритм гарантирует нахождение решения при условии его существования.

Корректность завершения работы алгоритма опирается на следующие два факта. Во-первых, согласно Алгоритму 2, каждое состояние может быть раскрыто не более одного раза. После извлечения состояния из списка $OPEN$ оно добавляется в список $CLOSED$ (строка 13 Алгоритма 2) и больше не может быть добавлено в список $OPEN$ для повторного раскрытия (строки 20-21 Алгоритма 2). Во-вторых, число состояний конечно. Изначально, на этапе инициализации (строки 2-5 Алгоритма 2), для каждой вершины $v \in V$ создается одно состояние с безопасным интервалом $[0, +\infty)$. Затем каждое ограничение из множества $Cons$, наложенное на действие ожидания, может разделить один безопасный интервал на два подинтервала и привести к появлению двух состояний вместо одного. Следовательно, общее количество состояний не превышает значения $|V| + |Cons|$. Таким образом, имея конечное число состояний, каждое из которых может быть раскрыто не более одного раза, алгоритм за конечное число итераций завершит свою работу даже в тех случаях, когда решение задачи не существует.■

Утверждение 2. Алгоритм CSIPP гарантирует, что найденное решение обладает минимально возможной стоимостью.

Доказательство. Доказательство этого утверждения, по сути, является следствием доказательства Утверждения 1. Стоимость траектории эквивалента моменту времени достижения состояния, соответствующего целевому положению, у которого безопасный интервал имеет бесконечную продолжительность. Благодаря тому, что алгоритм достигает любое состояние в минимально возможный момент времени, достигая целевое состояние, алгоритм находит траекторию минимально возможной стоимости.■

3.2.4 Псевдокод алгоритма конфликтно-ориентированного поиска с действиями произвольной продолжительности

Комбинируя подход конфликтно-ориентированного поиска с введенным определением конфликта, интервальными ограничениями, а также алгоритмом планирования индивидуальных траекторий, учитывающим интервальные ограничения, был получен алгоритм, способный решать задачу многогагентного планирования с учетом возможности совершения действий произвольной продолжительности. Предложенный алгоритм был назван Continuous Conflict Based Search или же CCBS. Далее представлен псевдокод этого алгоритма.

Псевдокод, представленный в Алгоритме 3, описывает основную логику работы алгоритма CCBS. Прежде чем начинается основной цикл работы алгоритма, происходит инициализация. Для этого осуществляется независимое планирование траекторий всех агентов (строки 1-2). В данном случае может использоваться стандартный алгоритм A*, т.к. при создании начального частичного решения нет ограничений, которые нужно было бы учитывать. Затем создается вершина - корень дерева ограничений, которое не содержит ограничений (строка 3) и добавляется в список $OPEN$ (строка 4).

На каждом шаге основного цикла сперва из списка $OPEN$ выбирается вершина с минимальной стоимостью (строка 6). Если совокупность траекторий N.П не содержит конфликтов, то искомое решение найдено (строки 7-8). Если же N.П содержит конфликты, то происходит их поиск и выбор одного из них (строка 9). Затем, для каждого из агентов, участвующих в конфликте, рассчитывается кон-

Algorithm 3: Псевдокод алгоритма CCBS

input: $\mathcal{G} = (V,E)$, Starts, Goals

1 **foreach** *agent* i **do**
2 $\pi_i \leftarrow$A*$\big(\mathcal{G},Starts(i),Goals(i)\big)$
3 $N \leftarrow \Big(\varnothing,(\pi_1,\ldots,\pi_k)\Big)$
4 Create OPEN; Add N to OPEN
5 **while** *OPEN is not empty* **do**
6 $N \leftarrow$ pop N from OPEN such that $cost(N.\Pi) = \min\limits_{N' \in OPEN} cost(N'.\Pi)$
7 **if** $N.\Pi$ *has no conflicts* **then**
8 **return** $N.\Pi$
9 $\langle i,j,(a_i,t_i),(a_j,t_j)\rangle \leftarrow$ FindConflict($N.\Pi$)
10 **for** $l \in \{i,j\}$ **do**
11 $[t_l,t_l^u) \leftarrow$ compute unsafe interval for agent l
12 $const \leftarrow N.const \cup \{\langle l,a_l,[t_l,t_l^u)\rangle\}$
13 $\pi_l' \leftarrow$ CSIPP$\big(\mathcal{G},Starts(l),Goals(l),const\big)$
14 $\Pi' \leftarrow (N.\Pi \setminus \{N.\pi_l\}) \cup \{\pi_l'\}$
15 $N_l \leftarrow (const,\Pi')$
16 Add N_l to OPEN

фликтный интервал (строка 11) и добавляется новое ограничение (строка 12). После чего осуществляется планирование индивидуальной траектории агента с учетом всех наложенных на него ограничений (строка 13), частичное решение модифицируется с учетом новой траектории (строка 14) и создается новая вершина (строка 15), которая добавляется в список $OPEN$ (строка 16).

Как видно из псевдокода алгоритма, CCBS вызывает процедуру планирования индивидуальных траекторий на каждой итерации своей работы для каждого из агентов, участвующих в выбранном конфликте, т.е. дважды за итерацию. При этом планирование для одного и того же агента может производиться большое количество раз. Разница между разными вызовами заключается лишь в наборе ограничений, наложенных на агента. В связи с этим, для повышения эффективности работы алгоритма CSIPP, в качестве значений эвристической функции используются стоимости путей, предварительно посчитанные без каких-либо ограничений, т.е. в статичной среде.

Следуя логике псевдокода алгоритма, в строке 9 осуществляется поиск и выбор конфликта. Однако, для нахождения конфликтов между траекториями всех агентов необходимо проверить $K(K-1)/2$ пар траекторий. Проводить эту проверку на каждой итерации работы алгоритма вычислительно затратно. Более того, такая проверка является избыточной т.к. каждое новое частичное решение отличается от предыдущего модификацией траектории лишь одного агента. В связи с этим, предлагается использовать подход, когда информация о конфликтах хранится в вершинах дерева ограничений вместе с ограничениями, совокупностью траекторий и её стоимостью. Проверка наличия конфликтов между всеми парами агентов производится только для корневой вершины дерева, а в процессе работы основного цикла алгоритма проверка на наличие конфликтов осуществляется только между модифицированной траекторией и всеми остальными. В случае, когда частичное решение содержит несколько конфликтов, алгоритм использует эвристическое правило, выбирая тот конфликт, который происходит по времени раньше остальных. Аналогичный подход использовался в оригинальном алгоритме конфликтно-ориентированного поиска. Подробнее о способах выбора конфликта см. раздел 4.1.1.

3.3 Теоретические свойства

Разработанный алгоритм CCBS обладает свойством оптимальности, т.е. гарантирует, что найденное им решение имеет минимально возможную стоимость. Прежде чем доказывать свойства алгоритма CCBS, введем ряд обозначений. Пусть $\boldsymbol{P}$ – множество задач многоагентного планирования, $\boldsymbol{P}^+$ – подмножество всех задач многоагентного планирования, имеющих решение, а $\boldsymbol{P}^-$ – подмножество всех задач многоагентного планирования, не имеющих решения.

Для любой задачи p из множества $\boldsymbol{P}^+$ существует множество решений $\mathbf{\Pi}(p)$. Множество решений также делится на два подмножества: $\mathbf{\Pi}(p) = \mathbf{\Pi}_{opt}(p) \cup \mathbf{\Pi}_{subopt}(p)$, где $\mathbf{\Pi}_{opt}(p)$ – подмножество всех оптимальных решений, а $\mathbf{\Pi}_{subopt}(p)$ – подмножество всех субоптимальных решений:

$$\begin{gathered} \forall \Pi \in \mathbf{\Pi}_{opt}(p), \forall \Pi' \in \mathbf{\Pi}(p) : cost(\Pi) \leqslant cost(\Pi') \\ \forall \Pi \in \mathbf{\Pi}_{subopt}(p), \forall \Pi' \in \mathbf{\Pi}_{opt}(p) : cost(\Pi) > cost(\Pi') \end{gathered} \tag{3.11}$$

Определение 6. Пусть дано множество оптимальных решений $\mathbf{\Pi}_{opt}(p)$ некоторой решаемой задачи p, а также пара интервальных ограничений $\langle i,a_i,[t_i,t_i^u)\rangle$ и $\langle j,a_j,[t_j,t_j^u)\rangle$. Будем называть пару ограничений *согласованной*, если любое оптимальное решение удовлетворяет по крайней мере одному из ограничений:

$$\forall \Pi \in \mathbf{\Pi}_{opt}(p), \forall \pi_i, \pi_j \in \Pi : (a_i, t_i') \notin \pi_i, \forall t_i' \in [t_i, t_i^u) \vee (a_j, t_j') \notin \pi_j, \forall t_j' \in [t_j, t_j^u) \tag{3.12}$$

Иными словами, если пара ограничений является согласованной, то ни одно решение из множества $\mathbf{\Pi}_{opt}(p)$ не может включать в себя оба действия, совершаемые в моменты времени, принадлежащие конфликтным интервалам, на которые была наложена *согласованная* пара ограничений.

Лемма 1. *Для произвольного конфликта $InConflict((a_i, t_i), (a_j, t_j))$, ограничения $\langle i,a_i,[t_i,t_i^u)\rangle$ и $\langle j,a_j,[t_j,t_j^u)\rangle$, заданные в соответствии с Определением 5, являются согласованной парой ограничений.*

Доказательство. Пусть дано некоторое частичное решение П, которое содержит конфликт $InConflict((a_i,t_i),(a_j,t_j))$. Для устранения этого конфликта необходимо наложить ограничение на одного из агентов, участвующих в конфликте, – либо $\langle i,a_i,[t_i,t_i^u)\rangle$, либо $\langle j,a_j,[t_j,t_j^u)\rangle$. Допустим, что данная пара ограничений не является согласованной, т.е. существует такое неконфликтное решение, в котором агент i совершает действие a_i в некоторый момент t_i' в течение интервала $[t_i,t_i^u)$, а агент j совершает действие a_j в некоторый момент t_j' в течение интервала $[t_j,t_j^u)$ и при этом эти действия не приводят к возникновению конфликта. Известно, что если агент i начнет совершать действие a_i в момент t_i, а агент j начнет совершать действие a_j в момент t_j, то это приведет к возникновению конфликта. Следовательно, синхронно смещая моменты начала совершения обоих действий, конфликт между действиями сохранится: $InConflict((a_i,t_i+\delta),(a_j,t_j+\delta)), \forall \delta \in \mathbf{R}$. Однако, смещение момента начала совершения действия для агентов i и j может быть различным. Введем обозначения $\delta_i = t_i' - t_i$ и $\delta_j = t_j' - t_j$. Сместим моменты начала совершения действий на величину $min(\delta_i,\delta_j)$. Рассмотрим ситуацию, когда $\delta_j \leqslant \delta_i$ и сместим моменты t_i' и t_j' на величину $-\delta_j$. В таком случае агент j начнет совершать действие a_j в момент времени t_j, а агент i – в момент $t_i' - \delta_j$. При этом момент начала совершения действия агентом i принадлежит интервалу $[t_i,t_i^u)$, т.к., во-первых, рассматривается ситуация в которой $t_i' \in [t_i,t_i^u)$, во-вторых, $t_i' = t_i + \delta_i$ и $\delta_j \leqslant \delta_i$, следовательно $t_i + \delta_i - \delta_j \geqslant t_i$.

Изначально сделанное допущение гласит, что между действиями (a_i,t'_i) и (a_j,t'_j) нет конфликта, следовательно, между действиями $(a_i, t'_i - \delta_j)$ и (a_j,t_j) конфликта также нет. Однако, это утверждение вызывает противоречие с Определением 5, которое гласит, что t_i^u – первый момент времени, когда агент i может совершить действие a_i без конфликта с действием a_j, которое агент j начинает совершать в момент времени t_j. Полученное противоречие доказывает, что пары ограничений, заданные в соответствии с Определением 5, являются *согласованными*.■

Пусть дана произвольная задача многагентного планирования p, имеющая решение, $p \in \boldsymbol{P}^+$, а также некоторое решение этой задачи, найденное с помощью алгоритма CCBS, обозначаемое как Π_{CCBS}.

Теорема 1. $\forall p \in \boldsymbol{P}^+ : \Pi_{CCBS} \in \mathbf{\Pi}_{opt}(p)$ – *решение, найденное алгоритмом CCBS для любой решаемой задачи, является оптимальным.*

Доказательство При инициализации (см. Алгоритм 3) алгоритм CCBS создаст корень дерева ограничений N_0, содержащий совокупность траекторий, спланированных независимо. Корень дерева не содержит никаких ограничений, следовательно, любое решение из множества $\mathbf{\Pi}_{opt}$, во-первых, имеет стоимость не меньше чем стоимость начального частичного решения $cost(N_0.\Pi)$, во-вторых, удовлетворяет всем его ограничениям ввиду их отсутствия.

Если совокупность траекторий $N_0.\Pi$ не содержит конфликтов, то искомое решение найдено. В противном случае, совокупность траекторий $N_0.\Pi$ содержит по крайней мере один конфликт $InConflict((a_i,t_i),(a_j,t_j))$. Тогда на шаге 1 основного цикла работы алгоритма будут созданы две новых вершины дерева ограничений N_1, N_2, содержащие ограничения $\langle i,a_i,[t_i,t_i^u)\rangle$ и $\langle j,a_j,[t_j,t_j^u)\rangle$, наложенные на агентов i и j соответственно. При этом любое решение из множества $\mathbf{\Pi}_{opt}(p)$ удовлетворяет по крайней мере одному из ограничений, наложенному для устранения этого конфликта. Это утверждение является прямым следствием согласованности накладываемых ограничений (см. Лемма 1). Т.к. любое оптимальное решение удовлетворяет по крайней мере одному из ограничений, то его стоимость не может быть ниже минимальной стоимости среди имеющихся альтернативных частичных решений: $\forall \Pi \in \mathbf{\Pi}_{opt}(p) : cost(\Pi) \geqslant min(cost(N_1.\Pi), cost(N_2.\Pi))$.

Допустим, что эти утверждения выполняются на шаге k основного цикла работы алгоритма. Покажем, что они выполняются и на шаге $k+1$.

На шаге $k+1$ из списка $OPEN$ будет извлечена вершина N, содержащая решение наименьшей стоимости из всех, имеющихся в $OPEN$. Пусть в решении $N.\Pi$ найден конфликт $InConflict((a_l,t_l),(a_m,t_m))$. Для его устранения будут созданы две новых вершины дерева ограничений N_l, N_m, которые содержат по одному дополнительному ограничению $\langle l,a_l,[t_l,t_l^u)\rangle$ и $\langle m,a_m,[t_m,t_m^u)\rangle$. Если в множестве $\mathbf{\Pi}_{opt}(p)$ есть решения, удовлетворяющие множеству ограничений $N.cons$, то они будут удовлетворять по крайней мере одному из ограничений, добавленных в $N_l.cons$, $N_m.cons$. Следовательно их стоимость не может быть меньше минимальной стоимости среди новых созданных частичных решений: $\forall\Pi \in \mathbf{\Pi}_{\boldsymbol{N}opt}(p) : cost(\Pi) \geqslant min(cost(N_l.\Pi), cost(N_m.\Pi))$, где $\mathbf{\Pi}_{\boldsymbol{N}opt}(p)$ – подмножество всех оптимальных решений, удовлетворяющих ограничениям $N.cons$. Т.к. стоимости всех оптимальных решений эквивалентны, это неравенство справедливо для любого оптимального решения: $\forall\Pi \in \mathbf{\Pi}_{opt}(p) : cost(\Pi) \geqslant min(cost(N_l.\Pi), cost(N_m.\Pi))$. Вершины N_l и N_m добавляются в список $OPEN$, следовательно $\forall\Pi \in \mathbf{\Pi}_{opt}(p) : cost(\Pi) \geqslant argmin_{N'\in OPEN}\{cost(N'.\Pi)\}$, где $argmin_{N'\in OPEN}\{cost(N'.\Pi)\}$ – вершина с минимальной стоимостью из списка $OPEN$.

Если же совокупность траекторий $N.\Pi$ не содержит конфликтов, то искомое решение найдено. Учитывая неравенство $\forall\Pi \in \mathbf{\Pi}_{opt}(p) : cost(\Pi) \geqslant argmin_{N'\in OPEN}\{cost(N'.\Pi)\}$, где $argmin_{N'\in OPEN}\{cost(N'.\Pi)\}$ – это вершина N, а также определение множества всех оптимальных решений $\mathbf{\Pi}_{opt}$, решение $N.\Pi$ имеет ту же стоимость, что и любое решение из $\mathbf{\Pi}_{opt}$: $cost(N.\Pi) = cost(\Pi), \forall\Pi \in \mathbf{\Pi}_{opt}(p)$, т.е. $N.\Pi \in \mathbf{\Pi}_{opt}(p)$.■

Стоит отметить, что утверждения, на которых строится доказательство Теоремы 1, в частности, выполнение неравенства $\forall\Pi \in \mathbf{\Pi}_{opt}(p) : cost(\Pi) \geqslant min(cost(N_1.\Pi), cost(N_2.\Pi))$, справедливы только в том случае, если алгоритм планирования индивидуальных траекторий агентов гарантирует нахождение решения минимально возможной стоимости. Алгоритм CSIPP, используемый в качестве планировщика индивидуальных траекторий с учетом накладываемых ограничений, этим свойством обладает (подробнее см. раздел 3.2.3).

3.4 Выводы по главе

В данной главе был описан подход конфликтно-ориентированного поиска, а также предлагаемый алгоритм Continuous Conflict Based Search, который решает задачу многоагентного планирования с учетом возможности совершения действий произвольной продолжительности. Был описан принцип работы подхода конфликтно-ориентированного поиска, а также описан пример задания в классической постановке задачи. Затем были описаны ключевые особенности предлагаемого алгоритма. Было введено новое определение конфликта, которое не привязано к вершинам/ребрам графа, а задается через пары действий и моменты их совершения. Был описан метод проверки наличия конфликтов между парами действий, а также способ поиска конфликтов в совокупности траекторией. Затем были введены интервальные ограничения и описан аналитический метод их расчета. Для работы с таким типом ограничений был предложен алгоритм планирования нижнего уровня CSIPP, приведено описание принципа его работы, в тем числе его псевдокод. Был проведен анализ теоретических свойств алгоритма CSIPP. Результатом этого анализа стали два утверждения с доказательствами, гарантирующие нахождение оптимального решения при условии его существования. Также был приведен и описан псевдокод алгоритма CCBS. Помимо этого, был проведен анализ теоретических свойств предложенного алгоритма CCBS. Основным результатом этого анализа стала теорема, доказывающая оптимальность решений, отыскиваемых алгоритмом CCBS.

Глава 4. Модификации алгоритма

4.1 Оптимальные модификации

Несмотря на наличие у алгоритма CCBS свойства, гарантирующего нахождение решения задачи многоагентного планирования за конечное число итераций, эффективность его работы может быть повышена за счет внедрения дополнительных модификаций. В данном разделе будут рассмотрены модификации, которые повышают эффективность работы алгоритма и при этом сохраняют его главное свойство - гарантию нахождения оптимального решения.

Существует большое число различных модификаций для оригинального алгоритма конфликтно-ориентированного поиска, решающего задачу многоагентного планирования с дискретным временем. Многие из них, такие как, например, прямоугольные конфликты [67] неприменимы к рассматриваемой постановке задачи, т.к. опираются на допущения о 4-связности графа и одинаковую стоимость всех действий. Ряд существующих модификаций может быть применен к рассматриваемой постановке задачи, однако требует адаптации. В данной работе были предложены адаптации следующих трех модификаций:

1. Приоритизация конфликтов (Prioritizing Conflicts, PC) [64]
2. Эвристические функции верхнего уровня (High-Level Heuristics, HL) [65]
3. Непересекающееся разделение (Disjoint Splitting, DS) [96]

Далее будет представлено описание каждой из модификаций, адаптированных к алгориту CCBS и рассматриваемой постановке задачи.

4.1.1 Приоритизация конфликтов

При наличии множества различных конфликтов в одном альтернативном решении встает вопрос о выборе конкретного конфликта, который будет разрешен. Выбор конфликта не влияет на свойства алгоритма, однако, может существенным образом повлиять на эффективность его работы. В алгоритме CCBS, как и в оригинальном алгоритме конфликтно-ориентированного поиска, использует-

ся эвристическое правило, выбирающее наиболее ранний конфликт. Этот выбор обусловлен тем, что устранение наиболее раннего конфликта может повлиять и на конфликты, которые происходят по времени позднее, что может привести к их устранению.

В работе [64] было введено понятие *кардинальности* конфликта и было предложено разделять конфликты на три типа: кардинальный, полукардинальный и некардинальный. Конфликт является кардинальным в том случае, если его устранение приводит к увеличению стоимости решения вне зависимости от того, на кого из конфликтующих агентов было наложено ограничение. Полукардинальным является тот конфликт, чье устранение приводит к увеличению стоимости только в одном из случаев. Некардинальным является конфликт, устранение которого не приводит к увеличению стоимости решения ни в одном из случаев. Авторы предлагают выбирать в первую очередь кардинальные конфликты, т.к. их устранение позволяет повысить стоимость рассматриваемых частичных решений и приблизить её к стоимости искомого неконфликтного решения. В случае отсутствия кардинальных конфликтов в частичном решении предлагается выбирать полукардинальные.

Подобный подход может быть применен и в рассматриваемой постановке задачи. Однако, он требует адаптации. В отличие от классической постановки задачи, где стоимость решения меняется минимум на 1, ввиду дискретности времени, в рассматриваемой постановке задачи стоимость решения может измениться на произвольную величину. Более того, при решении задачи многоагентного планирования на графах нерегулярной структуры, большинство конфликтов являются кардинальными. В связи с чем требуется дополнительный критерий, который позволит ранжировать кардинальные конфликты. Для этого было введено понятие добавочной стоимости, которое характеризует насколько сильно увеличится стоимость решения при разрешении определенного конфликта.

Определение 7. Пусть дан конфликт $InConflict((a_i,t_i),(a_j,t_j))$, который содержится в некотором решении N. При разрешении этого конфликта будут созданы решения N_i и N_j, в которых было наложено ограничение на агента i или j, устраняющее этот конфликт. Разницу в стоимости решений $cost(N.\Pi)$ и $cost(N_i.\Pi)$ обозначим как δ_i, а добавочную стоимость конфликта обозначим как $\Delta(InConflict((a_i,t_i),(a_j,t_j)))$. В таком случае добавочная стоимость конфликта

равна минимальной разнице в стоимости решений между N и $N_k, k = \{i, j\}$:

$$\Delta(InConflict((a_i,t_i),(a_j,t_j))) = min(\delta_i, \delta_j) \quad (4.1)$$

Имея значения добавочной стоимости каждого из конфликтов, содержащихся в решении, алгоритм выбирает тот конфликт, который обладает максимальной добавочной стоимостью. Для определения добавочной стоимости алгоритм производит планирование траекторий агентов, участвующих в конфликте, с учетом дополнительного ограничения, которое накладывается на агента для устранения соответствующего конфликта. Информация о дополнительных стоимостях конфликтов хранится вместе с конфликтами в вершинах дерева ограничений.

4.1.2 Эвристические функции верхнего уровня

Следующая модификация, позволяющая повысить вычислительную эффективность алгоритма, заключается в использовании эвристической функции на верхнем уровне алгоритма. Оригинальный алгоритм CCBS в качестве критерия выбора вершины из дерева ограничений использует стоимость решения (строка 6 алгоритма 3), что, по сути, является неинформированным поиском. Однако этот критерий может быть модифицирован и дополнительно учитывать информацию о текущем решении, которое бы позволило оценивать то, насколько сильно его стоимость отличается от стоимости решения, не содержащего конфликтов. Для этого было предложено использовать информацию о добавочной стоимости конфликтов, содержащихся в решениях. Добавление подобного рода эвристической функции на верхнем уровне алгоритма позволяет осуществить переход от неинформированного поиска к поиску по первому лучшему совпадению (англ. best-first search).

Предлагаемая эвристическая функция оценивает разницу в стоимости между рассматриваемым решением и искомым, которое не содержит конфликтов. Для того, чтобы эвристическая функция не нарушала свойств алгоритма CCBS, она должна обладать свойством допустимости. Свойство допустимости означает, что эвристическая функция никогда не переоценивает минимальную стоимость решения (то есть является нижней оценкой фактической стоимости). Использование суммы добавочных стоимостей всех конфликтов невозможно, т.к. конфликты

могут быть взаимосвязаны и устранение одного конфликта может привести к устранению и других конфликтов, что может привести к нарушению свойства допустимости, т.к. добавление суммарной добавочной стоимости всех конфликтов к стоимости решения может привести к тому, что итоговое значение превысит стоимость оптимального решения. Вместо этого было предложена два способа расчета эвристической функции.

Первый способ основан на решении задачи линейного программирования, которая представима в виде системы неравенств:

$$\begin{cases} x_1 + x_2 \geqslant \Delta(InConflict((a_1,t_1),(a_2,t_2))) \\ x_3 + x_4 \geqslant \Delta(InConflict((a_3,t_3),(a_4,t_4))) \\ \dots \\ x_i + x_j \geqslant \Delta(InConflict((a_i,t_i),(a_j,t_j))) \\ i,j \in [1,K] \end{cases} \tag{4.2}$$

, где x_i, x_j – переменные, соответствующие агентам, участвующим в конфликте, а число неравенств в системе эквивалентно числу конфликтов в рассматриваемом частичном решении. При этом целевая функция имеет вид:

$$\sum_{k=0}^{K} x_k \to min \tag{4.3}$$

Таким образом, каждый агент участвует в итоговой сумме только один раз, даже если имеет несколько конфликтов.

Второй способ расчета эвристической функции для верхнего уровня алгоритма CCBS заключается в выборе независимых конфликтов, т.е. таких, в которых участвуют непересекающиеся пары агентов. Благодаря тому, что конфликты являются независимыми, устранение одного конфликта не может привести к уменьшению добавочной стоимости другого конфликта. Следовательно, добавочные стоимости независимых конфликтов могут быть просуммированы и при этом значение их суммы обладает свойством допустимости. Для максимизации значения суммы был предложен подход, в котором конфликты сортируются в порядке убывания добавочных стоимостей, после чего последовательно выбираются конфликты, не имеющие пересечений по агентам с предыдущими выбранными конфликтами.

Рассмотрим пример, изображенный на Рисунке 4.1. Имеются три агента, каждый из которых имеет радиус 0.5. Вершины s_i соответствуют стартовым

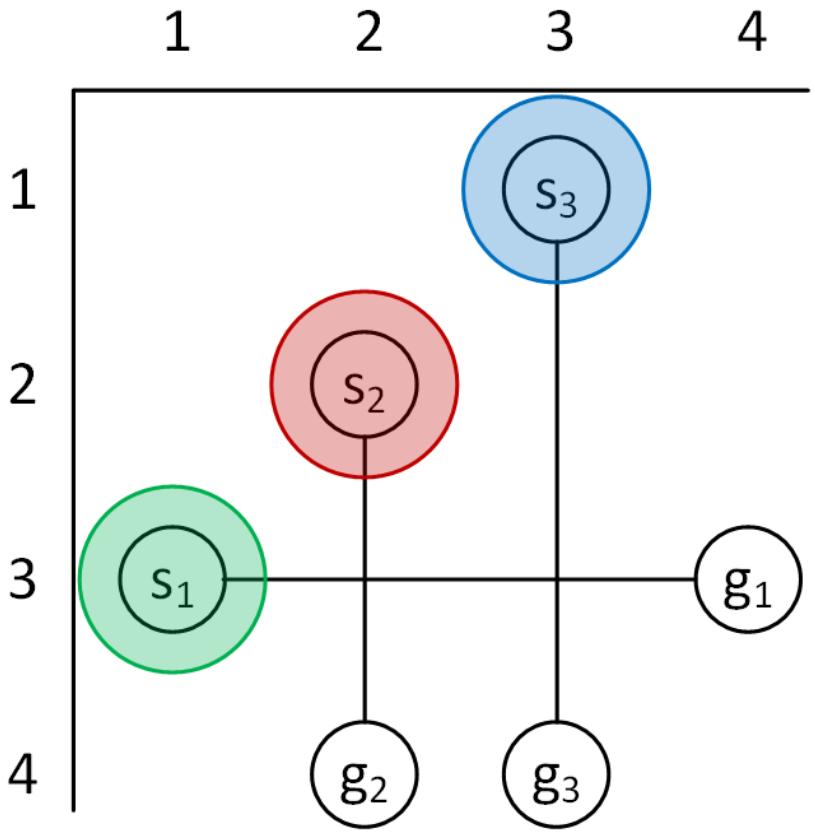


Рисунок 4.1 — Пример задания с несколькими взаимосвязанными конфликтами. s_i – стартовые положения агентов, g_i – целевые. Радиус каждого агента – 0.5

положениям агентов, g_i – целевым. Начальное решение, содержащее траектории, спланированные для каждого из агентов независимо, имеет два конфликта: $InConflict((s_1 \to g_1,0),(s_2 \to g_2,0))$, $InConflict((s_1 \to g_1,0),(s_3 \to g_3,0))$. Добавочные стоимости конфликтов имеют следующие значения: $\Delta(InConflict((s_1 \to g_1,0),(s_2 \to g_2,0))) = \sqrt{2}$, $\Delta(InConflict((s_1 \to g_1,0),(s_3 \to g_3,0))) = \sqrt{2}$, т.к. для устранения каждого из этих конфликтов один из агентов, участвующих в конфликте, должен совершить действие-ожидание продолжительностью $\sqrt{2}$. Начальное решение имеет стоимость 8. В случае использования в качестве эвристической функции суммы добавочных стоимостей всех конфликтов, значение эвристической функции будет равно $2\sqrt{2}$, однако оптимальное решение имеет стоимость $8 + \sqrt{2}$, т.к. для устранения обоих конфликтов необходимо чтобы первый агент совершил действие ожидание продолжительностью $\sqrt{2}$ и тем самым устранил оба конфликта. Таким образом, этот пример показывает почему использование суммарной добавочной стоимости всех конфликтов может привести к превышению стоимости оптимального решения и нарушить свойства алгоритма. Использова-

ние любого из предложенных методов даст оценку $\sqrt{2}$ для начального решения и тем самым не превысит стоимость оптимального решения.

4.1.3 Непересекающееся разделение

В алгоритме CCBS для устранения конфликтов используется подход, когда на одного из агентов, участвующих в конфликте, накладывается ограничение, запрещающее ему совершать конфликтное действие в течение определенного интервала времени. При этом рассматриваются оба альтернативных варианта, а соответствующая пара ограничений является согласованной, т.е. любое неконфликтное решение удовлетворяет по крайней одному из из этих ограничений. Однако, решение может удовлетворять и обоим ограничениям из этой пары и при этом быть оптимальным. Это приводит к тому, что в дереве ограничений могут содержаться вершины с одинаковым набором путей. Наличие идентичных решений в дереве приводит к замедлению работы алгоритма. Подход непересекающегося разделения использует два типа ограничений – *негативные* и *положительные*.

Определение 8. Ограничение, которое запрещает агенту i начинать совершать действие a_i в течение интервала $[t_i,t_i^u)$, называется негативным и обозначается как $\overline{\langle i,a_i,[t_i,t_i^u)\rangle}$.

Определение 9. Ограничение, которое требует, чтобы агент i начал совершать действие a_i в течение интервала $[t_i,t_i^u)$, называется положительным и обозначается как $\langle i,a_i,[t_i,t_i^u)\rangle$.

Пусть дана вершина дерева N, у которой в решении $N.\Pi$ содержится конфликт $InConflict((a_i,t_i),(a_j,t_j))$. Для устранения этого конфликта создаются две новых вершины N_i, N_j, в которых для устранения конфликта на одного из агентов накладывается негативное ограничение – $\overline{\langle i,a_i,[t_i,t_i^u)\rangle}$ или $\overline{\langle j,a_j,[t_j,t_j^u)\rangle}$ соответственно. Однако, помимо наложения негативных ограничений, в одной из ветвей накладывается положительное ограничение $\langle i,a_i,[t_i,t_i^u)\rangle$. Согласно Лемме 1 негативные ограничения, накладываемое на агентов, являются согласованными. Пара ограничений $\langle i,a_i,[t_i,t_i^u)\rangle$ и $\overline{\langle i,a_i,[t_i,t_i^u)\rangle}$ также является согласованной, т.к. любое оптимальное решение, которое включает в себя действие a_i, которое агент i совершает в интервал времени $[t_i,t_i^u)$, может быть найдено через ветвь дерева, в котором

на агента i было наложено положительное ограничение. Если же оптимальное решение не содержит действие a_i для агента i, либо оно должно быть совершено за пределами интервала $[t_i,t_i^u)$, то такое решение может быть найдено через ветвь дерева, в котором на агента i было наложено негативное ограничение $\overline{\langle i,a_i,[t_i,t_i^u)\rangle}$. Таким образом происходит непересекающееся разделение, т.к. агент i, на которого наложено положительное ограничение, в одной ветви может начать совершать действие a_i только внутри конфликтного интервала $[t_i,t_i^u)$, а в другой - только за его пределами.

Наличие положительных ограничений приводит к изменению логики работы алгоритма планирования индивидуальных траекторий. Ограничение $\langle i,a_i,[t_i,t_i^u)\rangle$ представляет собой своего рода промежуточную цель L_1, т.е. прежде чем агент достигнет целевого положения, он должен совершить действие a_i в какой-либо момент времени в течение интервала $[t_i,t_i^u)$. Таким образом, задача планирования индивидуальной траектории агента вместо планирования из стартового положения $Starts(i)$ в целевое $Goals(i)$ преобразуется к задаче планирования последовательности вида $Starts(i) \rightarrow L_1 \rightarrow Goals(i)$. В случае наличия нескольких положительных ограничений, соответствующие им промежуточные цели сортируются по времени, выстраивая тем самым последовательность.

Пусть дано всего одно ограничение, которое является положительным – $\langle i,a_i,[t_i,t_i^u)\rangle$. Действие a_i представляет собой действие перемещения $A \rightarrow B$. В таком случае алгоритм планирования нижнего уровня выполняет следующие три процедуры:

1. Планирует траекторию из стартового положения $Starts(i)$ до промежуточной цели – вершины A.
2. Выполняет действие, на которое было наложено положительно ограничение, т.е. переход из A в B.
3. Планирует траекторию из вершины B до целевого состояний $Goals(i)$.

В данном примере рассматривалась ситуация, когда было наложено всего одно ограничение. Однако в общем случае на агента i могут быть наложены и другие ограничения, в том числе на действия-ожидания в вершинах A, B или даже на действие-перемещение $(A \rightarrow B)$. Наличие дополнительных негативных ограничений может приводить к возникновению ситуаций, когда достижение состояния в вершине A и выполнение действия $(A \rightarrow B)$ не приводит к нахождению оптимального решения. Рассмотрим пример, представленный на Рисунке 4.2. На агента i наложены следующие 3 ограничения:

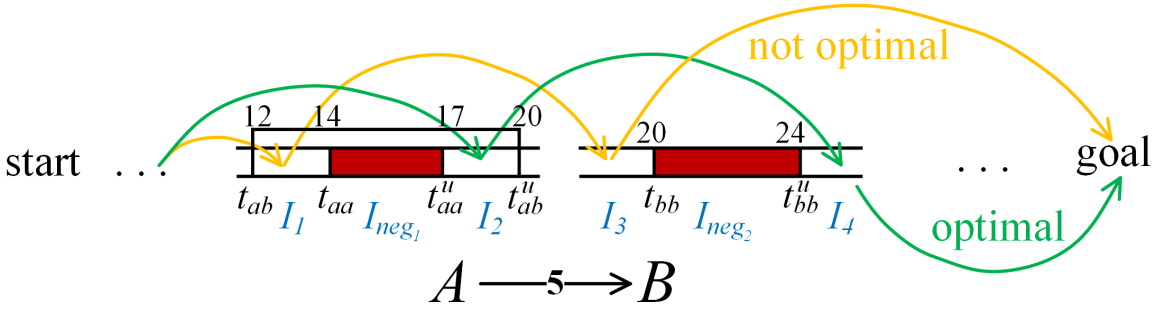


Рисунок 4.2 — Пример ситуации, в которой достижение состояния в более ранний интервал времени приводит к получению субоптимального решения.

– Положительное ограничение на действие перемещения из A в B: $(i, (A \rightarrow B), [t_{ab}, t^u_{ab})$.
– Негативное ограничение на действие ожидания в A: $(i, (A \rightarrow A), [t_{aa}, t^u_{aa})$.
– Негативное ограничение на действие ожидания в B: $(i, (B \rightarrow B), [t_{bb}, t^u_{bb})$.

, где $t_{ab} < t_{aa} < t^u_{aa} < t^u_{ab}$.

Таким образом, ограничение на действие ожидания в A приводит к возникновению двух безопасных интервалов $I_1 = [0,t_{aa}]$ и $I_2 = [t^u_{aa},\infty)$, которые пересекаются с интервалом позитивного ограничения. В свою очередь негативное ограничение на действие ожидания в вершине B также приводит к появлению двух безопасных интервалов $I_3 = [0,t_{bb}]$ и $I_4 = [t^u_{bb},\infty)$.

Теперь предположим, что есть две траектории, которые удовлетворяют положительному ограничению, один из которых достигает A до момента времени t_{aa} (показан желтым цветом), а другой достигает A после t^u_{aa}. (показан зеленым цветом). Очевидно, что траектория с наименьшей стоимостью, достигающая промежуточную цель — это та, которая достигает A до t_{aa}, но для нахождения оптимального решения необходимо использовать вторую траекторию. Рисунке 4.3 иллюстрирует еще более экстремальный случай, когда выбор траектории с наименьшей стоимостью достижения промежуточной цели, не может привести к построению полной траектории до конечного целевого положения, поскольку она достигает B в течение конфликтного интервала (показана красным цветом).

Теоретически существует бесконечное число траекторий, удовлетворяющих положительному ограничению $\langle i,(A \rightarrow B),[t_i,t^u_i)\rangle$, т.е. таких, которые достигают A в пределах $[t_i,t^u_i)$. Однако, поиск только одной траектории, достигающей A в минимально возможное время приводит к потере свойств полноты и оптимальности. Чтобы гарантировать полноту и оптимальность, необходимо найти траекторию с наименьшей стоимостью достижения A *для каждого безопасного*

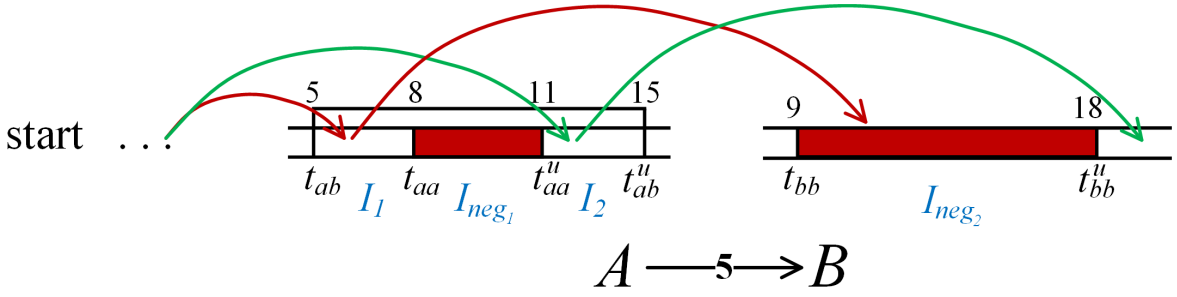


Рисунок 4.3 — Пример ситуации, в которой достижение состояния в более ранний интервал времени приводит к невозможности совершения действия, на которое было наложено позитивное ограничение.

интервала в A, который имеет пересечение с интервалом $[t_i,t_i^u)$. Только в этом случае можно считать, что были рассмотрены все возможные варианты достижения вершины A, что сохраняет полноту. Оптимальность сохраняется благодаря нахождению траектории с наименьшей стоимостью для каждого безопасного интервала.

Для учета возможности наличия нескольких интервалов в промежуточных целях была предложена модификация GSIPP – Generealized Safe Interval Path Planning, которая 1) допускает наличие нескольких целевых состояний, по одному на каждый безопасный интервал в вершине A, пересекающийся с $[t,t^u)$, и 2) строит набор траекторий, по одной для каждого целевого состояния. Для каждой из этих траекторий выполняется действие положительного ограничения, т.е. переход $(A \rightarrow A)$. Получившиеся траектории могут заканчиваться в разных безопасных интервалах в B, которые затем становятся разными начальными состояниями при поиске траекторий от B к следующей промежуточной. Таким образом, алгоритм GSIPP способен находить решение в случаях наличия множественных стартовых и целевых состояний. Принцип работы алгоритма $GSIPP$ заключается в следующем:

1. На этапе инициализации в список $OPEN$ добавляется *все* стартовые состояния с различными безопасными интервалами.
2. Идет стандартный процесс работы алгоритма SIPP.
3. Алгоритм завершает свою работу тогда, когда *все* состояния, соответствующие целевому положению, будут раскрыты.

Благодаря тому, что поиск из *всех* стартовых состояний до *всех* целевых состояний осуществляется одновременно внутри одного и того же цикла работы

Algorithm 4: Псевдокод алгоритма GSIPP

Input: Negative constraints $C^{(-)}$
Input: Positive constraints $C^{(+)}$
Input: Agent i
$\mathcal{S} \leftarrow$ ComputeSafeIntervals($C^{(-)}$)
$\mathcal{L} \leftarrow$ ComputeLandmarks($C^{(+)}$, $\mathcal{S}$)
Starts $\leftarrow \{s_i\}$
foreach *landmark* $l = (i,$*move*$(A,B),[t,t^u))$ *in* $\mathcal{L}$ **do**
    Goals $\leftarrow$ computeGoals(l)
    Plans $\leftarrow$ GSIPP(Starts, Goals)
    Starts $\leftarrow \varnothing$
    foreach *plan in Plans* **do**
        Append move(A,B) to $plan$
        Add last state in $plan$ to Starts
    Starts $\leftarrow$ Prune Plans/Starts if possible
return GSIPP(Starts, g_i)

алгоритма, сложность алгоритма $GSIPP$ такая же как у оригинального алгоритма $SIPP$. В худшем случае алгоритм раскроет все состояния, которые есть в графе.

Псевдокод планировщика нижнего уровня алгоритма CCBS, использующего модификацию непересекающегося разделения, представлен в Алгоритме 4.

4.1.4 Анализ теоретических свойств

Все вышеописанные модификации, т.е. приоритизация конфликтов, непересекающееся разделение и эвристические функции верхнего уровня, могут быть скомбинированы вместе с алгоритмом CCBS. Алгоритм, который комбинирует в себе все улучшения, был назван Improved Continuous Conflict Based Search (ICCBS).

Пусть дана произвольная задача многагентного планирования p, имеющая решение, $p \in \boldsymbol{P}^+$, а также некоторое решение этой задачи, найденное с помощью алгоритма ICCBS, обозначаемое как Π_{ICCBS}.

Теорема 2. $\forall p \in \boldsymbol{P}^{+} : \Pi_{ICCBS} \in \mathbf{\Pi}_{opt}(p)$ – *решение, найденное алгоритмом ICCBS для любой решаемой задачи, является оптимальным.*

Доказательство Следуя доказательству Теоремы 1, решения, отыскиваемые алгоритмом CCBS, являются оптимальными. Таким образом, для того чтобы гарантировать, что решение Π_{ICCBS} является оптимальным, необходимо и достаточно доказать, что ни одна из модфикаций не приводит к нарушению этого свойства.

Модификация непересекающегося разделения вводит дополнительные ограничения – положительные. Необходимо доказать, что накладываемые дополнительные положительные ограничения являются согласованными. Пара ограничений называется согласованной, если любое оптимальное решение удовлетворяет по крайней мере одному из них (подробнее см. Определение 6). Рассмотрим произвольную вершину дерева ограничений N, в которой совокупность траекторий $N.\Pi$ содержит конфликт $InConflict((a_i,t_i),(a_j,t_j))$. Для его устранения создаются 2 новые вершины N_1, N_2: вершина N_1 содержит дополнительное негативное ограничение $\overline{\langle i,a_i,[t_i,t_i^u)\rangle}$, а вершина N_2 – негативное ограничение $\overline{\langle j,a_j,[t_j,t_j^u)\rangle}$ и положительное ограничение $\langle i,a_i,[t_i,t_i^u)\rangle$. Следуя Лемме 1, пара ограничений $\overline{\langle i,a_i,[t_i,t_i^u)\rangle}$ и $\overline{\langle j,a_j,[t_j,t_j^u)\rangle}$ является согласованной. Пара ограничений $\overline{\langle i,a_i,[t_i,t_i^u)\rangle}$ и $\langle i,a_i,[t_i,t_i^u)\rangle$ также является согласованной, т.к., если агенту i требуется совершить действие a_i в течение интервала $[t_i,t_i^u)$, то он может (и должен) совершить его в ветви дерева, где на агента i было наложено ограничение$\langle i,a_i,[t_i,t_i^u)\rangle$. Если же агенту i требуется совершить действие a_i в некоторый момент времени $t \notin [t_i,t_i^u)$, то он может совершить его в той ветви, где было наложено ограничение $\overline{\langle i,a_i,[t_i,t_i^u)\rangle}$. Пара ограничений $\overline{\langle j,a_j,[t_j,t_j^u)\rangle}$ и $\langle i,a_i,[t_i,t_i^u)\rangle$ накладывается вместе в одной и той же вершине дерева ограничений и она также является согласованной. Как было показано в доказательстве Леммы 1, не существует такого оптимального решения, которое включает в себя оба действия a_i и a_j, совершаемые агентами i и j в некоторые моменты времени в течение интервалов $[t_i,t_i^u)$ и $[t_j,t_j^u)$ соответственно. Следовательно, любое оптимальное решение, удовлетворяющее ограничению $\langle i,a_i,[t_i,t_i^u)\rangle$, удовлетворяет и $\overline{\langle j,a_j,[t_j,t_j^u)\rangle}$. Таким образом, дополнительное положительное ограничение $\langle i,a_i,[t_i,t_i^u)\rangle$, накладываемое алгоритмом ICCBS, не может привести к потере какого-либо оптимального решения. Следовательно, модификация непересекающегося разделения не приводит к потере свойства оптимальности.

Критерий выбора конфликта в алгоритмах, использующих подход конфликтно-ориентированного поиска, может быть абсолютно произвольным. Следовательно, используемый в алгоритме ICCBS принцип приоритизации конфликтов на основе добавачной стоимости не может повлиять на его свойства, в частности, на свойство оптимальности.

Использование эвристической функции на верхнем уровне улгоритма CCBS может привести к потере свойства оптимальности в том случае, если выдаваемые значения приводят к переоценке стоимости оптимального решения. Оба предложенных способа расчета эвристической функции основаны на использовании добавочных стоимостей конфликтов $\Delta(InConflict((a_i,t_i),(a_j,t_j)))$. Следуя Определению 7, добавочная стоимость конфликта – это минимальная разница в стоимости текущего решения N и его вершин-потомков $N_k, k = \{i,j\}$. Следовательно, любое оптимальное решение, удовлетворяющее совокупности ограничений $N.cons$, не может иметь стоимость меньше, чем $cost(N) + \Delta(InConflict((a_i,t_i),(a_i,t_i)))$, т.к. ограничения, накладываемые на агентов i и j для устранения конфликта $InConflict((a_i,t_i),(a_i,t_i))$ являются согласованной парой ограничений. Иными словами, значение $cost(N)+\Delta(InConflict((a_i,t_i),(a_i,t_i)))$ является допустимой оценкой стоимости оптимального решения, при условии существования оптимального решения, удовлетворяющего $N.cons$. Предлагаемые в работе методы рассчета эвристической функции опираются на добавочные стоимости всех конфликтов, содержащихся в рассматриваемом частичном решении. Первый способ, основанный на решении задачи линейного программирования, не переоценивает фактическую стоимость бесконфликтного решения благодаря тому, что любой агент, имеющий конфликты, участвует в сумме, используемой для расчета целевой функции, лишь один раз, а значение функции необходимо минимизировать. Второй предложенный способ расчета эвристики верхнего уровня не переоценивает фактическую стоимость бесконфликтного решения благодаря тому, что для суммирования выбираются добавочные стоимости конфликтов только между непересекающимися парами агентов. Выбор конфликтов только между непересекающимися парами агентов позволяет гарантировать, что устранение одного конфликта не окажет влияния на добавочную стоимость другого конфликта. Следовательно, если существует оптимальное решение, удовлетворяющее ограничениям $N.cons$, то его стоимость должна включать в себя добавочные стоимости всех непересекающихся пар конфликтов. Таким образом, оба предложенных способа расчета

эвристической функции обеспечивают допустимую оценку для любой из вершин дерева, совокупностям ограничений которых удовлетворяет по крайней мере одно оптимальное решение. Допустимость оценки для вершин, совокупностям ограничений которых не удовлетворяет ни одно из оптимальных решений не требуется, т.к. эти вершины в любом случае не могут привести к отысканию оптимального решения.

Таким образом, ни одна из модификаций не приводит к нарушению свойства оптимальности, которым обладает оригинальный алгоритм CCBS. Следовательно, решение Π_{ICCBS}, отыскиваемое алгоритмом ICCBS, также гарантированно является оптимальным. ■

4.2 Субоптимальные модификации

Одним из возможных способов повышения вычислительной эффективности алгоритма CCBS является отказ от поиска гарантированно оптимальных решений и поиск ограниченно субоптимальных решений вместо них. Для этого необходимо изменить принцип выбора альтернативных решений, извлекаемых из дерева на каждой итерации работы алгоритма. Для этого предлагается использовать подход, используемый в таких алгоритмах как $A_\varepsilon{}^*$ [97] и EES [98]. Ключевой особенностью этих алгоритмов является то, что они используют вторичную эвристику, влияющую на порядок раскрытия вершин в процессе работы алгоритма. Применяя подход этих алгоритмов к алгоритму CCBS, можно получить модификацию, обладающую повышенной вычислительной эффективностью, которая отыскивает ограниченно-субоптимальные решения.

4.2.1 Модификация на основе алгоритма $A_\varepsilon{}^*$

В отличие от классических алгоритмов эвристического поиска, таких как A*, в которых используются два списка – OPEN и CLOSED, в алгоритме $A_\varepsilon{}^*$ используется третий список – FOCAL. Список FOCAL содержит в себе все вершины, f-значение которых не превышает минимальное f-значение из списка

OPEN более чем в (1+ε) раз. Таким образом множество вершин, находящихся в FOCAL, является подмножеством всех вершин, находящихся в OPEN. На каждом шаге алгоритм раскрывает не ту вершину, которая обладает минимальным f-значением в списке OPEN, а ту, что находится в начале списка FOCAL. При этом сам список FOCAL может быть отсортирован по абсолютно любому критерию. Гарантия того, что итоговое найденное решение не превысит стоимость оптимального решения более чем в $(1 + \varepsilon)$ раз достигается за счет того, что в список FOCAL попадают только те вершины, которые удовлетворяют ограничению субоптимальности. Применительно к алгоритмам, использующим подход конфликтно-ориентированного поиска, которые оперируют деревом альтернативных решений на верхнем уровне алгоритма, список FOCAL может использоваться для выбора текущего альтернативного решения на каждом шаге алгоритма. Наиболее подходящим критерием, по которому решения сортируются в списке FOCAL, представляется количество конфликтов, которые присутствуют в том или ином альтернативном решении. Предпочитая рассматривать те решения, которые содержат наименьшее число конфликтов, алгоритм может найти решение, не содержащее конфликтов, за меньшее число итераций. Подобный подход был ранее применен в алгоритме ECBS [76].

4.2.2 Модификация на основе алгоритма Explicit Estimation Search

В работе [77] был предложен альтернативный вариант субоптимальной версии алгоритма CBS. В ней вместо подхода алгоритма ${A_\varepsilon}^*$ используется принцип метода Explicit Estimation Search (EES) [98]. Помимо списка FOCAL, аналогичный тому, что используется в алгоритме ${A_\varepsilon}^*$, EES оперирует еще одним списком – CLEANUP. Список CLEANUP по сути является регулярным списком OPEN, в котором все вершины отсортированы в порядке возрастания f-значения. Список OPEN в свою очередь отсортирован по $\hat{f}$ -значению, посчитанному с использованием произвольной эвристической функции $\hat{h}$. Список FOCAL содержит подмножество вершин из списка OPEN, удовлетворяющие ограничению $\hat{f}(n) \leqslant (1 + \varepsilon)\hat{f}(best_{\hat{f}}), \hat{f}(best_{\hat{f}}) - \hat{f}$ - значение вершины, обладающей минимальным $\hat{f}$-значением в списке OPEN. В свою очередь вершины в списке FOCAL отсортированы по собственному произвольному критерию d. На каждом шаге алгоритма

выбирается вершина из начала одного из этих списков. Принцип выбора заключается в следующем:

1. Берется первая вершина из списка FOCAL, если она удовлетворяет ограничению на фактор субоптимальности: $if\ f(best_d) \leqslant (1+\varepsilon)f(best_f)$
2. Если условие п.1 не выполняется, то берется первая вершина из списка OPEN, если она удовлетворяет ограничению на фактор субоптимальности: $if\ f(best_{\hat{f}}) \leqslant (1+\varepsilon)f(best_f)$
3. Если п.1 и п.2 не удовлетворяют условиям, то берется первая вершина из списка CLEANUP, отсортированного по f-значениям.

Для вычисления значений функции $\hat{h}$ в оригинальном алгоритме EES используется онлайн-обучение, которое основано на расчете ошибок оценки стоимости и расстояния в процессе поиска решения. Пусть заданы две эвристические функции: h – допустимая эвристическая функция, которая оценивает стоимость пути от произвольной вершины до целевой; d – эвристическая функция, оценивающая расстояние до целевой вершины. Используя их, можно рассчитать ошибку стоимости или расстояния за одну итерацию работы алгоритма:

$$\varepsilon_d(n) = d(bc(n)) - (d(n) - 1); \varepsilon_h(n) = h(bc(n)) - (h(n) - c(n,bc(n))) \qquad (4.4)$$

, где $bc(n)$ – это лучший потомок n, т.е. тот, что имеет наименьшее $\hat{f}$-значение, а $c(n,bc(n))$ – стоимость перехода от n к $bc(n)$. Значения этих ошибок вычисляются на каждой итерации работы алгоритма и на их основе рассчитываются средние значения ошибок $\overline{\varepsilon_d}$ и $\overline{\varepsilon_h}$. Эти значения используются для расчета функции $\hat{h}$:

$$\hat{h}(n) = h(n) + d(n)\overline{\varepsilon_h}(n)/(1 - \overline{\varepsilon_d}(n)) \qquad (4.5)$$

Для использования алгоритма EES на верхнем уровне алгоритма CCBS возможно применение формулы, предложенной для алгоритма EECBS. В алгоритмах CBS и CCBS на верхнем уровне не используется какая-либо эвристическая функция. Поэтому f-значениями вершин являются стоимости альтернативных решений, а для расчета ошибок используются следующие формулы:

$$\begin{aligned} \varepsilon_h(n) &= cost(bc(n)) - cost(n) \\ \varepsilon_d(n) &= h_c(bc(n)) - (h_c(n) - 1) \end{aligned} \qquad (4.6)$$

, где $cost(n)$ – стоимость альтернативного решения, содержащегося в вершине n, а $h_c(n)$ – количество конфликтов соответствующего решения. В свою очередь $\hat{h}$

имеет линейную зависимость от числа конфликтов, содержащихся в альтернативном решении вершины n и рассчитывается следующим образом:

$$\hat{h}(n) = h_c(n)\overline{\varepsilon_h}(n)/(1 - \overline{\varepsilon_d}(n)), \tag{4.7}$$

Таким образом, в алгоритме CCBS+EES вершины в списке FOCAL отсортированы по возрастанию значения h_c, т.е. по количеству конфликтов, в списке OPEN вершины отсортированы по возрастанию значения $\hat{f}(n) = cost(n) + \hat{h}(n)$, а список CLEANUP отсортирован по возрастанию значения стоимости решения $cost(n)$.

Псевдокод алгоритмов CCBS+FOCAL и CCBS+EES

С точки зрения основного цикла работы алгоритма, предложенные модификации практически ничем не отличаются от оригинального алгоритма CCBS (за исключением строки 14). Все основные изменения заключены в процедурах getBestNode и addNode. Процедура addNode различается лишь тем, в какие списки попадает альтернативное решение, т.к. у субоптимальных модификаций, помимо списка OPEN, также есть дополнительные списки FOCAL (у CCBS+FOCAL) или FOCAL и CLEANUP (у CCBS+EES). Далее представлены псевдокоды процедуры getBestNode в зависимости от модификации. Предполагается, что списки OPEN, FOCAL и CLEANUP являются отсортированными по соответствующему критерию.

Наиболее простую процедуру выбора текущего лучшего решения имеет оригинальный алгоритм CCBS – из списка OPEN выбирается вершина минимальной стоимости.

Algorithm 5: Процедура выбора текущего решения алгоритма CCBS

1 $bestN \leftarrow OPEN.front()$
2 delete $bestN$ from $OPEN$
3 **return** bestN

В случае CCBS+FOCAL выбирается решение, обладающее минимальным числом конфликтов, т.к. именно по этому критерию отсортирован список FOCAL.

При этом нет необходимости проверять ограничение субоптимальности, т.к. в списке FOCAL содержатся только те решения, которые ему удовлетворяют. При этом возможна ситуация, что после удаления выбранного решения из списков меняется значение минимальной стоимости решений в списке OPEN (т.е. текущее выбранное решение находилось в начале не только списка FOCAL, но и OPEN). В этом случае необходимо обновить список FOCAL, т.к. возможно, что в него можно добавить большее число решений, которые теперь удовлетворяют ограничению на субоптимальность (строки 3-4 Алгоритма 6).

Algorithm 6: Процедура выбора текущего решения алгоритма CCBS+FOCAL

```
1 bestN ← FOCAL.front()
2 delete bestN from OPEN and FOCAL
3 if minimal cost in OPEN has changed then
4     update FOCAL
5 return bestN
```

В свою очередь алгоритм CCBS+EES выбирает решение из начала списка FOCAL, если оно удовлетворяет ограничению на субоптимальность (строки 2-4 Алгоритма 7). В противном случае выбирается решение из начала списка OPEN, если оно удовлетворяет ограничению на субоптимальность (строки 5-6 Алгоритма 7). В крайнем случае берется решение из начала списка CLEANUP, которое гарантированно удовлетворяет ограничению на субоптимальность, т.к. этот список отсортирован именно по критерию стоимости (строки 7-8 Алгоритма 7). Затем решение удаляется из соответствующих списков (строка 9 Алгоритма 7) и в случае изменения минимального $\hat{f}$ - значения происходит обновление списка FOCAL, т.к. теперь в него могут быть добавлены дополнительные решения (строки 10-11 Алгоритма 7).

Теоретические свойства

Предложенные модификации алгоритма CCBS, а именно CCBS+FOCAL и CCBS+EES обладают свойствами ограниченной субоптимальности, т.е. гаранти-

Algorithm 7: Процедура выбора текущего решения алгоритма CCBS+EES

```
1  bestCost ← CLEANUP.front().cost
2  if FOCAL.front().cost ⩽ bestCost · w then
3      bestN ← FOCAL.front()
4      delete bestN from FOCAL
5  else if OPEN.front().cost ⩽ bestCost · w then
6      bestN ← OPEN.front()
7  else
8      bestN ← CLEANUP.front()
9  delete bestN from OPEN and CLEANUP
10 if minimal f̂ in OPEN has changed then
11     update FOCAL
12 return bestN
```

руют, что отыскиваемые решения имеют стоимость, не превышающую стоимость оптимального решения более чем в w раз. Для доказательства этих утверждений, сформулируем следующую Лемму:

Лемма 2. *Значение минимальной стоимости решения в списке OPEN монотонно не убывает.*

Доказательство Допустим, что утверждение Леммы 2 ложно. В таком случае должна существовать итерация алгоритма n, на которой в список OPEN будет добавлено некоторая вершина N', содержащая решение N'.П, стоимость которого меньше, чем стоимость решения вершины N, раскрываемой на итерации n. Иными словами, решение N'.П является альтернативным решением, которое было сгенерировано на основе решения N.П, при этом $N'.cons$ содержит в себе все его ограничения $N.cons$, а также некоторое дополнительное ограничение $\langle i,a_i,[t_i,t_i^u)\rangle$. Таким образом, набор траекторий, содержащихся в N', отличается от набора траекторий в N только траекторией агента i, на которого было наложено дополнительное ограничение. При условии, что алгоритм, используемый для планирования индивидуальных траекторий, обладает свойствами полноты и оптимальности, как в случае с алгоритмом CSIPP, дополнительное ограничение не может привести к уменьшению стоимости траектории. Следовательно, стоимость

всего решения N'.П так же не может быть уменьшена и не может быть меньше стоимости решения N.П, что вызывает противоречие со сделанным допущением.

Утверждение 3. Алгоритм CCBS+FOCAL является ограниченно-субоптимальным, т.е. гарантирует, что стоимость найденного им решения не превышает оптимального решения более чем в w раз, где w – фактор субоптимальности.

Доказательство Доказательство этого утверждения основано на том, что, во-первых, алгоритм CCBS является оптимальным, т.е. значение минимальной стоимости решения, содержащегося в списке OPEN, не превышает стоимость оптимального решения. Во-вторых, в список FOCAL попадают только те альтернативные решения, стоимость которых удовлетворяет ограничению на субоптимальность. Проверка на ограничение осуществляется в момент, когда решение попадает в список FOCAL. При этом на любой последующей итерации, все решения, содержащиеся в списке FOCAL, продолжают удовлетворять ограничению на субоптимальность. Доказательство этого утверждения не требуется, т.к. оно является прямым следствием Леммы 2. Таким образом, какое бы решение не было извлечено из списка FOCAL на любой итерации алгоритма, оно удовлетворяет ограничению на субоптимальность. Следовательно, итоговое решение, извлеченное на последней итерации работы алгоритма, является ограниченно субоптимальным.■

Утверждение 4. Алгоритм CCBS+EES является ограниченно-субоптимальным, т.е. гарантирует, что стоимость найденного им решения не превышает оптимального решения более чем в w раз, где w – фактор субоптимальности.

Доказательство Как и в случае с алгоритмом CCBS+FOCAL, доказательство этого утверждения основано в первую очередь на том, что алгоритм CCBS является оптимальным, т.е. значение минимальной стоимости решения, содержащегося в списках OPEN и CLEANUP, не превышает стоимость оптимального решения. В отличие от алгоритма CCBS+FOCAL, в алгоритме CCBS+EES список FOCAL, а также список OPEN, могут содержать решения, которые не удовлетворяют ограничению на субоптимальность. Однако проверка на ограничение осуществляется на каждой итерации работы алгоритма непосредственно в момент выбора текущего лучшего решения. Если решение не удовлетворяет ограничению, то оно не извлекается из списка. В крайнем случае, если оба лучших решения

в списках OPEN и FOCAL не удовлетворяют ограничению на субоптимальность, то в качестве текущего лучшего решения выбирается решение из списка CLEANUP, которое отсортировано по стоимости, т.е. фактически выбирается решение минимальной стоимости. Таким образом, на любой итерации алгоритма в качестве текущего лучшего решения может быть выбрано только то решение, которое удовлетворяет ограничению на субоптимальность. Следовательно, итоговое решение, извлеченное на последней итерации работы алгоритма, является ограниченно субоптимальным.■

4.3 Выводы по главе

В данной главе были описаны различные модификации алгоритма CCBS, позволяющие повысить вычислительную эффективность алгоритма. Первые три модификации – приоритизация конфликтов, непересекающееся разделение, а также эвристики верхнего уровня позволяют повысить вычислительную эффективность алгоритма и при этом сохранить свойство оптимальности алгоритма CCBS. Для осуществления приоритизации конфликтов было введено понятие добавочной стоимости конфликтов. Устраняя конфликты с наибольшей добавочной стоимостью, алгоритм за меньшее число итераций приближается к стоимости оптимального решения. Эвристики верхнего уровня также используют добавочные стоимости конфликтов для оценки разницы в стоимости текущего решения и оптимального решения, не содержащего конфликтов. Было предложено два способа расчета эвристической функции – на основе решения задачи линейного программирования, а также выбора подмножества независимых конфликтов. Последняя модификация, неперсекающееся разделение, вводит дополнительный тип ограничений – положительный. Это ограничение требует, чтобы агент совершил соответствующее действие в заданный интервал времени. Для возможности осуществления планирования индивидуальных траекторий была предложена модификация алгоритма CSIPP, названная GSIPP, которая позволяет осуществлять планирование с промежуточными целями, а также множественными стартовыми и целевыми состояниями. Стоит также отметить, что все три вышеперечисленные модификации могут быть скомбинированы вместе.

Помимо этого, были предложены две субоптимальные модификации алгоритма – CCBS+FOCAL и CCBS+EES. Обе этих модификации меняют принцип выбора текущего лучшего решения для рассмотрения. Оригинальный алгоритм CCBS в качестве критерия выбора лучшего текущего решения использует стоимость решения, в то время как предложенные модификации – число конфликтов. Алгоритм CCBS+FOCAL использует принцип работа алгоритма $A*_{\varepsilon}$, в то время как алгоритм CCBS+EES использует принцип работы алгоритма EES. Был проведен анализ их теоретических свойств и было показано, что эти модификации гарантируют нахождение субоптимальных решений, чья стоимость не превышает стоимость оптимального решения более чем в w раз, где w – фактор субоптимальности, задаваемый пользователем.

Глава 5. Экспериментальные исследования

Для проведения экспериментальных исследований разработанного алгоритма, его модификаций, а также для сравнения с другими существующими алгоритмами, использовались карты и задания, взятые из открытой коллекции MovingAI[99]. Данная коллекция заданий является широко используемым инструментом, применяемым для тестирования алгоритмов планирования [33; 37; 64; 77; 89; 90]. Коллекция содержит большое количество карт, однако, для проведения тестирования были выбраны 4 карты, различающиеся размерами и топологией:

- `den520d` – карта имеет размер 256×257 вершин и представляет собой вид сверху одной из локаций игры Dragon Age:Origins.
- `empty-16-16` – карта имеет размер 16×16 и является полностью проходимой, т.е. не содержит никаких статических препятствий.
- `rooms-32-32-4` – карта имеет размер 32×32 и состоит из множества комнат, соединенных узкими проходами.
- `warehouse-170-84-2-2` – карта имеет размер 170×84 и содержит большое количество прямоугольных препятствий размером 10×1. Карта имеет регулярную структуру и имитирует складские помещения.

На Рисунке 5.1 показаны графические изображения всех четырех карт.

Для каждой из карт в коллекции MovingAI даны 25 сценариев, содержащие стартовые и целевые вершины для очень большего числа агентов (вплоть до 1000 в зависимости от размеров карты). В связи с этим, тестирование алгоритмов проводилось по принципу постепенного увеличения числа агентов в заданиях. Изначально из сценария брались лишь первые 2 агента. В случае, если алгоритм

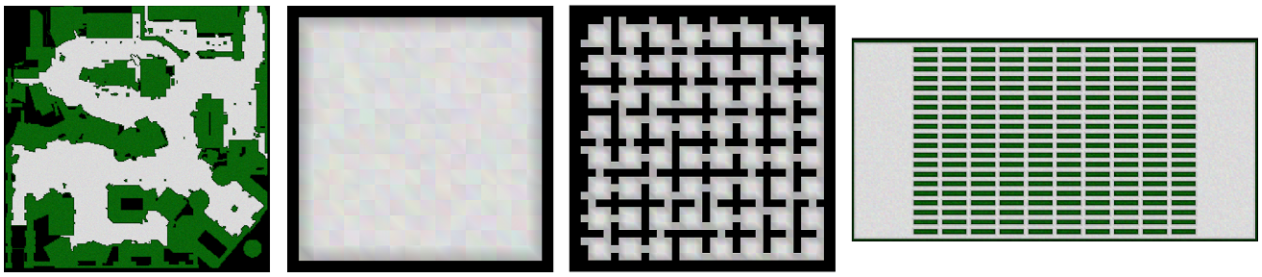


Рисунок 5.1 — Графическое представление всех 4х карт, взятых из коллекции MovingAI, на которых осуществлялось тестирование алгоритмов. а) `den520d`, б) `empty-16-16`, в) `rooms-32-32-4`, г) `warehouse-170-84-2-2`.

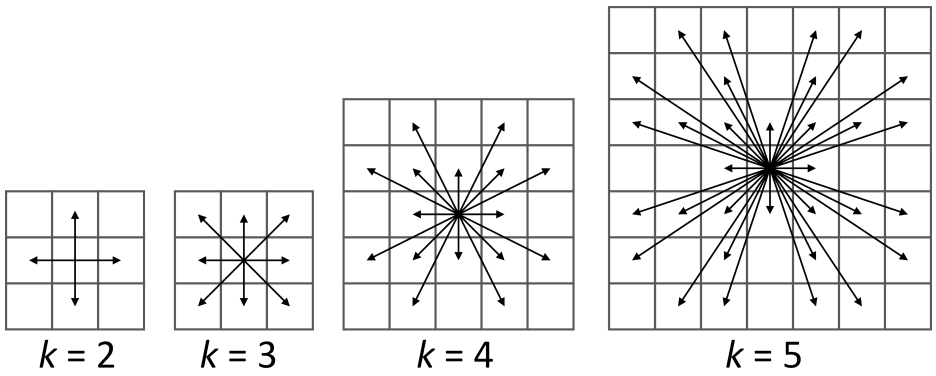


Рисунок 5.2 — Примеры различной связности графов регулярной декомпозиции.

успешно решил задание с текущим количеством агентов, из сценария бралась информация о стартовом/целевом положении следующего агента и эта информация добавлялась в задание. Таким образом, число агентов в заданиях постепенно увеличивалось. Их число увеличивалось до тех пор, пока алгоритм был способен найти решение в течение установленного лимита времени, составлявшего 30 секунд. При тестировании алгоритмов, решающих задачу многогаентного планирования в классической постановке, используется только одна связность графа – когда агент могут перемещаться в 4х направлениях. Однако в рассматриваемой постановке связность графа можно варьировать, увеличивая количество ребер, т.е. возможных действий перемещений. В проведенных экспериментальных исследованиях использовались 2^k-связные ГРД, где k – коэффициент связности. Возможные переходы в зависимости от коэффициента связности графа показаны на Рисунке 5.2. Более высокая связность графа позволяет находить решения меньшей стоимости, однако усложняет задачу его отыскания, т.к. увеличивают количество возможных действий агентов и повышает коэффициент ветвления. Значение радиуса агентов r было установлено равным $\sqrt{2}/4$.

5.1 Исследование алгоритма на графах регулярной структуры

В первой серии экспериментов проводилось тестирование оригинального алгоритма CCBS на графах регулярной декомпозиции с различной связностью. Для этого использовались следующие четыре значения коэффициента связности: $k = 2, 3, 4, 5$.

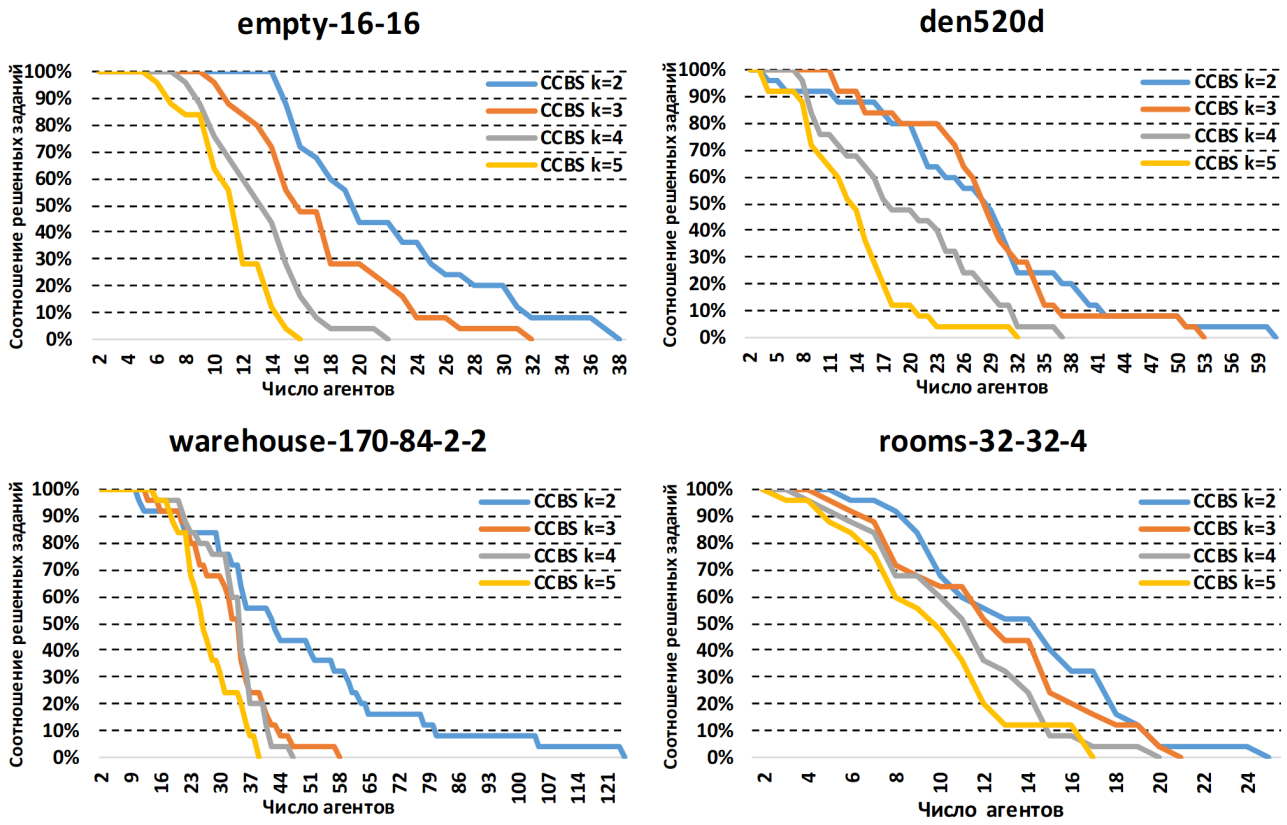


Рисунок 5.3 — Соотношение успешно решенных заданий к их общему количеству в зависимости от тестируемой карты и используемого коэффициента связности графа.

Для оценки эффективности работы алгоритма оценивалось соотношение успешно решенных заданий к их общему количеству. Детальные графики, показывающие соотношение числа решенных задний на каждой из карт в зависимости от числа агентов в заданиях, а также связности графа, показаны на Рисунке 5.3. Полученные результаты показывают, что увеличение связности графа снижает эффективность работы алгоритма, т.к. сами задания, т.е. расположение статических препятствий, а также стартовые и целевые положения агентов остаются неизменными вне зависимости от используемой связности графа. Снижение числа успешно решаемых заданий объясняется тем, что, во-первых, алгоритму нижнего уровня, т.е. алгоритму CSIPP, требуется больше времени на поиск индивидуальных траекторий агентов, во-вторых, наличие большего числа альтернативных действий приводит к тому, что между одной и той же парой агентов возникает большее число конфликтов, в результате чего алгоритм вынужден накладывать большее число ограничений для нахождения бесконфликтного решения.

	k=2	k=3	k=4	k=5
empty-16-16	583	454	354	254
den520d	866	754	537	292
warehouse-170-84-2-2	1355	1135	1038	819
rooms-32-32-4	360	339	273	217

Таблица 1 — Общее число заданий, решенных алгоритмом CCBS на каждой из протестированных карт в зависимости от используемого коэффициента связности.

Несмотря на то, что в большинстве случаев алгоритм смог решить наибольшее число заданий при использовании наиболее низкой связности графа ($k = 2$), в ряде случаев алгоритм смог решить большее число заданий при использовании значения коэффициента $k = 3$. Такое поведение может объясняться тем, что в случае использования 4-связной модели агенты имеют большее число различных альтернативных траекторий, которые являются “симметричными”. Эти траектории обладают идентичной стоимостью и отличаются лишь моментами времени, когда агент меняет направление движения. Наличие большого числа подобных траекторий может привести к необходимости рассмотрения большого числа альтернативных решений идентичной стоимости, каждое из которых содержит в себе конфликт между одной и той же парой агентов. Более высокая степень связности графа снижает количество “симметричных” траекторий, что положительно сказывается на эффективности работы алгоритма и в ряде случае позволяет компенсировать возросшую сложность поиска решений и решить ряд сценариев с бОльшим числом агентов.

В Таблице 1 представлены значения общего числа решенных заданий алгоритмом CCBS на каждой из протестированных карт в зависимости от используемой связности графа. Они подтверждают сделанные ранее выводы о снижении эффективности работы алгоритма и позволяют численно оценить разницу.

Необходимо отметить, что полученные значения общего числа решенных заданий напрямую зависят от времени работы, которое выделено алгоритму на поиск решения. Как уже отмечалось ранее, все экспериментальные исследования проводились с ограничением в 30 секунд. Для оценки влияния этого параметра на эффективность работы алгоритма дополнительно были проведено тестирование алгоритма CCBS с ограничением в 300 секунд. Также был проведен анализ, сколько времени потребовалось алгоритму на отыскание каждого решения. Результаты

этого анализа приведены на Рисунке 5.4. Представленные данные были агрегированы по всем 4-м ГРД. Каждый столбец показывает суммарное число заданий, каждое из которых было решено за время, не превышающее соответствующий лимит. Так, к примеру, при $k = 5$ и лимите в 1 секунду алгоритм суммарно решает 1187 заданий, в то время как при лимите в 5 секунд – 1289. Разница между этими двумя значениями показывает число заданий, которые были решены алгоритмом CCBS за время более 1 секунды, но не превышающее 5 секунд. Используемый принцип генерации заданий из сценариев и их постоянное постепенное усложнение приводят к тому, что большинство заданий, содержащих малое количество агентов, алгоритм CCBS способен решить за доли секунды. Увеличение лимита доступного времени позволяет повысить количество решаемых заданий, однако, даже в случае использования лимита времени в 300 секунд, разница в числе решаемых заданий не достигает кратных значений в сравнении с лимитами времени, которые на 1-2 порядка ниже. Такое поведение, с одной стороны, объясняется тем, что реализация алгоритма была написана на языке C++ с использованием оптимизированных структур хранения данных, что позволяет быстро решать задания, которые содержат малое количество конфликтов между агентами. С другой стороны, добавление дополнительного агента в задание, который конфликтует сразу с несколькими агентами, зачастую приводит к возникновению большого числа альтернативных решений, на рассмотрение которых алгоритму требуется слишком много времени.

Помимо оценки эффективности работы алгоритма CCBS, была также проведена оценка качества отыскиваемых решений. В рассматриваемой постановке задачи в качестве критерия качества решения используется суммарная стоимость всех траекторий, составляющих решение: $cost(\Pi) = \sum_{i=1}^{K} cost(\pi_i)$. Очевидно, что с ростом числа агентов будет также возрастать стоимость решений. Поэтому, для оценки качества решений использовались относительные значения. Все решения, найденные алгоритмом CCBS, были нормированы относительно соответствующего решения, найденного алгоритмом CCBS при $k = 2$. Данная оценка показывает, насколько сильно может сократиться стоимость решения при использовании графа регулярной декомпозиции, имеющего более высокую связность. Для подсчета этой оценки результаты были усреднены лишь по тем заданиям, которые были успешно решены алгоритмом CCBS для любой из протестированных связностей графа.

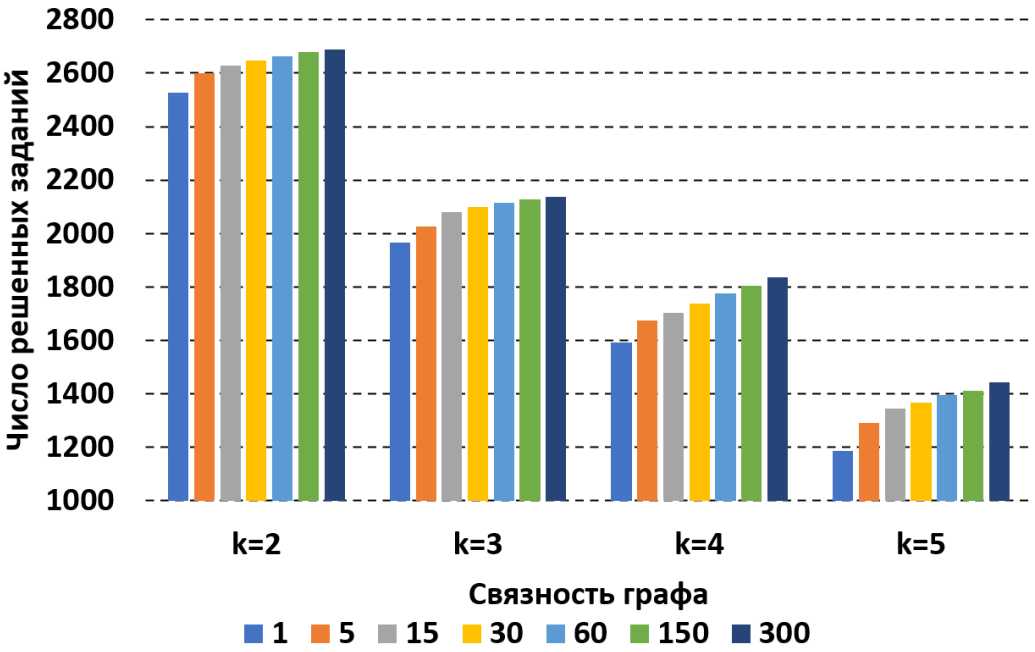


Рисунок 5.4 — Количество заданий решенных алгоритмом CCBS в зависимости от ограничения на время работы (в секундах).

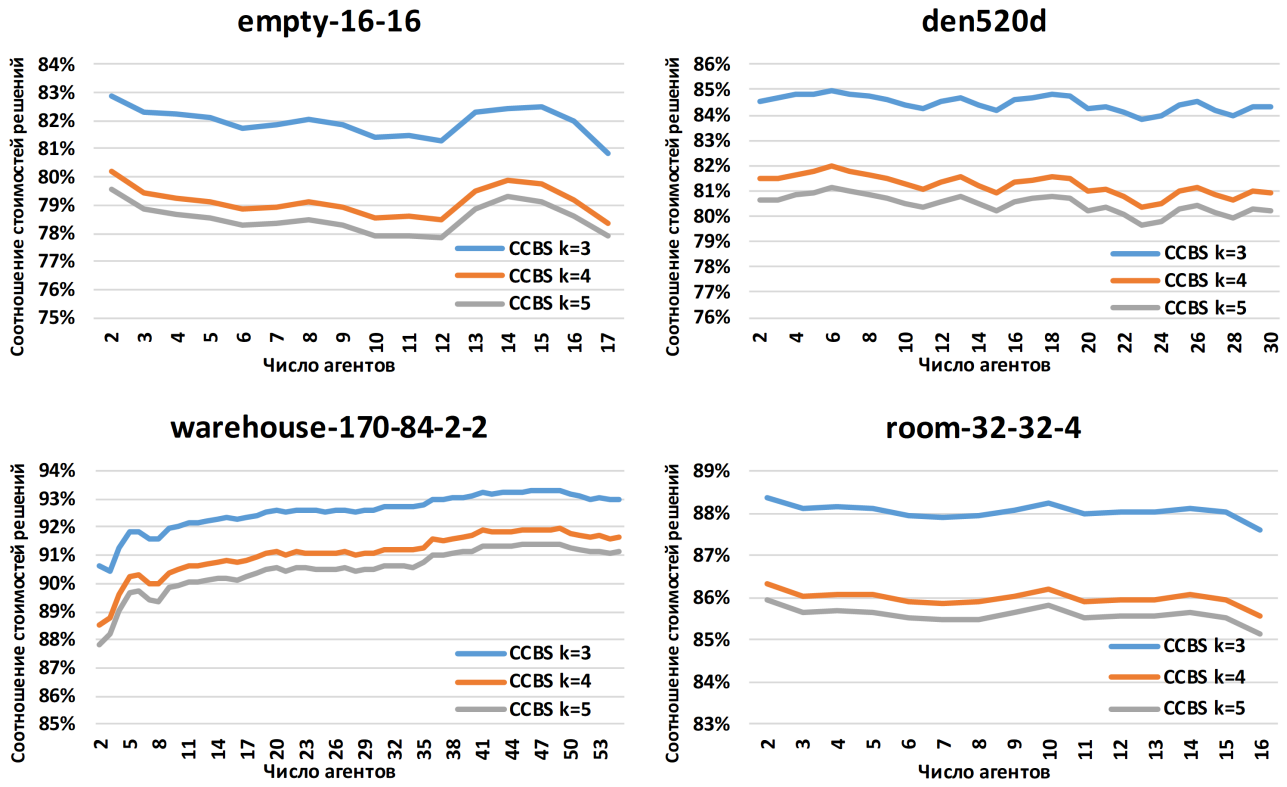


Рисунок 5.5 — Снижение стоимости решений, отыскиваемых алгоритмом CCBS, благодаря использованию ГРД с более высокой степенью связности, в зависимости от числа агентов.

Результаты, представленные на Рисунке 5.5, показывают, что снижение стоимости отыскиваемых решений почти не имеет зависимости от числа агентов в заданиях, однако, сильно зависит от карты, на которой проводилось тестирование. Так, наибольшее снижение стоимости решений (более 20%) удается достичь лишь на карте `empty-16-16` при использовании связности ГРД с $k = 4$ и $k = 5$. Это объясняется тем, что на карте `empty-16-16` отсутствуют статические препятствия, благодаря чему агенты могут свободно двигаться в любом доступном направлении. Наличие препятствий на других картах ограничивает агентов в выборе действий и вынуждает их двигаться вдоль препятствий, в результате чего агенты выполняют одни и те же действия как на 4-связном графе, так и на 32-связном. Особенно ярко это выражается на карте `warehouse-170-84-2-2`, где имеется большое количество прямоугольных препятствий и отсутствие больших открытых пространств, в которых агенты могли бы свободно двигаться во всех доступных им направлениях.

Стоит также заметить низкую разницу в стоимостях решений между $k = 4$ и $k = 5$. В большинстве случаев разница составляет менее 1%, что говорит о том, что дальнейшее увеличение связности графа не позволит существенно повысить качество решений в сравнении с протестированными значениями, такими как $k = 4$ и $k = 5$. Это объясняется тем, что наибольший выигрыш от добавления дополнительных направлений движений достигается при переходе от 4-связного ГРД к 8-связному, т.к. добавление диагональных переходов позволяет сократить стоимость перехода в диагонально смежную вершину графа с 2 до $\sqrt{2}$. Дальнейшее увеличение связности графа также, очевидно, позволяет достигать некоторых вершин за меньшее количество времени, однако в процентном соотношении это снижение будет меньше.

Кроме того, был также проведен сравнительный анализ стоимости решений, отыскиваемых алгоритмом CCBS с коэффициентом $k = 2$, с решениями, найденными стандартным алгоритмом CBS, использующим допущение о дискретности времени, который, как следствие, допускает совершение действий ожидения только продолжительностью равной 1. Получить снижение стоимости за счет действий перемещений в данном случае невозможно, т.к. оба алгоритма оперируют идентичным набором возможных действий перемещений. Единственным возможным вариантом снижения стоимости является использование действий ожиданий, имеющих продолжительность некратную 1. Возможность совершить действие-ожидание подобной продолжительности зависит не только от взаимного

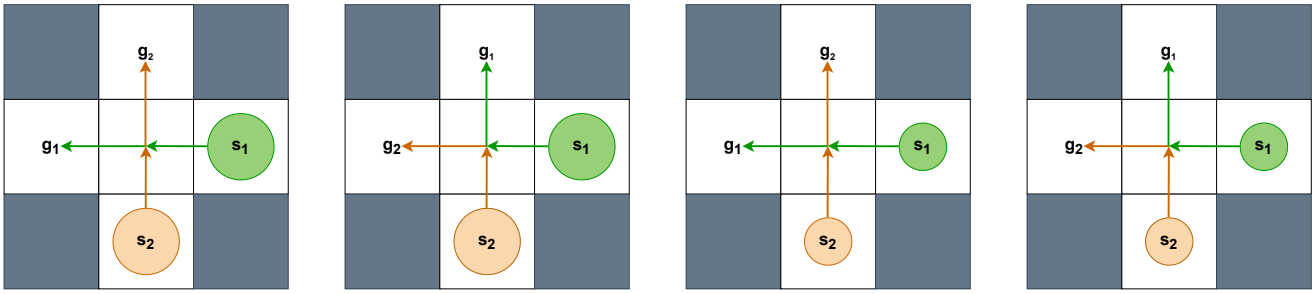


Рисунок 5.6 — Примеры сценариев, в которых для нахождения бесконфликтного решения один из агентов должен совершить действие ожидания, продолжительность которого зависит от радиусов безопасности и направлений движений агентов.

	$r = \sqrt{2}/4$	r = 0.25
empty-16-16	$3(1.5\%)$	$197(85.6\%)$
den520d	$13(9\%)$	$35(22.5\%)$
warehouse-170-84-2-2	$0(0\%)$	$98(40.2\%)$
rooms-32-32-4	$95(39.6\%)$	$236(79.2\%)$

Таблица 2 — Количество (и доля) заданий, в которых решения, найденные алгоритмом CCBS, имеют стоимость ниже чем у решений, найденных с помощью CBS.

расположения агентов, но и от их радиусов безопасности. На Рисунке 5.6 показаны 4 примера, в которых один из агентов должен совершить действие ожидания для нахождения бесконфликтного решения. В случае, если агенты имеют радиус $\sqrt{2}/4$ (случаи а) и б)) продолжительность действия ожидания равна либо 1, либо $\sqrt{2}/2$ в зависимости от направления движения агентов. Если же агенты имеют радиус 0.25, то для устранения конфликта достаточно совершить ожидание продолжительностью $\sqrt{2}/2$ и 0.5 соответственно. Уменьшение продолжительности действия ожидания в случаях б) и г) объясняется тем, что после того, как один из агентов достигает центральную вершину, оба агента движутся однонаправленно.

В Таблице 2 показано количество заданий и их доля, в которых решения, найденные алгоритмом CCBS, имеют стоимость ниже чем у решений, найденных с помощью CBS. При подсчете доли заданий из общей базы были исключены задания, стоимость решений которых невозможно улучшить, т.к. она совпадает со стоимостью начального частичного решения. Наибольшая доля заданий, на которых CCBS смог найти решения меньшей стоимости, была получена на карте

`room-32-32-4`. Такой результат объясняется тем, что ситуации, в которых агенты вынуждены ждать, возникают в тех случаях, когда агент не может построить аналогичную траекторию идентичной стоимости. Наиболее часто это происходит тогда, когда путь между стартовым и целевым положением проходит через узкое место, не имеющее альтернатив. Подобные узкие проходы в большом количестве присутствуют на карте `room-32-32-4`. Использование значения 0.25 для радиуса безопасности позволяет существенно повысить количество и долю заданий, в которых CCBS находит решение меньшей стоимости в сравнении с CBS. Объясняется этом тем, что при использовании радиуса $\sqrt{2}/4$ алгоритм CCBS не может уменьшить продолжительность действия ожидания в ряде случаев (см. Рисунок 5.6а). Что касается разницы в стоимости решений, отыскиваемых алгоритмами CCBS и CBS, то, ожидаемо, она составляет менее 1% от общей стоимости решения, т.к. основной вклад в стоимость решений вносят стоимости действий перемещений.

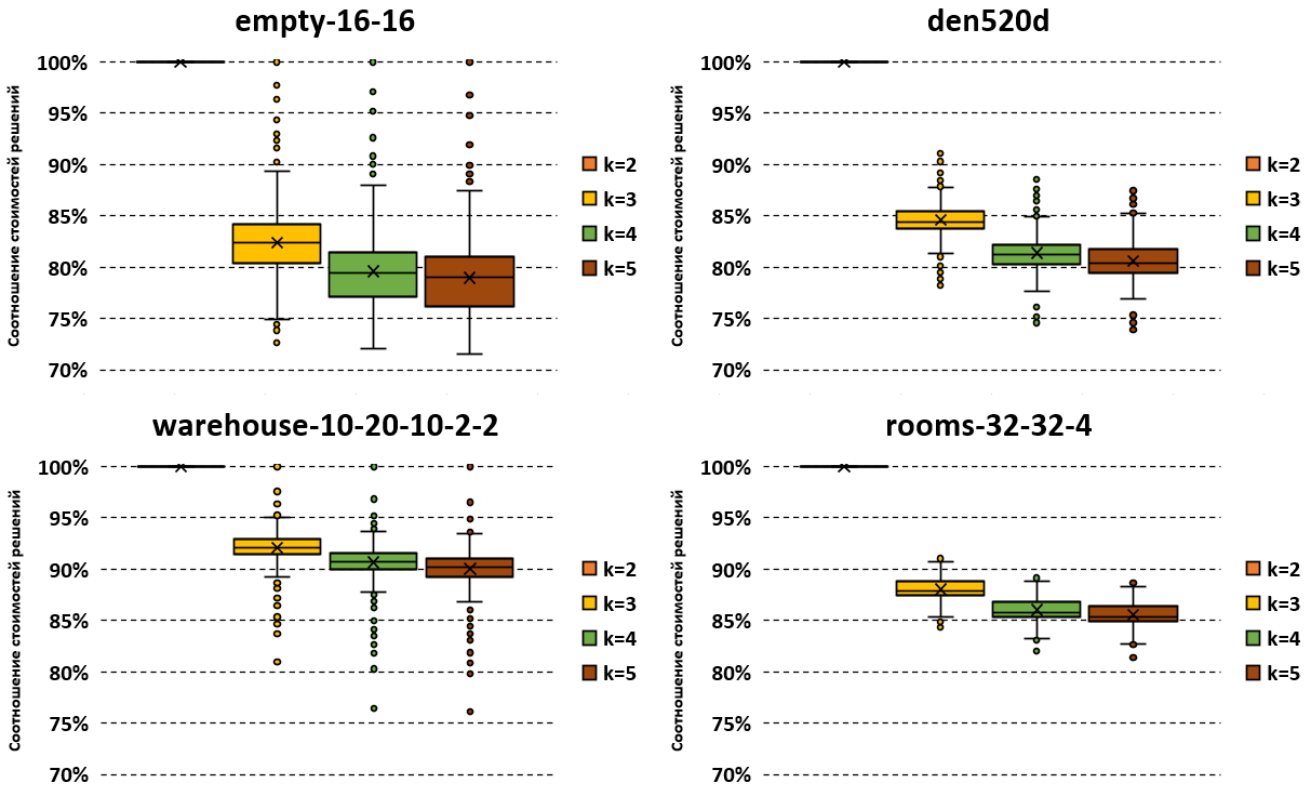


Рисунок 5.7 — Снижение стоимости решений, отыскиваемых алгоритмом CCBS, относительно $k = 2$.

На Рисунке 5.7 изображены диаграммы размаха, показывающие разброс снижения стоимости решения в зависимости от связности ГРД относительно 4-связного ГРД. Наибольший разброс наблюдается на карте `empty-16-16`. Это

объясняется тем, что в зависимости от расположения стартовых/целевых положений агентов, агенты могут либо сильно сократить стоимость решения за счет возможности движения по диагонали, либо же большинство стартовых/целевых положений расположены таким образом, что агентам не требуется совершение диагональных перемещений для скорейшего достижения целевых положений. Этот факт подтверждается наличием одного задания, в котором стоимость решения совпадает с $k = 2$. Аналогичное задание есть и на карте `warehouse-170-84-2-2`, где, как уже ранее отмечалось, ввиду структуры карты, в целом реже чем на других картах требуются переходы в направлениях, отличных от тех, что доступны при $k = 2$.

Первая серия экспериментов позволила оценить эффективность работы и поведение алгоритма CCBS на картах с разной топологией и графах разной связности, а также позволила оценить какого снижения стоимости решений можно достичь с увеличением связности ГРД. В зависимости от топологии карты снижение стоимости решений при наивысшем протестированном коэффициенте связности $k = 5$ в сравнении с $k = 2$ составляет в среднем от 10% до 21%, а в некоторых отдельных случаях эта разница может достигать значения в 28%.

5.2 Исследование оптимальных модификаций

Данный раздел посвящен изучению предложенных модификаций, позволяющих повысить вычислительную эффективность алгоритма и сохранить при этом свойства оптимальности. Напомню, что в работе были предложены следующие 3 модификации – приоритизация конфликтов (Prioritizing Conflicts, PC), эвристические функции верхнего уровня (High-Level Heurisitcs, HL), непересекающееся разделение (Disjoint Splitting, DS). Эти модификации могут быть скомбинированы вместе. Так образом, всего было протестировано 5 различных версий алгоритма – CCBS, CCBS+PC, CCBS+DS, CCBS+DS+PC, CCBS+DS+PC+H. Можно заметить отсутствие версии, использующей только HL модификацию. Это объясняется тем, что для работы этой модификации требуется вычислять добавочную стоимость конфликтов, что приводит к необходимости использования модификации приоритизации конфликтов, с помощью которой и рассчитываются добавочные стоимости.

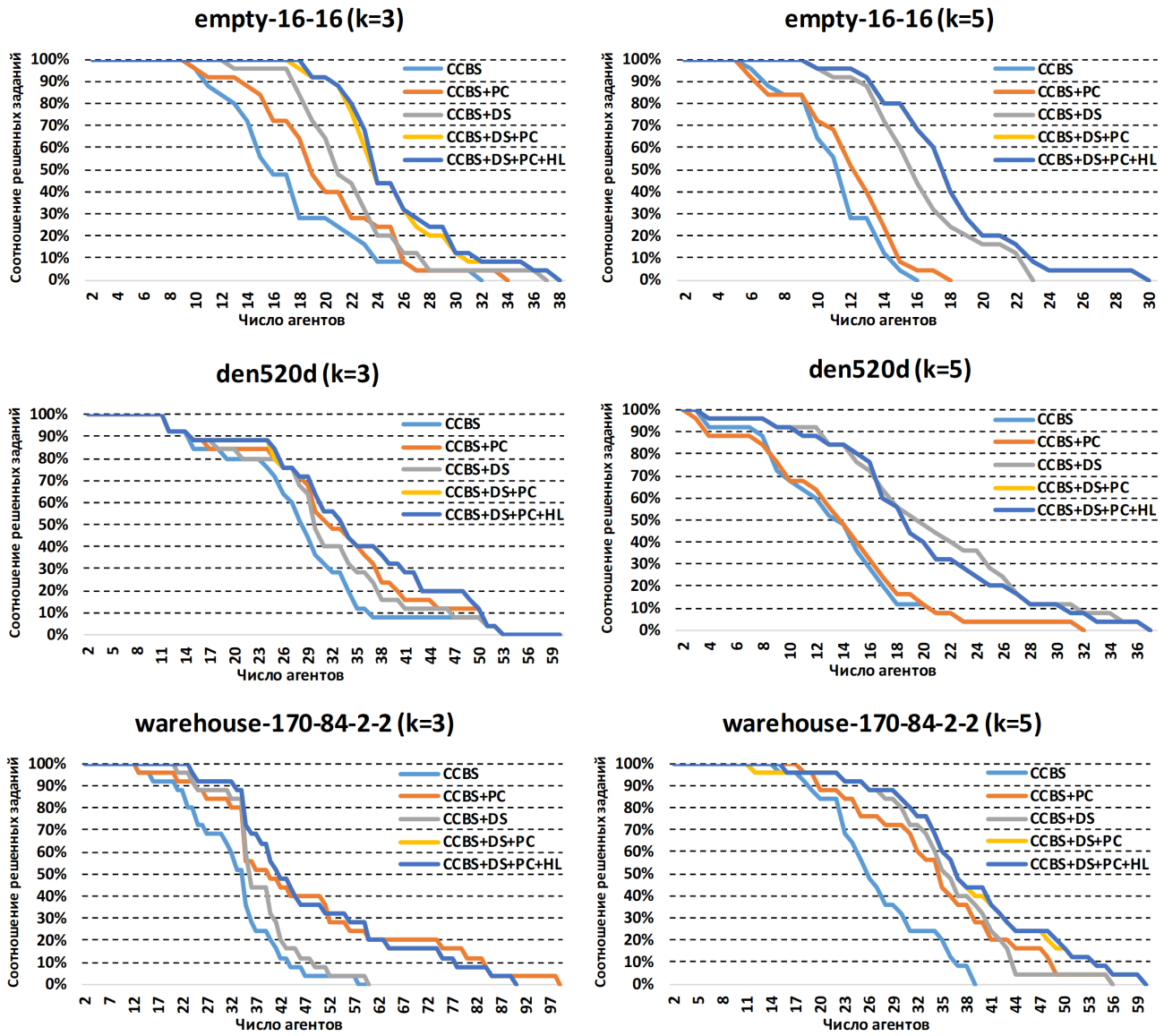


Рисунок 5.8 — Сравнение различных модификаций алгоритма CCBS.

Экспериментальное исследование проводилось на картах `den520d`, `empty-16-16` и `warehouse-170-84-2-2` с 2 значениями коэффициента связности – $k = 3$ и $k = 5$.

На Рисунке 5.8 представлены результаты тестирования различных модификаций алгоритма CCBS на всех протестированных картах, представленных в виде ГРД. Данные графики демонстрируют соотношение числа решенных заданий на каждой из карт в зависимости от числа агентов, содержащихся в заданиях. Агрегированные данные, а именно суммарное число решенных заданий каждой из модификаций в зависимости от карты, представлены в Таблице 3. Полученные результаты показывают, что в большинстве случаев наибольшее число заданий решает версия, комбинирующая в себе все улучшения. Однако, на карте `den520d` с коэффициентом связности $k = 5$ наибольшее число заданий решает версия, ко-

	CCBS	+PC	+DS	+PC+DS	+PC+DS+HL
empty-16-16 k=3	390	455	506	587	**595**
den520d k=3	667	766	729	809	**810**
warehouse-170-84-2-2 k=3	773	1135	901	**1163**	**1163**
empty-16-16 k=5	236	254	366	**406**	**406**
den520d k=5	298	301	**472**	451	451
warehouse-170-84-2-2 k=5	637	821	846	917	**925**

Таблица 3 — Суммарное число заданий, решенных каждой из модификаций в зависимости от протестированной карты и коэффициента связности.

торая использует только непересекающееся разделение. Такой результат можно объяснить двумя факторами. Во-первых, высокая связность графа приводит к тому, что непересекающееся разделение дает более существенный прирост по числу решаемых заданий в сравнении с оригинальной версией алгоритма, т.к. позволяет сократить перебор благодаря использованию положительных ограничений. Во-вторых, карта `den520d` имеет большой размер и при высокой степени связности для расчета добавочной стоимости конфликтов алгоритму требуется больше времени. Что касается использования эвристики верхнего уровня, то добавление этой модификации не дает существенного прироста с точки зрения числа решаемых заданий, однако эта модификация позволяет раскрывать меньшее число вершин верхнего уровня, способствуя ускорению работы алгоритма.

Для анализа вычислительной эффективности различных модификаций алгоритма CCBS было посчитано среднее число CT вершин, раскрываемых алгоритмом в процессе работы, т.е. фактически среднее число итераций верхнего уровня, а также среднее время работы. Не смотря на то, что первый критерий не зависит от нюансов конкретной реализации алгоритма и вычислителя, на котором производилось тестирование, он не позволяет в полной мере оценить разницу в вычислительной эффективности алгоритмов. Объясняется это тем, что предложенные модификации дополняют компоненты алгоритма CCBS, что требует дополнительных вычислительных ресурсов. Так, например, при использовании модификации PC, алгоритму требуется вычислять добавочные стоимости конфликтов, что увеличивает число планирований нижнего уровня на каждой итерации работы алгоритма. При использовании модификации DS требуется использование алгоритма GSIPP для планирования индивидуальных траекторий с промежуточными целями, что не способствует ускорению работы алгоритма. Мо-

дификация HL также требует дополнительных вычислений, связанных либо с решением задачи линейного программирования, либо с процедурой выбора подмножества независимых конфликтов. Для расчета обоих этих критериев были использованы только те задания, которые были успешно решены всеми модификациями алгоритма CCBS. При этом из усреднения были исключены те задания, в которых начальное решение, полученное путем независимого планирования траекторий каждого из агентов, не содержит в себе ни одного конфликта. Исключение подобных заданий объясняется тем, что на них невозможно увидеть разницу между модификациями алгоритма CCBS.

	CCBS	+PC	+DS	+PC+DS	+PC+DS+HL
empty-16-16 k=3	3498.6	1026.6	83.4	42.0	**33.8**
den520d k=3	192.5	47.3	71.0	42.2	**37.5**
warehouse-170-84-2-2 k=3	402.0	287.2	46.0	25.3	**21.7**
empty-16-16 k=5	2576.6	931.7	55.4	37.5	**31.6**
den520d k=5	65.8	70.2	23.8	24.4	**21.5**
warehouse-170-84-2-2 k=5	376.4	**44.4**	130.2	114.2	80.8

Таблица 4 — Среднее число раскрытых CT вершин каждой из модификаций в зависимости от протестированной карты и коэффициента связности.

В Таблице 4 приведены средние значения числа раскрытых CT вершин. Чем меньше значение, тем меньшее количество итераций требуется алгоритму для нахождения решения, не содержащего конфликтов. Как и в случае с показателем числа успешно решаемых заданий, лучшие результаты в 5 из 6 случаев демонстрирует версия, комбинирующая в себе все предложенные улучшения. Более того, в некоторых случаях, таких как, например, на карте `empty-16-16` разница с среднем числе итераций достигает 2х порядков. Столь большая разница на карте `empty-16-16` и её отсутствие на других картах объясняется тем, что на этой небольшой карте алгоритм CCBS способен сгенерировать десятки и даже сотни тысяч альтернативных решений в течении доступного лимита времени, т.к. на планирование индивидуальных траекторий алгоритму требуется чрезвычайно малое количество времени. На других протестированных картах, в частности на карте `den520d`, алгоритму требуется намного больше времени на каждую итерацию, т.к. в этом случае на планирование индивидуальных траекторий требуется значительно большее количество времени. Не смотря на это, предложенные мо-

дификации позволяют кратно сократить число итераций, требуемых алгоритму CCBS для отыскания решения.

	CCBS	+PC	+DS	+PC+DS	+PC+DS+HL
empty-16-16 k=3	0.577	0.502	0.011	0.010	**0.009**
den520d k=3	1.109	0.473	0.272	0.211	**0.182**
warehouse-170-84-2-2 k=3	0.657	0.398	0.114	**0.083**	**0.083**
empty-16-16 k=5	0.596	0.423	**0.013**	0.017	0.015
den520d k=5	2.637	6.568	**1.322**	3.087	3.262
warehouse-170-84-2-2 k=5	1.249	0.325	**0.266**	0.381	0.383

Таблица 5 — Среднее время работы (в секундах) каждой из модификаций в зависимости от протестированной карты и коэффициента связности.

В Таблице 5 приведены средние значения времени работы алгоритма (в секундах). В целом, эти данные подтверждают те выводы, сделанные на основе результатов, представленных в Таблице 4, однако, имеют ряд отличий и подтверждают тот факт, что на выполнение одной итерации каждой из модификаций требуется разное количество времени. Наиболее “затратной” с точки зрения вычислительных ресурсов оказалась модификация PC, которая повышает среднее время, требуемое на выполнение каждой итерации, примерно в два раза. При этом модификация DS в среднем обеспечивает одинаковое снижение как по времени работы, так и по числу итераций. В итоге, на ГРД с $k = 5$ лучший результат по времени работы демонстрирует именно та модификация, которая использует только непересекающееся разделение. В среднем, больше всего времени на каждую итерацию, ожидаемо, затрачивает модификация CCBS+PC+DS+HL. В результате, несмотря на существенно меньшее число раскрываемых CT вершин, модификации CCBS+PC+DS+HL и CCBS+PC+DS демонстрируют достаточно близкие значения по среднему времени работы.

5.3 Исследование субоптимальных модификаций

В данном разделе приводятся результаты экспериментальных исследований субоптимальных модификаций алгоритма – CCBS+EES и CCBS+FOCAL. Данные

алгоритмы были протестированы с различными коэффициентами субоптимальности на 4-х ГРД с коэффициентом связности $k = 3$.

Стоит заметить, что для разных графов использовались различные наборы коэффициентов субоптимальности. Так, для `empty-16-16` и `rooms-32-32-4` использовались значения 1.01, 1.03, 1.05, 1.1 и 1.25, для `den520d` и `warehouse-170-84-2-2` использовались значения 1.001, 1.003, 1.005, 1.01, 1.03, а для `Sparse`, `Dense` и `Super-Dense` – 1.001, 1.003, 1.005, 1.01, 1.03, 1.05, 1.1, 1.25. Использование различных коэффициентов объясняется различными размерами ГРД. Карты `den520d` и `warehouse-170-84-2-2` имеют большие размеры, в связи с чем устранение конфликтов между траекториями агентов не вносит существенного вклада в общую стоимость решения. Поэтому дальнейшее увеличение коэффициента субоптимальности на этих картах не приносит положительного эффекта, т.к. все вершины дерева ограничений добавляются в список FOCAL уже при значении фактора субоптимальности 1.03. Напротив, изменение траекторий для устранения конфликтов на картах малого размера, т.е. `empty-16-16` и `rooms-32-32-4`, приводит к более значительному увеличению стоимости решения, что объясняет использование более высоких значений для коэффициента субоптимальности. Помимо этого, алгоритмы были протестированы с коэффициентом субоптимальности 1, т.е. фактически они действовали как алгоритм CCBS, отыскивающий оптимальные решения.

На Рисунке 5.9 представлены результаты тестирования на ГРД. Столбчатые диаграммы (слева) показывают общее число решенных заданий, а также их процентное соотношение относительно алгоритма CCBS, отыскивающего оптимальные решения, в зависимости от используемого значения коэффициента субоптимальности. На графиках, представленных в виде диаграммы размаха, показана фактическая разница в стоимости решения между оптимальным алгоритмом CCBS и его субоптимальными модификациями. Полученные результаты показывают, что обе субоптимальные версии решают в 2 и более раза большее число заданий, что говорит о том, что они способны находить решения для заданий содержащих, как минимум, вдвое большее число агентов. Количество решенных заданий резко возрастает даже с минимальным значением коэффициента субоптимальности, который превышает 1. При этом нельзя однозначно сказать, какая из версий показывает лучшие результаты. CCBS+EES однозначно проявляет себя лучше на карте `rooms-32-32-4`, в то время как на остальных картах лучшие результаты показывают разные алгоритмы в зависимости от используемо-

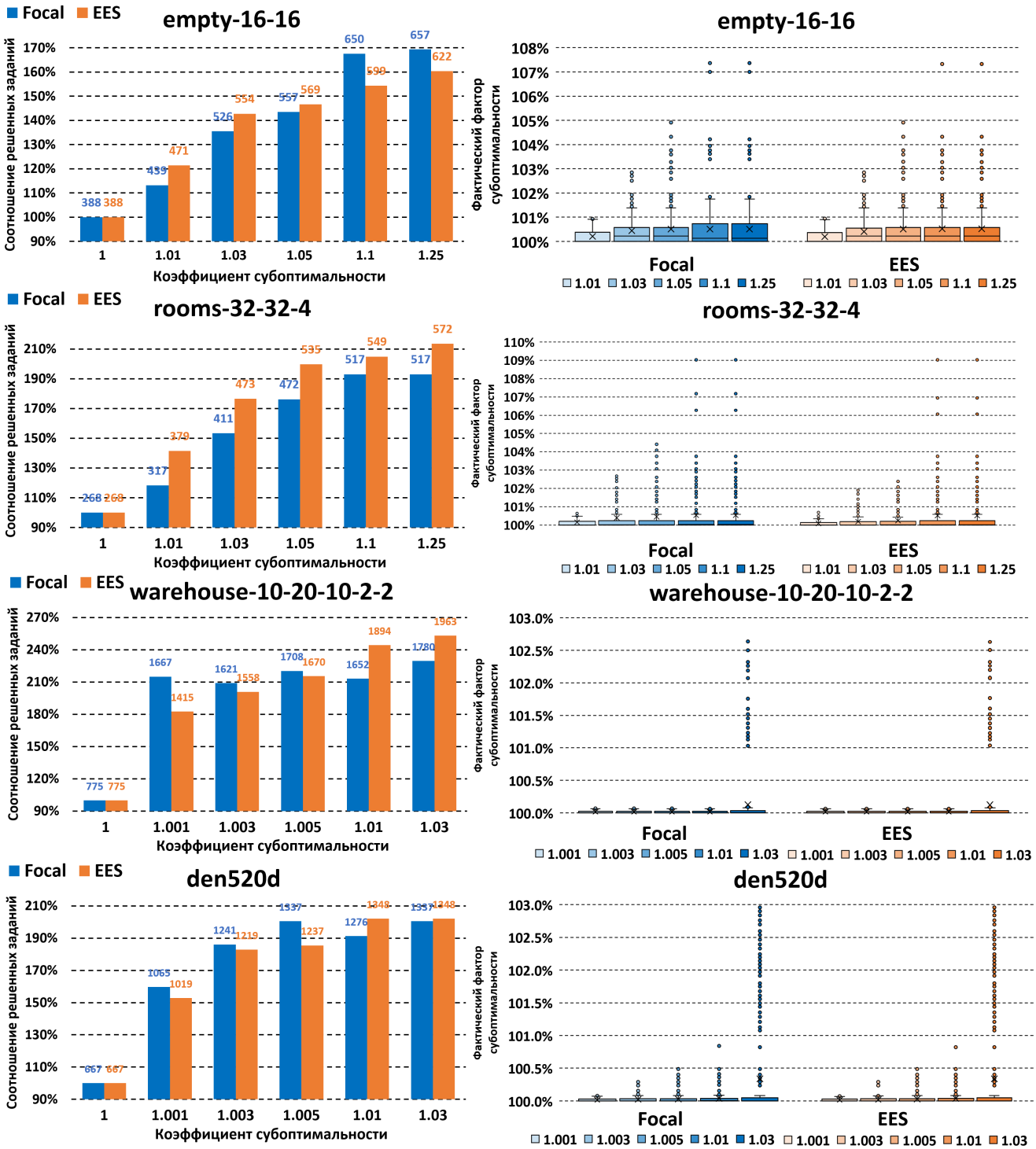


Рисунок 5.9 — Сравнение субоптимальных модификаций алгоритма CCBS на ГРД с $k = 3$ в зависимости от коэффициента субоптимальности.

го коэффициента субоптимальности. Стоит отметить, что дальнейшее увеличение коэффициента субоптимальности не приводит к дальнейшему улучшению результатов, т.к. уже при имеющихся максимальных значениях алгоритмы добавляют в список FOCAL все сгенерированные альтернативные решения.

С точки зрения качества решений, можно заметить, что в большинстве случаев решения, отыскиваемые обеими модификациями, близки к оптимальным вне зависимости от используемого значения коэффициента субоптимальности. При этом на графиках присутствует точки-выбросы, соответствующие отдельным заданиям, в которых алгоритмы все же находят решения, стоимость которых существенно отличается от стоимости оптимального решения. Однако, разница в стоимости никогда не превышает допустимый порог, что экспериментально подтверждает утверждения о том, что алгоритмы CCBS+EES и CCBS+Focal гарантируют нахождение ограниченно-субоптимальных решений. При этом, глядя на графики сравнения качества решений, может показаться, что обе модификации находят одинаковые решения. Такой эффект наблюдается в виду того, что для построения этих графиков использовались только те задания, которые были успешно решены всеми алгоритмами, в том числе алгоритмом CCBS (для получения значения стоимости оптимального решения).

	EES > Focal	EES < Focal	$\Delta_{average}$	Δ_{max}
warehouse w=1.001	11	**77**	0.024%	0.073%
den520d w=1.001	66	**100**	0.009%	0.034%
empty w=1.01	**73**	67	0.236%	0.778%
rooms w=1.01	14	**63**	0.244%	0.739%
warehouse w=1.01	227	**359**	0.084%	0.940%
den520d w=1.01	**192**	166	0.029%	0.959%
empty w=1.1	**139**	138	0.890%	6.374%
rooms w=1.1	45	**135**	1.016%	7.793%

Таблица 6 — Сравнение качества решений, отыскиваемых алгоритмами CCBS+EES и CCBS+Focal.

Для того, чтобы оценить разницу в поведении алгоритмов и сравнить качество отыскиваемых решений, были проанализированы все задания, в которых алгоритмы CCBS+EES и CCBS+Focal нашли решения различной стоимости. Результаты этого сравнения приведены в Таблице 6. Каждой строке соответствуют

данные, агрегированные по всем заданиям, успешно решенным обоими алгоритмами на указанной карте и с соответствующим значением коэффициента субоптимальности w. Значения в столбце “EES>Focal” показывают число заданий, в которых алгоритм CCBS+Focal нашел решение, стоимость которого меньше, чем стоимость решения, найденного алгоритмом CCBS+EES. Аналогично, значения в столбце “EES<Focal” показывают число заданий, в которых решение, найденное с помощью CCBS+EES, имеет стоимость меньше чем то, что было найдено алгоритмом CCBS+Focal. С точки зрения этого критерия чаще решение меньшей стоимости находит модификация CCBS+EES, однако, разница в стоимости, как правило несущественная, особенно в случае использования низких значений коэффициента субоптимальности. Столбец “$\Delta_{average}$” показывает среднюю разницу в стоимостях решений, а “Δ_{max}” – максимальную. Ожидаемо, наибольшая разница наблюдается при использовании значения 1.1, т.к. в остальных случаях алгоритмы ограничены более низким значением коэффициента субоптимальности и не способны найти решение, которые бы отличалось по стоимости более чем на 1% или 0.1% соответственно.

Проведенные экспериментальные исследования субоптимальных версий алгоритма CCBS – CCBS+EES и CCBS+Focal продемонстрировали существенное повышение эффективности работы алгоритма в сравнении как с оригинальным алгоритмом CCBS, так и с разработанными модификациями. При этом в большинстве случаев достаточно использовать небольшое значение коэффициента субоптимальности, например, 1.01, который позволяет гарантировать, что найденное алгоритмом решение не превысит стоимость оптимального решения более чем на 1%. Что касается сравнения этих двух версий, то они показывают достаточно близкие результаты и нельзя однозначно сказать, что при любых условиях, одна версия превосходит другую. Однако, при выборе конкретной версии, предпочтительнее использовать CCBS+EES, т.к. она в большинстве случаев показывает лучшие результаты как по числу решаемых заданий, так и по качеству отыскиваемых решений.

5.4 Планирование с учетом дополнительных ограничений

Предлагаемый в работе алгоритм CCBS способен учитывать действия произвольной продолжительности, что позволяет отказаться от допущения о дискретности времени в постановке задачи. Однако, используемая модель движения агента все еще имеет множество допущений и не учитывает многие из ограничений, накладываемые реальными робототехническими системами. Так, к примеру, считается, что агенты движутся с постоянной скоростью, и при этом могут мгновенно останавливаться или мгновенно менять направление движения. В данной серии экмпериментов будет показано, что предлагаемый подход является более универсальным и может быть применен к более сложной модели перемещения агентов, в которой необходимо учитывать направление движения, а также время, требуемое на его смену.

Для учета направления движения необходимо модифицировать идентификатор состояния, использующийся на нажнем уровне алгоритма, т.е. в алгоритме CSIPP. Стоит напомнить, что состояние описывается парой $\langle cfg, interval \rangle$, в которой cfg - конфигурация, описывающая положение агента в пространстве. В оригинальном алгоритме CCBS положение агента задается с помощью пары координат $\langle i, j \rangle$. Необходимо добавить в конфигурацию еще один компонент: θ – угол, задающий ориентацию агента, т.е. его текущее направление движения. Таким образом положение агента в пространстве теперь описывается с помощью тройки $\langle i, j, \theta \rangle$.

Однако, если ввести ограничение, что агент может двигаться только по направлению угла θ, то агент не сможет достичь своего целевого положения, т.к. у него нет ни одного действия, которое могло бы сменить направление движения. Поэтому, был введен дополнительный тип действий – поворот на месте. Для его осуществления агент должен находиться в центре одной из вершин графа. В отличие от действия ожидания, продолжительность которого может быть произвольной, продолжительность действия поворота зависит от угла, на который необходимо осуществить поворот, и скорости: $a_D = (\theta - \theta')/\omega$, где ω – скорость, с которой агент осуществляет вращение. Таким образом, прежде чем осуществить действие перемещение в направлении θ', агент должен предварительно сменить ориентацию до требуемого направления движения.

Добавление учета направления движения не меняет логику работы алгоритма CCBS на верхнем уровне, однако приводит к значительному увеличению числа возможных состояний на этапе планирования индивидуальных траекторий, что негативно сказывается на эффективности работы алгоритма. Для повышения эффективности работы алгоритма CSIPP с учетом направления движения был предложен механизм доминации, который позволяет отсекать часть состояний и тем самым снизить число состояний, рассматриваемых алгоритмом CSIPP в процессе планирования индивидуальных траекторий. Проверка на доминирование применятся в тех случаях, когда в список OPEN добавляется состояние s и при этом в этом списке уже содержится состояние s', которое соответсвует то му же безопасному интервалу и координатам вершины, однако, отличается значением угла θ. В таком случае, если выполняется неравенство $g(s')+(s.\theta-s'.\theta)/\omega \leqslant g(s)$, то состояние s' *доминирует* над состоянием s, т.к. любое состояние, которое агент может достичь из s, он может достичь из s' за то же самое или меньшее время. Поэтому, добавлять состояние s в список $OPEN$ бессмысленно. Эта же проверка осуществляется в обратном направлении и если s' доминирует над состоянием s, то состояние s удаляется из списка $OPEN$. В случае, если выполняется неравенство $(s.\theta-s'.\theta)/\omega > |g(s)-g(s')|$, то ни одно из состояний s, s' не доминирует над другим и они оба должны быть добавлены в список $OPEN$.

Данная модификация алгоритма, учитывающая направления движения агентов, а также время, требуемое на смену направления движения, была названа CCBS-kc. Для сравнения с CCBS-kc также были протестированы оригинальный алгоритм CCBS и алгоритм AA-SIPP(m) [100]. Алгоритм AA-SIPP(m) основан на подходе безопасно-интервального планирования, благодаря чему может оперировать действиями произвольной продолжительности, как и алгоритм CCBS. Однако, в отличие от алгоритма CCBS, для разрешения возможных конфликтов между траекториями агентов, в нем используется приоритизированный подход. Алгоритм AA-SIPP(m), как и любой другой алгоритм, использующий подход приоритизированного планирования, не гарантирует нахождение оптимального решения.

Экспериментальное исследование проводилось на ГРД `empty-16-16` со степенью связности $k = 3$. Ввиду того, что оригинальные сценарии из коллекции MovingAI для этой карты содержат в себе только координаты стартовых и целевых положений, для этого эксперимента были сгенерированы новые сценарии. Всего было сгенерировано 100 сценариев, каждый из которых содержит 100

агентов. Координаты и направление движения для каждого агента выбирались случайным образом, с ограничением на то, что стартовые/целевые положения для всех агентов являются уникальными. Как и в других сериях экспериментов, время работы каждого из алгоритмов было ограничено 30 секундами, а для параметра ω, влияющего на время, требуемое на смену направления движения, было выбрано значение 0.5, что соответствует повороту на 90 градусов за 1 у.е. времени.

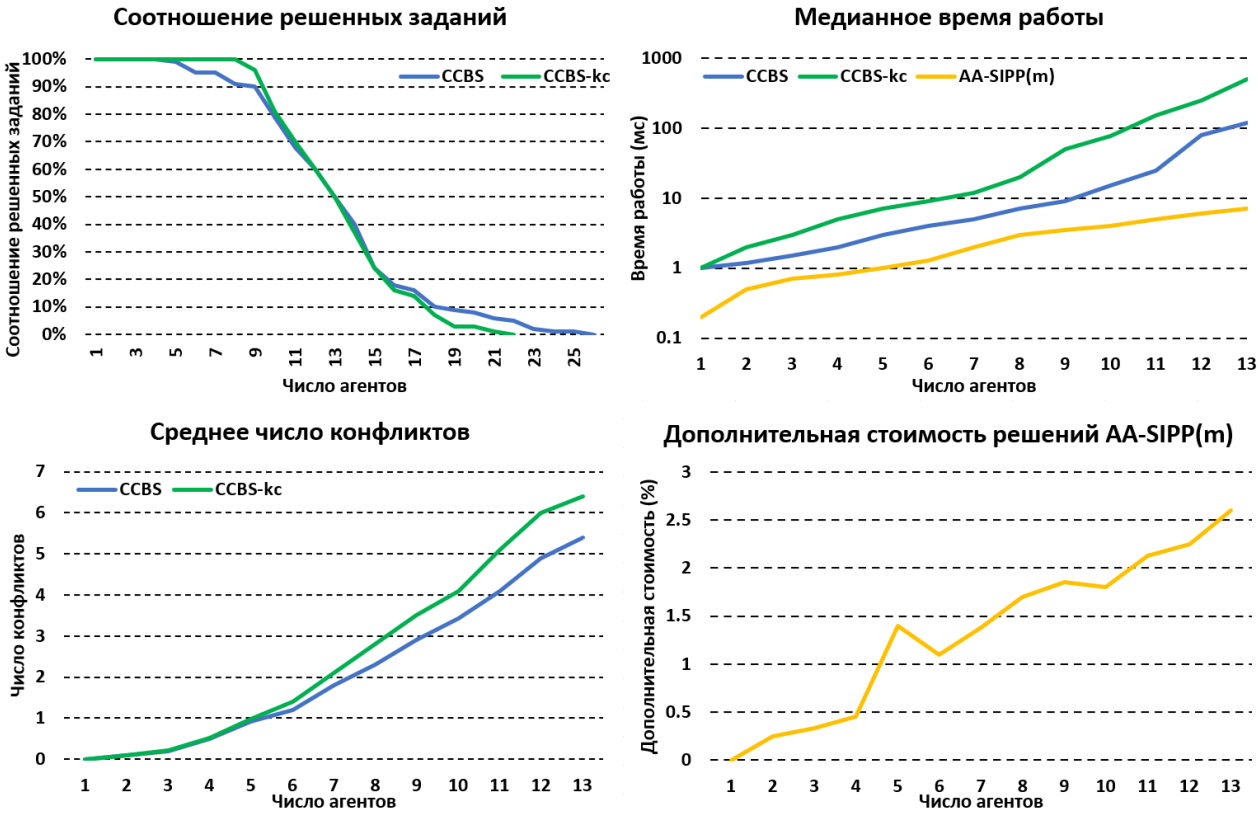


Рисунок 5.10 — Результаты тестирования алгоритма CCBS-kc на ГРД `empty-16-16` в сравнении с алгоритмами CCBS и AA-SIPP(m).

На Рисунке 5.10 представлены результаты тестирования алгоритма CCBS-kc в сравнении с CCBS и AA-SIPP(m). Левый верхний график показывает соотношение успешно решенных заданий. Отсутствие AA-SIPP(m) на этом графике объясняется тем, что этот алгоритм смог найти решение для 100% заданий, содержащих 26 и менее агентов. Как можно заметить, алгоритм CCBS-kc в случае малого количества агентов опережает оригинальный алгоритм CCBS, хотя работает в более сложной постановке задачи. Такое поведение можно объяснить тем, что дополнительное время, требуемое на смену направления движений, сокращает количество альтернативных траекторий, обладающих эквивалентной стоимостью. В результате чего алгоритму CCBS-kc в ряде случаев требуется рассмотреть меньшее число вершин верхнего уровня, которые имеют стоимость

меньшую, чем стоимость оптимального решения. Однако, из-за более медленной работы, в ряде случаев при более высоком количестве агентов в заданиях, алгоритм CCBS-kc не успевает решить часть заданий, которые решает оригинальный алгоритм CCBS. Этот вывод подтверждается верхним правым графиком, на котором показано медианное время, требуемое алгоритму на поиск решения. В данном случае использовались только те задания, которые были успешно решены всеми протестированными алгоритмами. На этом графике также можно наблюдать, что алгоритму AA-SIPP(m) требуется на порядок меньше времени для поиска решения в сравнении с алгоритмом CCBS-kc. На нижнем левом графике показано среднее число конфликтов, содержащихся в корневом узле дерева ограничений алгоритмов CCBS и CCBS-kc. Добавление учета направления движения приводит к тому, что при достаточно высоком числе агентов в заданиях, в корне дерева алгоритма CCBS-kc содержится в среднем на 1 конфликт больше, что дополнительно усложняет поиск бесконфликтного решения. Правый нижний график демонстрирует разницу в стоимостях решений, отыскиваемых алгоритмом AA-SIPP(m) в сравнении с оптимальными решениями, отыскиваемыми алгоритмом CCBS-kc. При малом количестве агентов разница не существенна и составляет менее 1%, однако, чем больше агентов содержится в задании, тем выше в среднем разница между стоимостями решений AA-SIPP(m) и CCBS-kc. Большее количество агентов приводит к большему числу конфликтов, которые алгоритм AA-SIPP(m) устраняет неоптимальным образом.

Данное экспериментальное исследование продемонстрировало возможность использования алгоритма CCBS и в других постановках задач, обладающих более сложной моделью движения агентов. При этом принцип работы алгоритма CCBS остался прежним, а все изменения, необходимые для учета дополнительных условий, потребовались лишь на нижнем уровне, отвечающем за планирование индивидуальных траекторий агентов.

5.5 Тестирование на графах нерегулярной структуры

Помимо экспериментов на графах регулярной декомпозиции, на которых проводилась основная часть экспериментальных исследований, было также проведено тестирование предложенного алгоритма на графах нерегулярной струк-

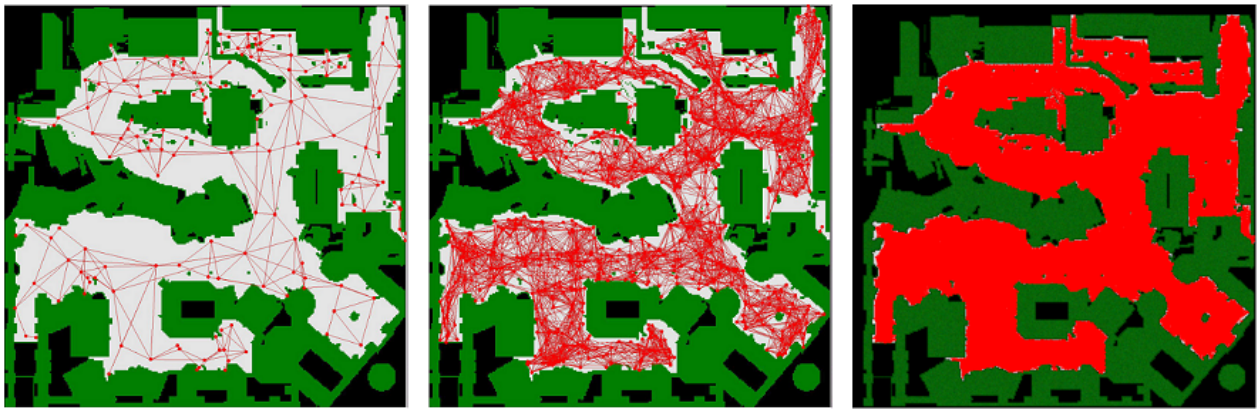


Рисунок 5.11 — Графическое представление ГНС, полученные из карты `den520d` с помощью алгоритма PRM. а) `Sparse`, б) `Dense`, в) `Super-Dense`.

туры (далее – ГНС). Ввиду отсутствия карт, представленных в виде ГНС в коллекции MovingAI, данные графы были сгенерированы самостоятельно. Для этого был использован алгоритм PRM (англ. Probabalistic Roadmap) [101], взятый из библиотеки OMPL(англ. Open Motion Planning Library) [102]. Ему на вход был подан ГРД `den520d` размером 256×257. Регулируя настройки алгоритма, были сгенерированы 3 графа различной плотности и связности:

– `Sparse` – содержит 158 вершин и 349 ребер.
– `Dense` – содержит 878 вершин и 7 341 ребер.
– `Super-Dense` – содержит 11 342 вершин и 263 533 ребер.

На Рисунке 5.11 показаны графические изображения всех трех сгенерированных графов.

Помимо генерации самих ГНС, для проведения тестирования на этих графах было также необходимо сгенерировать задания. По аналогии с картами, представленными в виде ГРД, для данного набора графов было сгенерировано по 25 различных сценариев, содержащих большое количество агентов (более 100). Стартовые и целевые положения для агентов выбирались случайным образом с условием отсутствия конфликтов между агентами, находящимися в стартовых/-целевых положениях.

При тестировании алгоритмов на ГНС не использовался какой-либо параметр, отвечающий за связность графа, т.к. все ребра в этих графах изначально заданы.

На Рисунке 5.12 показаны результаты тестирования алгоритма CCBS и его различных оптимальных модификаций в зависимости от графа. Агрегированные данные, т.е. суммарное число решенных зданий каждой из протестированных модификаций представлены в Таблице 7.

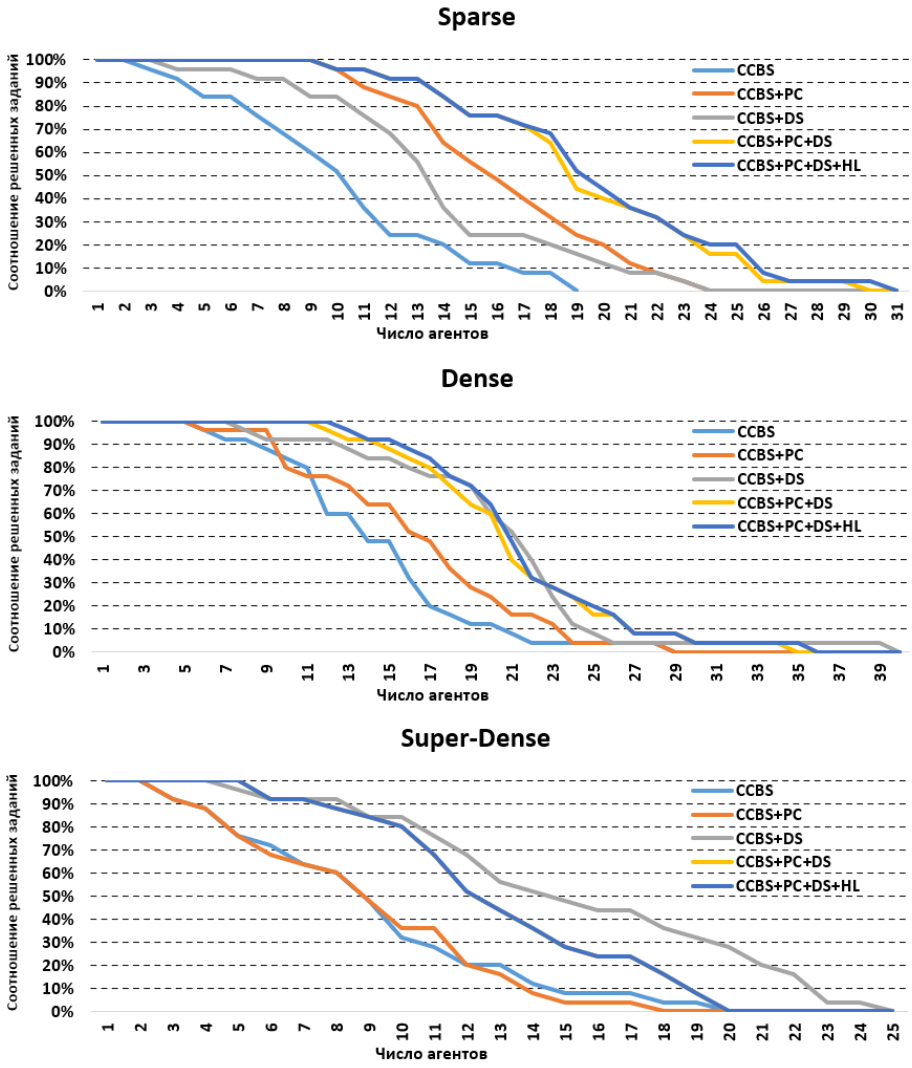


Рисунок 5.12 — Результаты сравнения различных модификаций алгоритма CCBS на ГНС.

Результаты этого теста подтвердили выводы, полученные на ГРД. В случаях, когда вершины имеют малое количество смежных вершин, как в случае с картой `Sparse`, непересекающееся разделение не демонстрирует существенного прироста эффективности, в отличие от процедуры приоритизации конфликтов. Однако, уже на карте `Dense`, где связность графа намного выше, непересекающееся разделение показывает результаты лучше чем модификация, использующая только приоритизацию конфликтов. В последнем случае, на карте `Super-Dense`, модификация, использующая только непересекающееся разделение демонстрирует результаты даже лучше чем версия, которая комбинирует в себе все улучшения. Такое поведение объясняется те ми же причинами, что были приведены для ГРД – высокая связность снижает эффективность от приоритизации конфликтов, т.к. снижаются их дополнительные стоимости и одновременно с этим повышается эффективность непересекающегося разделения, т.к. положительные ограниче-

	CCBS	+PC	+DS	+PC+DS	+PC+DS+HL
Sparse	239	389	329	468	**476**
Dense	344	392	494	507	**520**
Super-Dense	211	206	367	309	**309**

Таблица 7 — Суммарное число заданий, решенных каждой из модификаций на ГНС.

		CCBS	+PC	+DS	+PC+DS	+PC+DS+HL
итерации	Sparse	1499.3	133.4	67.7	32.0	**21.7**
	Dense	3969.1	432.4	83.4	50.3	**41.1**
	Super-Dense	302.0	323.0	28.1	33.9	**27.4**
время(с)	Sparse	0.204	0.025	0.008	0.006	**0.005**
	Dense	1.593	0.434	**0.033**	0.063	0.054
	Super-Dense	1.435	2.564	**0.213**	0.442	0.430

Таблица 8 — Среднее число итераций и время работы каждой из модификаций алгоритма CCBS на ГНС.

ния позволяют отсечь большее число различных альтернативных траекторий для агентов.

В Таблице 8 показаны среднее число итераций и время работы каждой из модификаций алгоритма CCBS в зависимости от протестированной карты. Полученные результаты демонстрируют те же тренды, что и на ГРД. С точки зрения числа итераций наилучшие результаты демонстрирует модификация CCBS+PC+DS+HL, однако, из-за того, что приоритизация конфликтов требует бОльшее количество времени на каждую итерацию, среднее время в 2 из 3 случаев показывает модификация, которая использует только непересекающееся разделение.

На Рисунке 5.13 показаны результаты тестирования алгоритмов CCBS+EES и CCBS+Focal на ГНС. Как и в случае с результатами тестирования на ГРД, нельзя однозначно сказать какая из предложенных модификаций показывает лучшие результаты. При малых значениях коэффициента субоптимальности лучше результаты показывает версия CCBS+ESS, при средних значениях – CCBS+Focal. При максимальном протестированном значение коэффициента субоптимальности обе версии показывают очень близкие результаты, а на ГНС `Super-Dense` – абсолютно идентичные. Более того, использование высоких значений коэффи-

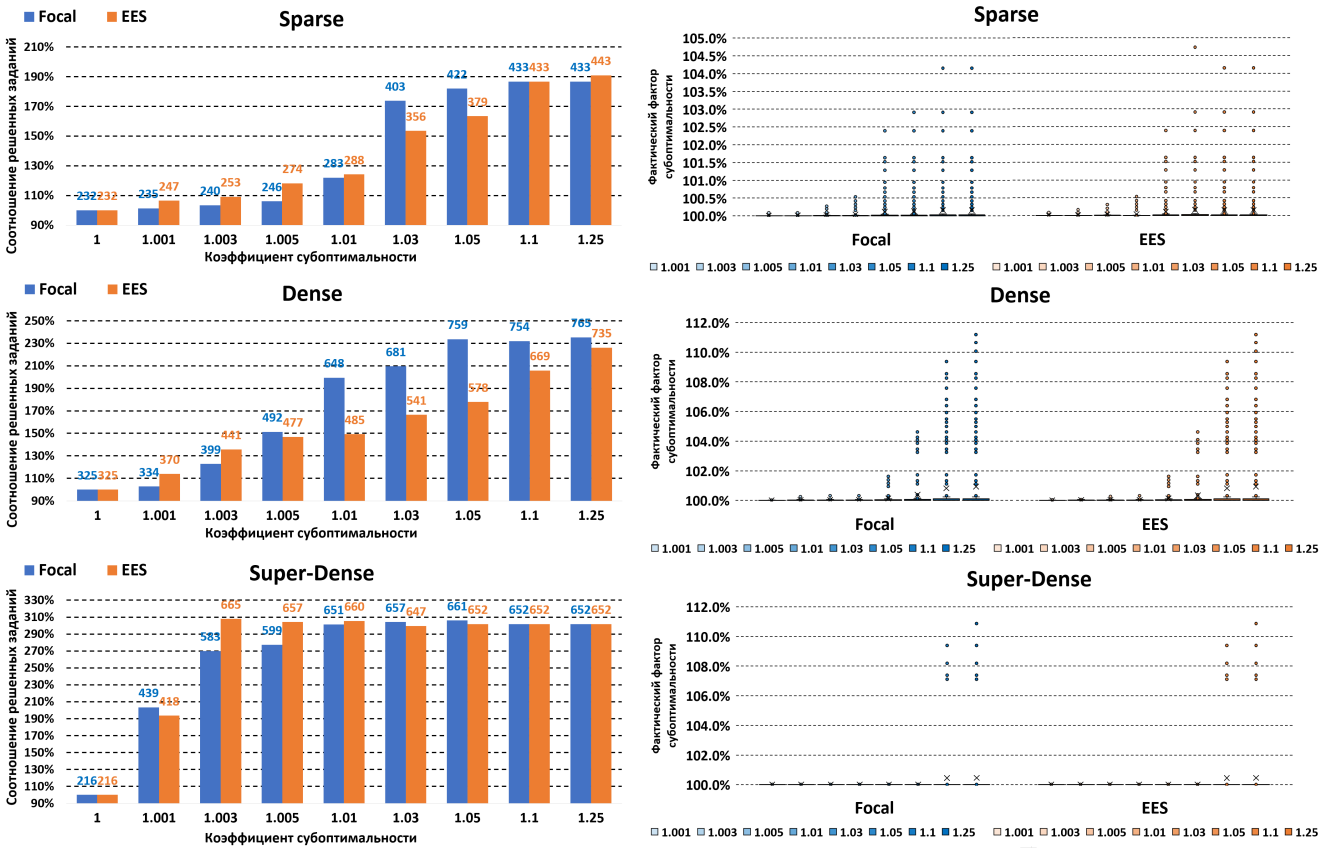


Рисунок 5.13 — Сравнение субоптимальных модификаций алгоритма CCBS на ГНС в зависимости от коэффициента субоптимальности.

циента субоптимальности на этой карте не приносит положительного эффекта. Такое поведение объясняется тем, что благодаря наличию большого числа вершин и ребер в этом графе, агенты имеют возможность строить траектории близкие к изначальным, благодаря чему итоговая стоимость бесконфликтного решения близка к стоимости начального решения. В связи с этим все альтернативные решения, рассматриваемые алгоритмом в процессе работы, имеют значения близкие к стоимости начального решения. Этот же вывод подтверждает правый график, на котором практически полностью отсутствуют выбросы, соответствующие заданиям, в которых стоимость решения существенно отличается от стоимости оптимального решения.

Экспериментальные исследования алгоритма CCBS и различных его модификаций продемонстрировали возможность применения предложенного подхода на графах нерегулярной структуры. Большинство существующих алгоритмов либо в принципе не способны с ними работать (т.к. принцип их работы завязан на дискретность ГРД), либо же их программные реализации не предполагают использование иных видов графов, отличных от ГРД. Результаты экспериментов на ГНС продемонстрировали аналогичные тренды, как и на ГРД, и позволили

подтвердить сделанные выводы о поведении и преимуществах тех или иных модификаций. В частности, приоритизация конфликтов проявляет себя наилучшим образом при использовании на графах с низкой связностью, в то время как непересекающееся разделение, напротив, демонстрирует лучшие результаты на графах с высокой степенью связности.

5.6 Сравнение с существующими аналогами

Заключительная серия экспериментов посвящена сравнению алгоритма CCBS с другими существующими алгоритмами, решающими задачу многоагнетного планирования, – E-ICTS[37], CBS+TAB[89], CBS-CT[38]. Данные алгоритмы нельзя в полной мере назвать аналогами, т.к. они не предполагают возможности совершения действий произвольной продолжительности. При этом они могут осуществлять планирование с действиями различной продолжительности, однако требуют установления параметра шага дискретизации времени. По результатам проведенного предварительного тестирования, для данного параметра было выбрано значение 1/1000. Это значение позволяет максимально приблизить эти алгоритмы к CCBS по качеству решений и при этом не оказать существенного негативного влияния на скорость работы алгоритмов. При сравнении с другими алгоритмами использовалась лучшая из оптимальных модификаций CCBS – CCBS+PC+DS+HL. Все алгоритмы имели идентичные входные данные, касающиеся графа и расположения стартовых/целевых положений. Они также имели одинаковый лимит времени работы - 30 секунд.

В качестве входных данных использовались 4 ГРД – `empty-16-16`, `rooms-32-32-4`, `warehouse-170-84-2-2`, `den520d` с коэффициентами связности $k = 2,3,4,5$. В данном эксперименте тестирование на ГНС не проводилось, т.к. программные реализации других алгоритмов не поддерживают работу с входными данными подобного рода и работают только при использовании карт, представленных в виде ГРД.

В качестве основного критерия, по которому осуществлялось сравнение, было суммарное число решенных заданий. В данном сравнении не использовался критерий качества решенных заданий, т.к. условно считалось, что все алгоритмы отыскивают оптимальные решения. При этом корректно оценить вычислитель-

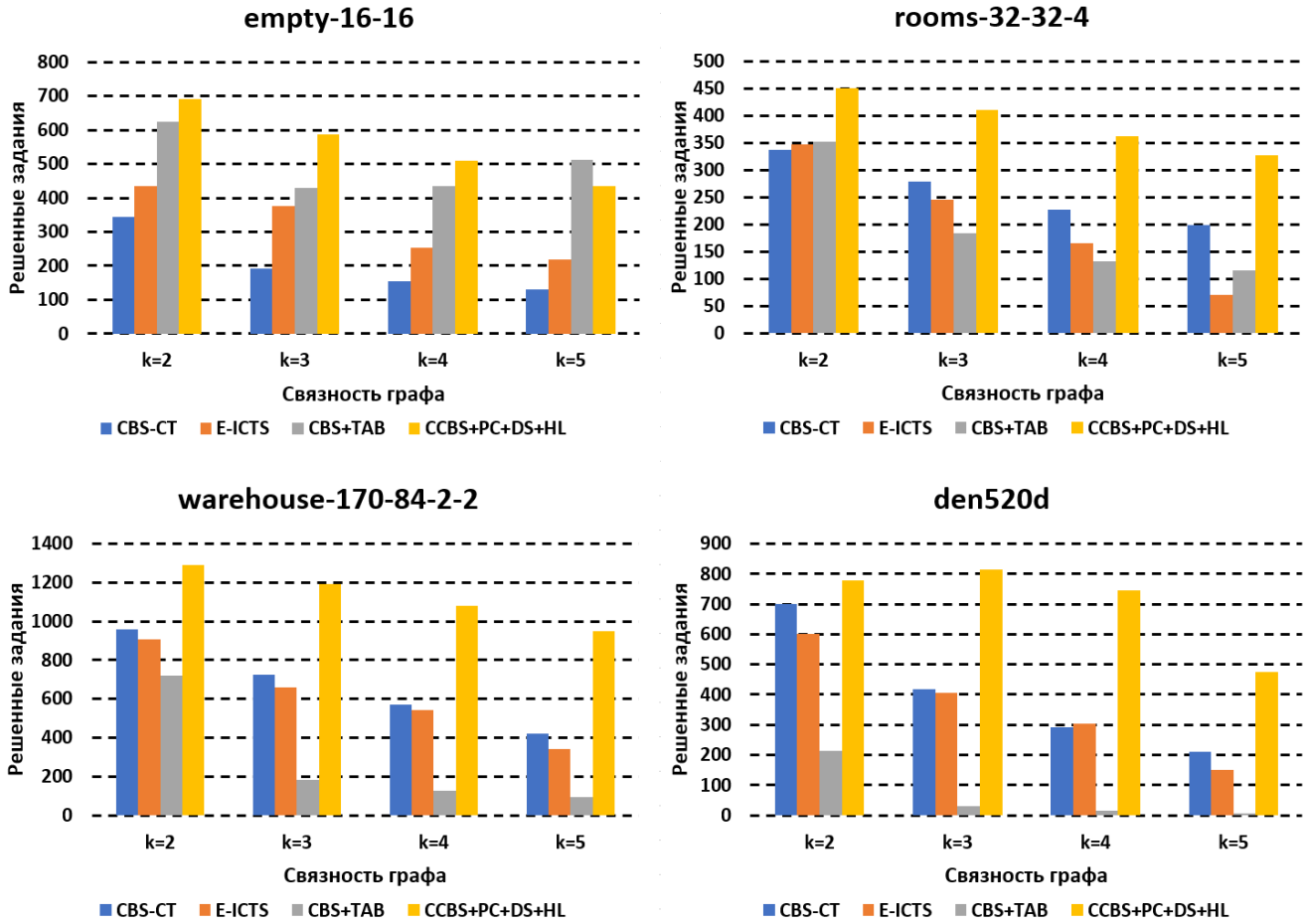


Рисунок 5.14 — Сравнение алгоритмов CBS-CT, E-ICTS, CBS+TAB и CCBS+PC+DS+HL на различных картах и связностях графа по числу решенных заданий.

ную эффективность алгоритмов, используя в качестве критерия абсолютное время работы алгоритмов, не представляется возможным, в связи с существенными различиями в реализациях алгоритмов. Реализации других алгоритмов были либо взяты из открытых источников (E-ICTS, CBS+TAB), либо получены непосредственно от авторов соответствующих алгоритмов (CBS-CT).

Результаты заключительной серии тестов, в которой проводилось сравнение предлагаемого алгоритма с другими подходами, решающими близкую к рассматриваемой постановке задачу многоагентного планирования, показаны на Рисунке 5.14. Почти во всех случаях предлагаемый алгоритм показывает результаты значительно превышающие результаты всех других протестированных подходов. Наибольшая разница наблюдается при использовании высоких значений для коэффициента связности ($k = 4{,}5$), а также на картах больших размеров. Единственный случай, в котором модифицированный алгоритм CCBS не показывает лучшие результаты наблюдается на карте `enpty-16-16` с коэффициентом

связности $k = 5$. В этом случае наибольшее число заданий сумел решить алгоритм CBS+TAB, однако на картах большого размера данный алгоритм показал худшие результаты из всех протестированных подходов вне зависимости от используемого коэффициента связности.

5.7 Выводы по главе

В данной главе было приведено описание проведенных экспериментальных исследований. Первая серия экспериментов была посвящена анализу эффективности работы алгоритма CCBS и анализу качества отыскиваемых им решений. Вторая серия экспериментов была посвящена исследованию модификаций алгоритма CCBS, сохраняющих оптимальность отыскиваемых решений. Результаты проведенных экспериментальных исследований продемонстрировали следующие тенденции: эффективность той или иной модификации может сильно зависеть от связности графа, на котором осуществляется планирования. При низкой связности наибольший прирост дает использование приоритизации конфликтов, в то время как на графах с высокой степенью связности лучше результаты демонстрирует непересекающееся разделение. Использование эвристической функции на верхнем уровне алгоритма не продемонстрировало существенного изменения числа решаемых заданий, однако, позволило снизить число итераций алгоритма. Наилучшие результаты в большинстве случаев демонстрирует версия, которая сочетает в себе все 3 улучшения. Следующая серия экспериментов была посвящена сравнению субоптимальных модификаций алгоритма CCBS. Обе модификации показали значительный прирост эффективности и общего числа решенных заданий в сравнении с оригинальным алгоритмом. Сравнивая две субоптимальные версии между собой, нельзя однозначно сказать какая из них лучше, т.к. во многом результаты получились довольно близкими и лучшие результаты показывали разные алгоритмы в зависимости от тестируемой карты. Помимо тестов на ГРД, алгоритм CCBS, а также его модификации были дополнительно протестированы на ГНС. Результаты этих тестов продемонстрировали возможность применения CCBS на ГНС, а также подтвердили выводы, сделанные по результатам тестов на ГРД. Заключительная серия экспериментов посвящена сравнению алгоритма CCBS с другими существующими подходами, способными решать за-

дачу многоагентного планирования с возможностью учета действий различной продолжительности. В данном случае использовалась модифицированная версия алгоритма, показавшая лучшие результаты в большинстве случаев при сравнении различных модификаций – CCBS+PC+DS+HL. Практически во всех случаях предлагаемый алгоритм значительно превосходит другие существующие подходы по числу решенных заданий, а в некоторых – разница более чем двукратная.

Заключение

В работе была рассмотрена задача планирования совокупности неконфликтных траекторий для множества агентов. Проведенный анализ существующих методов решения этой задачи продемонстрировал наличие большого числа ограничений и допущений, без соблюдения которых большинство существующих алгоритмов не способны функционировать. В частности, большинство алгоритмов опираются на допущение о дискретности времени, что ограничивает набор возможных действий агентов. В связи с этим в работе была рассмотрена постановка задачи, которая допускает возможность осуществления действий произвольной продолжительности. Для решения этой задачи было предложено использовать подход конфликтно-ориентированного поиска. Для возможности применения этого подхода к рассматриваемой постановке задачи были предложены оригинальный способ описания конфликтов, возникающих между агентами, использование интервальных ограничений, а также оригинальный метод планирования индивидуальных траекторий с учетом интервальных ограничений.

Основные результаты, полученные в ходе проведенного диссертационного исследования, заключаются в следующем:

1. Разработан алгоритм планирования совокупности неконфликтных траекторий для группы агентов с учетом возможности совершения действий произвольной продолжительности. Сформулирована и доказана теорема, гарантирующая, что отыскиваемые алгоритмом решения являются оптимальными.
2. Разработан ряд модификаций, в частности приоритизация конфликтов, непересекающееся разделение, а также эвристики верхнего уровня, позволяющие повысить вычислительную эффективность алгоритма. Сформулирована и доказана теорема, гарантирующая, что предложенные модификации не нарушают свойство оптимальности.
3. Разработаны две модификации алгоритма, которые позволяют отыскивать субоптимальные решения, обладают повышенной вычислительной эффективностью и позволяют устанавливать значение фактора субоптимальности, ограничивающие максимальное возможное превышение стоимости отыскиваемых решений в сравнении с оптимальными реше-

ниями. Ограниченная субоптимальность решений, отыскиваемых данными алгоритмами, была теоретически доказана.

Разработанный алгоритм и его модификации были программно реализованы на языке C++. Программный код, а также исходные данные, использованные во время проведения модельных экспериментальных исследований находятся в открытом доступе и доступны по адресу https://github.com/PathPlanning/Continuous-CBS

Список литературы

1. *Xiaohe*, *D.* The annual business volume of express delivery in China exceeded 100 billion [Электронный ресурс] / D. Xiaohe. — 2021. — URL: http://www.news.cn/2021-12/08/c_1128141875.htm (дата обр. 25.05.2022).
2. *Banker*, *S.* New Robotic Solutions For The Warehouse [Text] / S. Banker. — 2017. — URL: https://www.forbes.com/sites/stevebanker/2017/03/07/new-robotic-solutions-for-the-warehouse (visited on 05/25/2022).
3. *Zhang*, *D.* Artificial intelligence in E-commerce fulfillment: A case study of resource orchestration at Alibaba's Smart Warehouse [Текст] / D. Zhang, L. Pee, L. Cui // International Journal of Information Management. — 2021. — Т. 57. — С. 102304.
4. *Consulting*, *N. M. S.* Warehouse Robotics Market Report [Text] / N. M. S. Consulting. — 2021. — URL: https://www.nextmsc.com/report/warehouse-robotics-market (visited on 05/25/2022).
5. *Андрейчук*, *А. А.* Алгоритм планирования и согласования совокупности траекторий для группы интеллектуальных агентов [Текст] / А. А. Андрейчук // Искусственный интеллект и принятие решений. — 2018. — С. 72—85.
6. *Андрейчук*, *А. А.* Эффективный поиск ограниченно-субоптимальных решений задачи многоагентного планирования [Текст] / А. А. Андрейчук // Искусственный интеллект и принятие решений. — 2022. — № 1. — С. 57—70.
7. Multi-agent pathfinding with continuous time [Текст] / A. Andreychuk [и др.] // Artificial Intelligence. — 2022. — С. 103662.
8. Improving Continuous-time Conflict Based Search [Текст] / A. Andreychuk [и др.] // Proceedings of The 14th International Symposium on Combinatorial Search, SoCS 2021. — 2021. — С. 145—146.
9. Improving continuous-time conflict based search [Текст] / A. Andreychuk [и др.] // Proceedings of the AAAI Conference on Artificial Intelligence. Т. 35. — 2021. — С. 11220—11227.
10. *Andreychuk*, *A.* Multi-agent path finding with kinematic constraints via conflict based search [Текст] / A. Andreychuk // Lecture Notes in Artificial Intelligence. Т. 12412. — Springer, Cham, 2020. — С. 29—45.

11. Multi-agent pathfinding with continuous time [Текст] / A. Andreychuk [и др.] // Proceedings of the 28th International Joint Conference on Artificial Intelligence. — 2019. — С. 39—45.

12. *Андрейчук, А. А.* Исследование алгоритма конфликтно-ориентированного поиска для решения задачи планирования совокупности неконфликтных траектория для множества агентов [Текст] / А. А. Андрейчук // Семнадцатая Национальная конференция по искусственному интеллекту с международным участием. КИИ-2019 (21–25 октября 2019 г., г. Ульяновск, Россия). Сборник научных трудов. Т. 1. — 2019. — С. 93—101.

13. *Wurman, P. R.* Coordinating hundreds of cooperative, autonomous vehicles in warehouses [Текст] / P. R. Wurman, R. D'Andrea, M. Mountz // AI magazine. — 2008. — Т. 29, № 1. — С. 9—9.

14. Planning, scheduling and monitoring for airport surface operations [Текст] / R. Morris [и др.] // Workshops at the Thirtieth AAAI Conference on Artificial Intelligence. — 2016.

15. *Hagelback, J.* A multi-agent potential field-based bot for a full RTS game scenario [Текст] / J. Hagelback, S. Johansson // Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Т. 5. — 2009. — С. 28—33.

16. Feasibility study: Moving non-homogeneous teams in congested video game environments [Текст] / H. Ma [и др.] // Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference. — 2017.

17. *Лабутин, А. Н.* Агентно-ориентированное моделирование системы управления подразделениями при ликвидации чрезвычайных ситуаций на объектах химической промышленности [Текст] / А. Н. Лабутин, А. О. Семенов, Д. В. Тараканов // Современные наукоемкие технологии. Региональное приложение. — 2011. — № 3. — С. 94—99.

18. *Баранюк, В.* Планирование действий смешанных робототехнических группировок в условиях «балансирования на грани» [Текст] / В. Баранюк, Д. Миняйло, О. Смирнова // International Journal of Open Information Technologies. — 2016. — Т. 4, № 12. — С. 16—20.

19. Alibaba signs agreement with Belgium for e-commerce trade hub [Текст]. — 2018. — URL: https://www.reuters.com/article/usalibaba-logistics/alibaba-signs-agreement-with-belgium-for-ecommerce-trade-hub-idUSKBN1O410T.

20. *Yap*, *P.* Grid-based path-finding [Текст] / P. Yap // Conference of the canadian society for computational studies of intelligence. — Springer. 2002. — С. 44—55.

21. Primal: Pathfinding via reinforcement and imitation multi-agent learning [Текст] / G. Sartoretti [и др.] // IEEE Robotics and Automation Letters. — 2019. — Т. 4, № 3. — С. 2378—2385.

22. *Ma*, *Z.* Distributed heuristic multi-agent path finding with communication [Текст] / Z. Ma, Y. Luo, H. Ma // 2021 IEEE International Conference on Robotics and Automation (ICRA). — IEEE. 2021. — С. 8699—8705.

23. *Ma*, *Z.* Learning Selective Communication for Multi-Agent Path Finding [Текст] / Z. Ma, Y. Luo, J. Pan // IEEE Robotics and Automation Letters. — 2021. — Т. 7, № 2. — С. 1455—1462.

24. Searching with consistent prioritization for multi-agent path finding [Текст] / H. Ma [и др.] // Proceedings of the AAAI Conference on Artificial Intelligence. Т. 33. — 2019. — С. 7643—7650.

25. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery [Текст] / H. Ma [и др.] // Proceedings of the AAAI Conference on Artificial Intelligence. Т. 33. — 2019. — С. 7651—7658.

26. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks [Текст] / H. Ma [и др.] // Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems. — 2017. — С. 837—845.

27. *Ma*, *H.* Multi-agent path finding with delay probabilities [Текст] / H. Ma, T. S. Kumar, S. Koenig // Proceedings of the AAAI Conference on Artificial Intelligence. Т. 31. — 2017.

28. Robust multi-agent path finding and executing [Текст] / D. Atzmon [и др.] // Journal of Artificial Intelligence Research. — 2020. — Т. 67. — С. 549—579.

29. Conflict-based search with optimal task assignment [Текст] / W. Hönig [и др.] // Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems. — 2018.

30. *Ma*, *H.* Optimal Target Assignment and Path Finding for Teams of Agents [Текст] / H. Ma, S. Koenig // Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. — 2016. — С. 1144—1152.

31. Task and Path Planning for Multi-Agent Pickup and Delivery [Текст] / M. Liu [и др.] // Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. — 2019. — С. 1152—1160.

32. *Yoon*, *S.* Efficient multi-agent task allocation for collaborative route planning with multiple unmanned vehicles [Текст] / S. Yoon, J. Kim // IFAC-PapersOnLine. — 2017. — Т. 50, № 1. — С. 3580—3585.

33. Multi-agent path finding for large agents [Текст] / J. Li [и др.] // Proceedings of the AAAI Conference on Artificial Intelligence. Т. 33. — 2019. — С. 7627—7634.

34. *Atzmon*, *D.* Multi-Train Path Finding [Текст] / D. Atzmon, A. Diei, D. Rave // International Symposium on Combinatorial Search. Т. 10. — 2019.

35. Multi-Agent Path Finding with Kinematic Constraints [Текст] / W. Hoenig [и др.] // Proceedings of the International Conference on Automated Planning and Scheduling. Т. 26. — 2016. — С. 477—485.

36. *Wen*, *L.* CL-MAPF: Multi-Agent Path Finding for Car-Like robots with kinematic and spatiotemporal constraints [Текст] / L. Wen, Y. Liu, H. Li // Robotics and Autonomous Systems. — 2022. — Т. 150. — С. 103997.

37. *Walker*, *T. T.* Extended Increasing Cost Tree Search for Non-Unit Cost Domains. [Текст] / T. T. Walker, N. R. Sturtevant, A. Felner // IJCAI. — 2018. — С. 534—540.

38. Optimal and bounded-suboptimal multi-agent motion planning [Текст] / L. Cohen [и др.] // Twelfth Annual Symposium on Combinatorial Search. — 2019.

39. Multi-agent pathfinding: Definitions, variants, and benchmarks [Текст] / R. Stern [и др.] // Twelfth Annual Symposium on Combinatorial Search. — 2019.

40. *Цетлин*, *М. Л.* Исследования по теории автоматов и моделированию биологических систем [Текст] / М. Л. Цетлин. — М. : Наука, 1969.

41. *Варшавский*, *В. И.* Коллективное поведение автоматов [Текст] / В. И. Варшавский. — М. : Наука, 1973.

42. *Варшавский*, *В. И.* Оркестр играет без дирижера [Текст] / В. И. Варшавский, Д. А. Поспелов. — М. : Наука, 1973.

43. *Стефанюк*, *В. Л.* Коллективное поведение автоматов и задача устойчивого локального управления связи [Текст] : дис. … канд. / Стефанюк В. Л. — М.ИПУ, 1968. — С. 115.

44. *Кулинич*, *А. А.* Модель командного поведения агентов (роботов): когнитивный подход [Текст] / А. А. Кулинич // Управление большими системами. — 2014. — № 51. — С. 174—196.

45. *Каляев*, *И. А.* Модели и алгоритмы коллективного управления в группах роботов [Текст] / И. А. Каляев, А. Р. Гайдук, С. Г. Капустян. — 2009. — 280 с.

46. *Карпов*, *В. Э.* Модели социального поведения в групповой робототехнике [Текст] / В. Э. Карпов // Управление большими системами. — 2016. — № 59. — С. 165—232.

47. *Виноградов*, *А.* Динамические интеллектуальные системы. Ч.1. Представление знаний и основные алгоритмы [Текст] / А. Виноградов, Г. Осипов, Л. Жилякова // Известия АН. Теория и системы управления. — 2002. — № 6. — С. 119—127.

48. *Виноградов*, *А.* Динамические интеллектуальные системы. Ч.2. Моделирование целенаправленного поведения [Текст] / А. Виноградов, Г. Осипов, Л. Жилякова // Известия АН. Теория и системы управления. — 2003. — № 1. — С. 87—94.

49. *Осипов*, *Г. С.* Динамические интеллектуальные системы [Текст] / Г. С. Осипов // Искусственный интеллект и принятие решений. — 2008. — № 1. — С. 47—54.

50. *Германчук*, *М. С.* Метаэвристические алгоритмы для многоагентных задач маршрутизации [Текст] / М. С. Германчук, Д. В. Лемтюжникова, В. А. Лукьяненко // Проблемы управления. — 2020. — № 6. — С. 3—14.

51. *Жилякова*, *Л. Ю.* Графовые методы решения задачи об оптимальном назначении локомотивов на линейном участке железной дороги – без ограничений и с ограничениями [Текст] / Л. Ю. Жилякова, Н. А. Кузнецов // Автоматика и телемеханика. — 2021. — № 5. — С. 45—67.

52. Мультиагентный подход к решению сложной задачи построения расписаний в крупномасштабной системе управления железнодорожным движением [Текст] / А. А. Белоусов [и др.] // Управление развитием больших систем (MLSD'2014). — 2014. — С. 252—263.

53. Эффективное функционирование смешанной неоднородной команды в коллаборативной робототехнической системе [Текст] / Р. Р. Галин [и др.] // Информатика и автоматизация. — 2021. — Т. 20, № 6. — С. 1224—1253.

54. *Браништов, С. А.* Стратегии движения мобильных роботов в типовых ситуациях [Текст] / С. А. Браништов, П. А. Харланова, О. А. Байбакова // Управление развитием крупномасштабных систем (MLSD'2018). — 2018. — С. 37—41.

55. *Юдинцев, Б. С.* Синтез нейросетевой системы планирования траекторий для группы мобильных роботов [Текст] / Б. С. Юдинцев // Системы управления, связи и безопасности. — 2019. — № 4. — С. 163—186.

56. *Павлов, А. С.* Методика планирования траектории движения группы мобильных роботов в неизвестной замкнутой среде с препятствиями [Текст] / А. С. Павлов // Системы управления, связи и безопасности. — 2021. — № 3. — С. 38—59.

57. *Пшихопов, В. Х.* Планирование движения группы подвижных объектов в двумерной среде с препятствиями [Текст] / В. Х. Пшихопов, М. Ю. Медведев // Известия Южного федерального университета. Технические науки. — 2016. — 2 (175). — С. 6—22.

58. *Hart, P. E.* A formal basis for the heuristic determination of minimum cost paths [Текст] / P. E. Hart, N. J. Nilsson, B. Raphael // IEEE transactions on Systems Science and Cybernetics. — 1968. — Т. 4, № 2. — С. 100—107.

59. *Standley, T.* Finding optimal solutions to cooperative pathfinding problems [Текст] / T. Standley // Proceedings of the AAAI Conference on Artificial Intelligence. Т. 24. — 2010. — С. 173—178.

60. Enhanced partial expansion A [Текст] / M. Goldenberg [и др.] // Journal of Artificial Intelligence Research. — 2014. — Т. 50. — С. 141—187.

61. The increasing cost tree search for optimal multi-agent pathfinding [Текст] / G. Sharon [и др.] // Artificial intelligence. — 2013. — Т. 195. — С. 470—495.

62. Pruning techniques for the increasing cost tree search for optimal multi-agent pathfinding [Текст] / G. Sharon [и др.] // International Symposium on Combinatorial Search. T. 2. — 2011.

63. Conflict-Based Search For Optimal Multi-Agent Path Finding [Текст] / G. Sharon [и др.] // Proceedings of the AAAI Conference on Artificial Intelligence. T. 26. — 2012. — C. 563—569.

64. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding [Текст] / E. Boyarski [и др.] // Twenty-Fourth International Joint Conference on Artificial Intelligence. — 2015.

65. Adding heuristics to conflict-based search for multi-agent path finding [Текст] / A. Felner [и др.] // Proceedings of the International Conference on Automated Planning and Scheduling. T. 28. — 2018. — C. 83—87.

66. Symmetry-breaking constraints for grid-based multi-agent path finding [Текст] / J. Li [и др.] // Proceedings of the AAAI Conference on Artificial Intelligence. T. 33. — 2019. — C. 6087—6095.

67. New techniques for pairwise symmetry breaking in multi-agent path finding [Текст] / J. Li [и др.] // Proceedings of the International Conference on Automated Planning and Scheduling. T. 30. — 2020. — C. 193—201.

68. Symmetry breaking for k-robust multi-agent path finding [Текст] / Z. Chen [и др.] // Proceedings of the AAAI Conference on Artificial Intelligence. T. 35. — 2021. — C. 12267—12274.

69. Generalizing multi-agent path finding for heterogeneous agents [Текст] / D. Atzmon [и др.] // 13th International Symposium on Combinatorial Search, SoCS 2020. — The AAAI Press. 2020. — C. 101—105.

70. Efficient SAT approach to multi-agent path finding under the sum of costs objective [Текст] / P. Surynek [и др.] // Proceedings of the twenty-second european conference on artificial intelligence. — 2016. — C. 810—818.

71. *Surynek, P.* Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories [Текст] / P. Surynek // International Symposium on Combinatorial Search. T. 10. — 2019.

72. *Yu, J.* Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics [Текст] / J. Yu, S. M. LaValle // IEEE Transactions on Robotics. — 2016. — T. 32, № 5. — C. 1163—1177.

73. Branch-and-cut-and-price for multi-agent pathfinding [Текст] / E. Lam [и др.] // International Joint Conference on Artificial Intelligence 2019. — Association for the Advancement of Artificial Intelligence (AAAI). 2019. — С. 1289—1296.

74. *Lam*, *E.* New valid inequalities in branch-and-cut-and-price for multi-agent path finding [Текст] / E. Lam, P. Le Bodic // Proceedings of the International Conference on Automated Planning and Scheduling. Т. 30. — 2020. — С. 184—192.

75. Branch-and-cut-and-price for multi-agent path finding [Текст] / E. Lam [и др.] // Computers & Operations Research. — 2022. — Т. 144. — С. 105809.

76. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem [Текст] / M. Barer [и др.] // Proceedings of the Twenty-first European Conference on Artificial Intelligence. — 2014. — С. 961—962.

77. *Li*, *J.* Eecbs: A bounded-suboptimal search for multi-agent path finding [Текст] / J. Li, W. Ruml, S. Koenig // Proceedings of the AAAI Conference on Artificial Intelligence. Т. 35. — 2021. — С. 12353—12362.

78. *Huang*, *T.* Learning Node-Selection Strategies in Bounded Suboptimal Conflict-Based Search for Multi-Agent Path Finding [Текст] / T. Huang, B. Dilkina, S. Koenig // International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS). — 2021.

79. *Aljalaud*, *F.* Finding bounded suboptimal multi-agent path planning solutions using increasing cost tree search [Текст] / F. Aljalaud, N. Sturtevant // International Symposium on Combinatorial Search. Т. 4. — 2013.

80. Modifying optimal SAT-based approach to multi-agent path-finding problem to suboptimal variants [Текст] / P. Surynek [и др.] // International Symposium on Combinatorial Search. Т. 8. — 2017.

81. *Surynek*, *P.* Bounded Sub-optimal Multi-Robot Path Planning Using Satisfiability Modulo Theory (SMT) Approach [Текст] / P. Surynek // 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). — IEEE. 2020. — С. 11631—11637.

82. *Erdmann*, *M.* On multiple moving objects [Текст] / M. Erdmann, T. Lozano-Perez // Algorithmica. — 1987. — Т. 2, № 1. — С. 477—521.

83. A new constraint satisfaction perspective on multi-agent path finding: Preliminary results [Текст] / J. Wang [и др.] // 18th International Conference on Autonomous Agents and Multi-Agent Systems. — 2019.

84. Learning a Priority Ordering for Prioritized Planning in Multi-Agent Path Finding [Текст] / S. Zhang [и др.] // Proceedings of the International Symposium on Combinatorial Search. Т. 15. — 2022. — С. 208—216.

85. *Cap*, *M.* Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures [Текст] / M. Cap, J. Vokrinek, A. Kleiner // Proceedings of the international conference on automated planning and scheduling. Т. 25. — 2015. — С. 324—332.

86. *Wang*, *K.-H. C.* MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees [Текст] / K.-H. C. Wang, A. Botea // Journal of Artificial Intelligence Research. — 2011. — Т. 42. — С. 55—90.

87. *Luna*, *R. J.* Push and swap: Fast cooperative path-finding with completeness guarantees [Текст] / R. J. Luna, K. E. Bekris // Twenty-Second International Joint Conference on Artificial Intelligence. — 2011.

88. *De Wilde*, *B.* Push and rotate: a complete multi-agent pathfinding algorithm [Текст] / B. De Wilde, A. W. Ter Mors, C. Witteveen // Journal of Artificial Intelligence Research. — 2014. — Т. 51. — С. 443—492.

89. *Walker*, *T. T.* Generalized and sub-optimal bipartite constraints for conflict-based search [Текст] / T. T. Walker, N. R. Sturtevant, A. Felner // Proceedings of the AAAI Conference on Artificial Intelligence. Т. 34. — 2020. — С. 7277—7284.

90. Conflict-based search for optimal multi-agent pathfinding [Текст] / G. Sharon [и др.] // Artificial Intelligence. — 2015. — Т. 219. — С. 40—66.

91. *Guy*, *S. J.* Guide to Anticipatory Collision Avoidance [Текст] / S. J. Guy, I. Karamouzas // Game AI Pro 2: Collected Wisdom of Game AI Professionals / под ред. S. Rabin. — 2015. — Гл. 19. С. 195—208.

92. *Jiménez*, *P.* 3D collision detection: a survey [Текст] / P. Jiménez, F. Thomas, C. Torras // Computers & Graphics. — 2001. — Т. 25, № 2. — С. 269—285.

93. *Walker*, *T. T.* Collision detection for agents in multi-agent pathfinding [Текст] / T. T. Walker, N. R. Sturtevant // arXiv preprint arXiv:1908.09707. — 2019.

94. *Hendricks*, *M. C.* Rotated ellipses and their intersections with lines [Текст] / M. C. Hendricks. — 2012.

95. *Phillips*, *M.* Sipp: Safe interval path planning for dynamic environments [Текст] / M. Phillips, M. Likhachev // 2011 IEEE International Conference on Robotics and Automation. — IEEE. 2011. — С. 5628—5635.

96. Disjoint splitting for multi-agent path finding with conflict-based search [Текст] / J. Li [и др.] // Proceedings of the International Conference on Automated Planning and Scheduling. Т. 29. — 2019. — С. 279—283.

97. *Pearl*, *J.* Studies in semi-admissible heuristics [Текст] / J. Pearl, J. H. Kim // IEEE transactions on pattern analysis and machine intelligence. — 1982. — № 4. — С. 392—399.

98. *Thayer*, *J. T.* Bounded suboptimal search: a direct approach using inadmissible estimates [Текст] / J. T. Thayer, W. Ruml // Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume One. — 2011. — С. 674—679.

99. *Sturtevant*, *N. R.* Benchmarks for grid-based pathfinding [Текст] / N. R. Sturtevant // IEEE Transactions on Computational Intelligence and AI in Games. — 2012. — Т. 4, № 2. — С. 144—148.

100. *Yakovlev*, *K.* Prioritized multi-agent path finding for differential drive robots [Текст] / K. Yakovlev, A. Andreychuk, V. Vorobyev // 2019 European Conference on Mobile Robots (ECMR). — IEEE. 2019. — С. 1—6.

101. *Kavraki*, *L. E.* Analysis of probabilistic roadmaps for path planning [Текст] / L. E. Kavraki, M. N. Kolountzakis, J.-C. Latombe // IEEE Transactions on Robotics and automation. — 1998. — Т. 14, № 1. — С. 166—171.

102. *Sucan*, *I. A.* The open motion planning library [Текст] / I. A. Sucan, M. Moll, L. E. Kavraki // IEEE Robotics & Automation Magazine. — 2012. — Т. 19, № 4. — С. 72—82.

Список рисунков

Список таблиц