

Санкт-Петербургский государственный университет
Факультет Математики и Компьютерных наук

**Алгоритмы конфликтно-ориентированного поиска для задачи
много-агентного планирования: CBS Basics (CBS+PC, CBS+H,
CBS+DS)**

Работу выполнили студенты 4 курса
образовательная программа «Математика» 01.03.01
очной формы обучения
Шульженко Александр
Казовская Анастасия
Клименко Полина

Санкт-Петербург
2022 г.

Введение

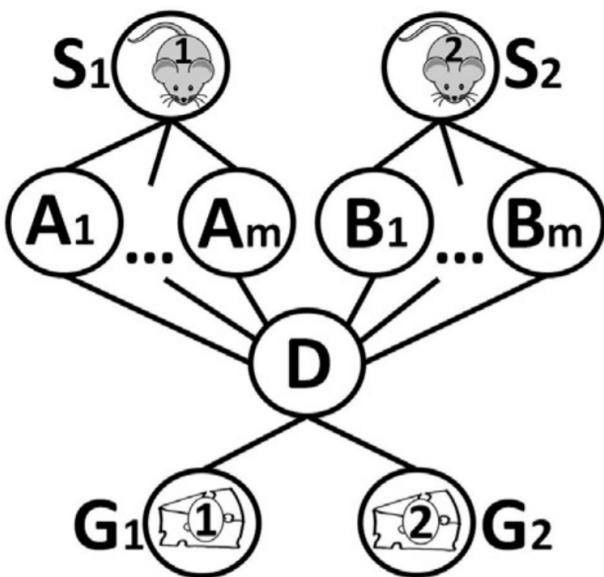
В данном проекте рассматривается задача мультиагентного планирования траекторий (Multi-Agent Path Finding (MAPF)). Данная задача подразумевает нахождение набора путей, состоящего из траекторий движения для каждого из агентов, без столкновений, где для каждого агента известны начальная и конечная позиции. Примеры применения включают автономные буксировщики самолетов, автоматизированные складские системы, офисных роботов и игровых персонажей в видеоиграх. Практические системы должны быстро находить пути без столкновений для таких агентов. Возьмем, к примеру, автоматизированные склады. Планирование движения роботов на таких складах затруднено, поскольку большая часть складских площадей используется для хранения, что приводит к узким коридорам, в которых роботы не могут разойтись. Складские роботы работают весь день, но упрощенная однократная версия задачи планирования пути — это простейшая форма задачи MAPF. Известны заблокированные ячейки и текущие ячейки н роботов. Каждому из н роботов назначается отдельная незаблокированная ячейка в качестве целевой ячейки. Проблема состоит в том, чтобы переместить роботов из их текущих ячеек в их целевые ячейки с дискретными временными шагами и позволить им ждать там. Цель оптимизации состоит в том, чтобы свести к минимуму период времени, то есть количество шагов по времени, пока все роботы не окажутся в своих целевых ячейках. В течение каждого временного шага каждый робот может ожидать в своей текущей ячейке или перемещаться из своей текущей ячейки в незаблокированную соседнюю ячейку в одном из четырех направлений (в том случае если в рассматриваемой модели доступны только четыре направления). Также можно построить модели для доступных восьми направлений и более). Роботы не должны сталкиваться. Два робота сталкиваются тогда и только тогда, когда в течение одного и того же временного шага они оба перемещаются в одну и ту же ячейку или оба перемещаются в текущую ячейку другого робота (вершинный конфликт) или же в одно и то же время переходят по одному и тому же ребру (то есть два агента совершают перемещение между соседними ячейками в одно и то же время). Существуют также версии задачи MAPF с целями оптимизации, отличными от промежутка времени (например, сумма временных шагов каждого робота, пока он не окажется в своей целевой ячейке) или немного другими правилами столкновения или движения.

Таким образом, задача MAPF встречается во многих практических областях, что подтверждает ее актуальность.

Формальная постановка задачи

Формальная постановка задачи мультиагентного планирования включает в себя формализацию понятия среды поиска, путей агентов, конфликтов и функции стоимости. Начнем описание с первого понятия — области поиска. Обычно пространством в котором могут находиться агенты является направленный граф $G(V, E)$, в котором вершины —

возможные положения агентов, а ребра — допустимые перемещения между положениями. Теперь, когда у нас задан график мы можем обозначить самих агентов (которых будет k штук) через a_1, a_2, \dots, a_k , у каждого есть стартовое положение $start_i$ и целевое положение $goal_i$, где $start$ и $goal$ — заданные вершины в графике. Таким образом мы получаем изначальную конфигурацию, которая является математически формальной задачей. В решаемой нами задаче вводится предположение о том, что



время дискретно. В t_0 каждый агент a_i находится в $start_i$, далее за один шаг агент может остаться в текущем положении (текущей вершине графа) или же перейти в одного из соседей данной вершины. Поскольку постановка задачи включает в себя несколько агентов, то в некоторый момент времени два из них могут оказаться в одной и той же вершине, или пройти по одному и тому же ребру. В связи с этим введем еще два обозначения: вершинного и реберного конфликта (vertex/edge conflict):

- *Vertex Conflict* — (a_i, a_j, v, t)
- *Edge Conflict* — (a_i, a_j, v_1, v_2, t)

Также нам потребуются обозначения, которые запрещают агенту находиться в конкретное время в конкретной вершине или проходить по конкретному ребру. Такие запреты называются vertex/edge constraint:

- *Vertex Constraint* — (a_i, v, t)
- *Edge Constraint* — (a_i, v_1, v_2, t)

Более детально, Vertex constraint запрещает агенту a_i находиться в момент времени t в вершине v графа, Edge constraint запрещает агенту a_i проходить в момент времени t по ребру e графа.

Также рассмотрим понятие функции стоимости (cost-function), которая оценивает полученное решение, то есть набор траекторий для каждого из агентов. В качестве функции стоимости часто выбирается одна из следующих функций:

Sum-of-cost, Makespan, Fuel.

Строго это можно выразить следующим образом: пусть решением для исходной задачи является набор траекторий (w_1, \dots, w_k) , где i -я траектория соответствует i -му агенту. Как уже упоминалось, траектория — это последовательность вершин нашего графа, где каждая следующая вершина является соседом предыдущей, первая вершина — стартовая, а последняя — финишная для данного агента. Тогда можно вычислить стоимость каждой траектории:

Пусть $w_i = (v_{i,1}, \dots, v_{i,n_i})$, тогда $cost(w_i) = n_i$, то есть просто длина данного пути. Тогда

$$Sum - of - costs(w_1, \dots, w_k) = \sum_{i=1}^k w_i.$$

Кроме такой функции стоимости можно использовать и другую, например Fuel. В данном случае каждому передвижению агента (за одну единицу времени) присваивается значение затраты топлива. Тогда $w_i = (v_{i,1}, \dots, v_{i,n_i})$, $cost(w_i) = \sum_j fuel(v_{i,j} \rightarrow v_{i,j+1})$, где

$fuel(v_{i,j} \rightarrow v_{i,j+1})$ обозначает затрату топлива при переходе из j в $j+1$ вершину.

Описания методов

CBS

Классическим методом решения задачи MAPF является алгоритм CBS. CBS — это двухуровневый алгоритм, на верхнем уровне которого поиск выполняется в дереве конфликтов (CT), которое представляет собой дерево, основанное на конфликтах между

Algorithm 1: high-level of CBS

Input: MAPF instance

1 $R.constraints = \emptyset$

2 $R.solution =$ find individual paths using the low-level()

3 $R.cost = SIC(R.solution)$

4 insert R to OPEN

5 **while** OPEN *not empty do*

6 $P \leftarrow$ best node from OPEN // *lowest solution cost*

7 Validate the paths in P until a conflict occurs.

8 **if** P has no conflict **then**

9 **return** $P.solution$ // P is goal

10 $C \leftarrow$ first conflict (a_i, a_j, v, t) in P

11 **foreach** agent a_i in C **do**

12 $A \leftarrow$ new node

13 $A.constraints \leftarrow P.constraints + (a_i, s, t)$

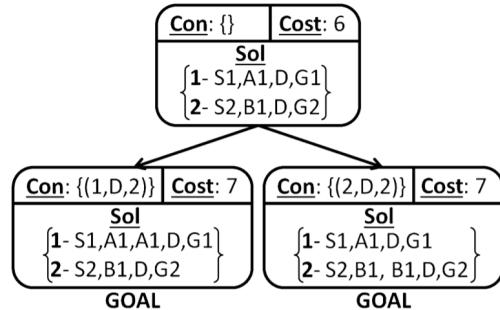
14 $A.solution \leftarrow P.solution$.

15 Update $A.solution$ by invoking low-level(a_i)

16 $A.cost = SIC(A.solution)$

17 Insert A to OPEN

отдельными агентами. Каждый узел в СТ представляет набор ограничений на движение агентов. На низком уровне выполняются быстрые поиски с одним агентом, чтобы



удовлетворить ограничениям, наложенным узлом

СТ высокого уровня.

Рис. 2. Псевдокод для верхнего уровня алгоритма CBS.

Рис.3. Пример СТ на верхнем уровне CBS.

Во многих случаях двухуровневая структура позволяет CBS проверять меньше состояний, чем A*, сохраняя при этом оптимальность. Рассмотрим каждый из уровней алгоритма более подробно:

На верхнем уровне CBS выполняет поиск в дереве ограничений (СТ)

(Рис 2). СТ — это бинарное дерево. Каждый узел N в СТ содержит следующие данные:

(1) Набор ограничений (N.constraints). Корень СТ содержит пустой набор ограничений. Дочерний узел узла в СТ наследует ограничения родителя и добавляет одно новое ограничение для одного агента.

(2) Набор (N.solution). Набор из k путей, по одному пути для каждого агента. Путь для агента a_i должен соответствовать ограничениям. Такие пути находятся на нижнем уровне алгоритма

(3) Общую стоимость (N.cost) текущего решения (сумма по всем стоимостям пути с одним агентом). Обозначим эту стоимость как f -значение узла.

Узел N в СТ является целевым узлом, когда N.solution корректно, т. е. множество путей для всех агентов не имеет конфликтов.

CBS и Disjoint Splitting

Основная идея этого метода состоит в том, что при разделении узла в СТ наборы траекторий, доступных для агентов в правом и левом поддереве могут сильно перекрываться, поэтому, чтобы избежать повторяющихся вычислений, вводятся так называемые positive constraints, которые обязывают агента быть в данном узле или пройти по данному ребру в указанное время.

CBS и High Level Heuristics

В данном варианте улучшения алгоритма рассматриваются эвристики на верхнем уровне СТ. Наиболее известные допустимые эвристики (то есть не переоценивающие реальное значение) для верхнего уровня СТ в CBS - это CG (эвристика, основанная на построении минимального вершинного покрытия графа конфликтов), DG (эвристика, основанная на построении минимального вершинного покрытия графа зависимостей агентов) и WDG (эвристика, основанная на построении минимального взвешенного вершинного покрытия графа зависимостей агентов). В нашем проекте реализована эвристика DG, поскольку она более информативна чем CG (то есть DG всегда больше либо равна CG), и в то же время требует не так много времени для подсчета, как WDG.

Основу эвристики DG, как уже отмечалось ранее, составляет граф зависимостей агентов. В каждой вершине СТ строится свой граф зависимостей $G_D = (V_D, E_D)$, а формально он определяется так. Каждый агент a_i индуцирует вершину $v_i \in V_D$. Между вершинами существует ребро $(v_i, v_j) \in E_D$ тогда и только тогда, когда агенты a_i и a_j зависимы, то есть все их пути минимальной стоимости, удовлетворяющие запретам в вершине, имеют конфликты. Как только граф G_D построен, остается найти его минимальное вершинное покрытие - оно и будет являться допустимой эвристикой.

Экспериментальное исследование

В данном проекте были реализованы следующие алгоритмы для решения задачи многоагентного планирования: CBS, CBS + disjoint splitting, CBS + MetaAgents, CBS + HighLevelHeuristics, CBS+PrioritizingConflicts, также был выполнен сравнительный анализ эффективности выбранных алгоритмов. Все

version	1								
23	room-64-64-16.map	64	64	45	7	4	56	92.04163055	
28	room-64-64-16.map	64	64	54	36	3	55	113.84062042	
17	room-64-64-16.map	64	64	7	23	50	26	70.28427124	
37	room-64-64-16.map	64	64	23	59	39	59	150.46803741	
2	room-64-64-16.map	64	64	54	33	44	33	10.00000000	
5	room-64-64-16.map	64	64	29	7	44	20	22.72792206	
19	room-64-64-16.map	64	64	58	31	1	26	79.11269836	
7	room-64-64-16.map	64	64	44	24	20	15	29.72792206	
24	room-64-64-16.map	64	64	20	38	7	26	99.76955261	
38	room-64-64-16.map	64	64	60	57	26	49	155.53910522	

необходимые материалы хранятся в репозитории: <https://github.com/PolinaKlimenko/CBS>.

Описание входных данных

Для тестирования алгоритмов использовалась открытая библиотека тестов movingai.com (конкретно, ее раздел посвященный тестированию алгоритмов для MAPF¹). Нами были выбраны 5 различных типов карт, на каждом из которых мы тестировали все приведенные выше алгоритмы и сравнивали их эффективность. Использованы следующие типы карт:

- Пустая
- Случайные препятствия
- Лабиринт
- Комната
- Склад

Примеры карт каждого типа изображены на рисунке 4. Нами были выбраны по две карты каждого из типов, для каждой из которых добавлена папка сценариев. Сценарий представляет набор координат начала и конца пути для каждого из агентов, где максимальное число агентов — 200 штук. Пример сценария изображен на рисунке выше.

Каждая строка файла со сценарием представляет

- 1) Размеры карты
- 2) Координаты агента
- 3) Координаты конца пути агента
- 4) Его оптимальный путь

¹ <https://movingai.com/benchmarks/mapf/index.html>

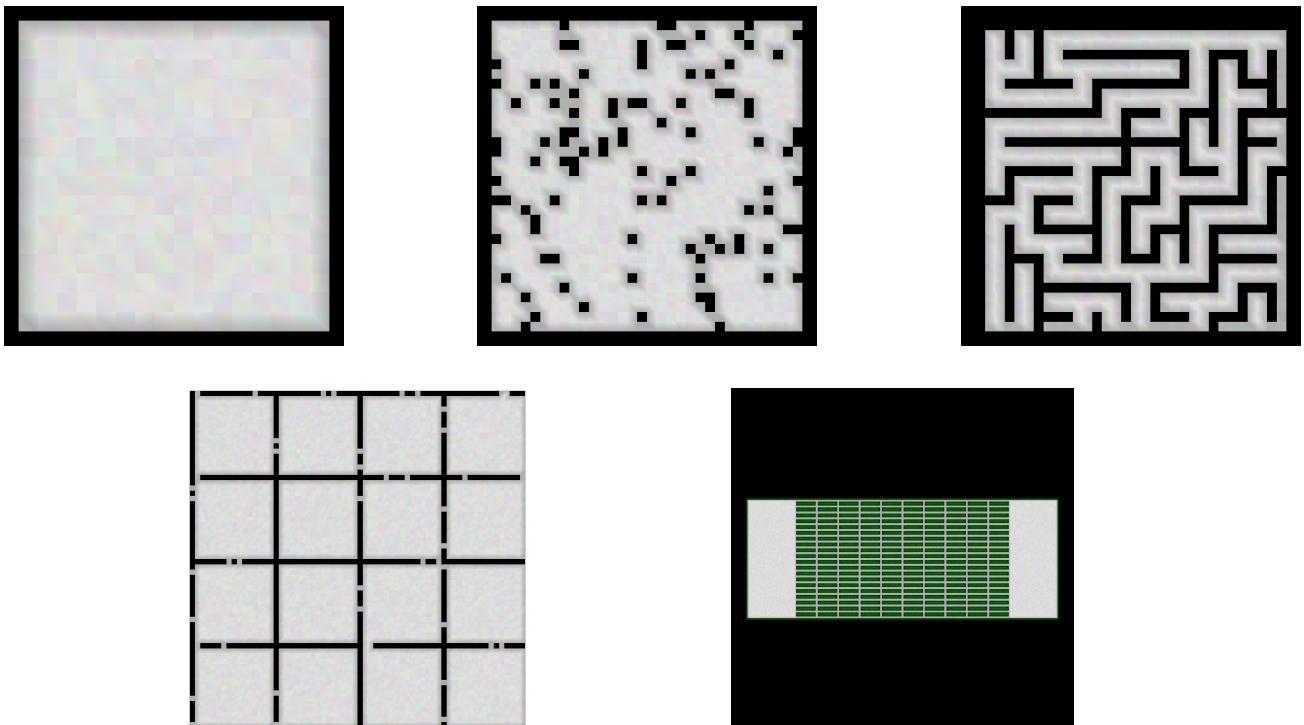


Рис. 4. Примеры карт каждого из приведенных типов: слева-направо, сверху-вниз:
Пустая, случайная, лабиринт, комната, склад.

Для тестирования оптимальности полученного решения с помощью каждого из алгоритмов мы рассматривали сумму длин оптимальных путей (с учетом конфликтов) для каждого из агентов.

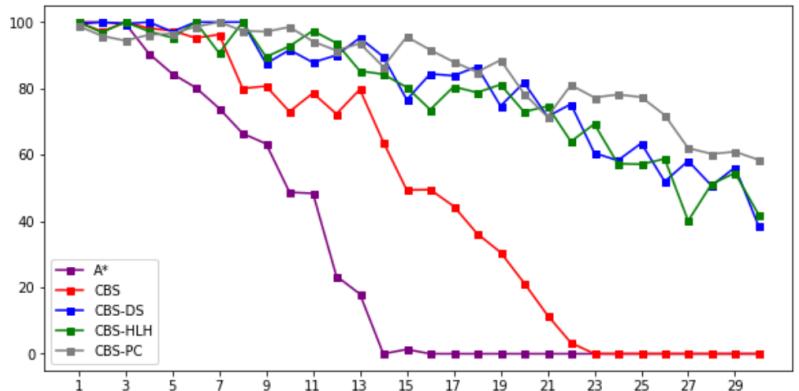
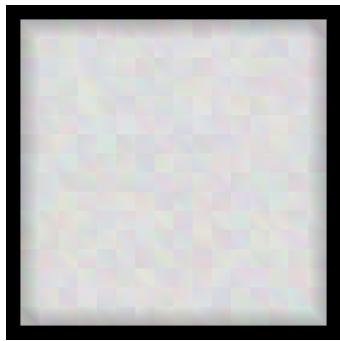
Эксперименты

Все проведенные эксперименты делятся на две группы: эксперименты на разных видах карт, сравнивающие эффективность по времени каждого из алгоритмов с другим и эксперименты на одном виде карт, сравнивающие эффективность нескольких алгоритмов по различным параметрам: числу раскрытых вершин в дереве конфликтов, и тд.

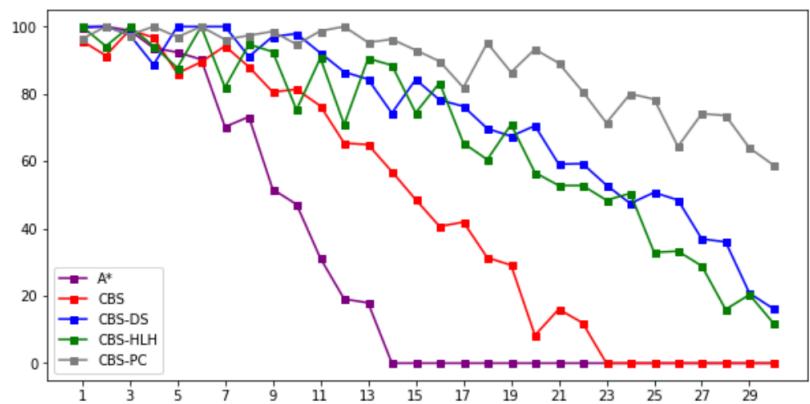
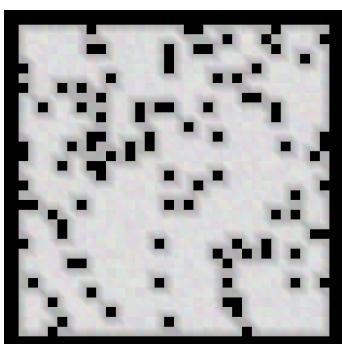
Размеры использованных карт 64x64 для всех карт, кроме склада, карта склад использовалась 161x63.

В первую очередь представим описание тестов первого типа. Как было упомянуто выше, мы рассматривали пять видов карт. Для каждого из видов карт мы рассматривали графики следующего вида: по оси ОХ откладывается число

агентов, которые помещаются на карту согласно сценарию, по оси ОY откладывается процент «решенных» сценариев от общего числа за время работы — 5 минут. «Решенных» в данном случае означает, что алгоритм вывел решение и это решение оптимально. Ниже представлены графики, полученные в результате проведения описанного эксперимента для каждого из видов карт.

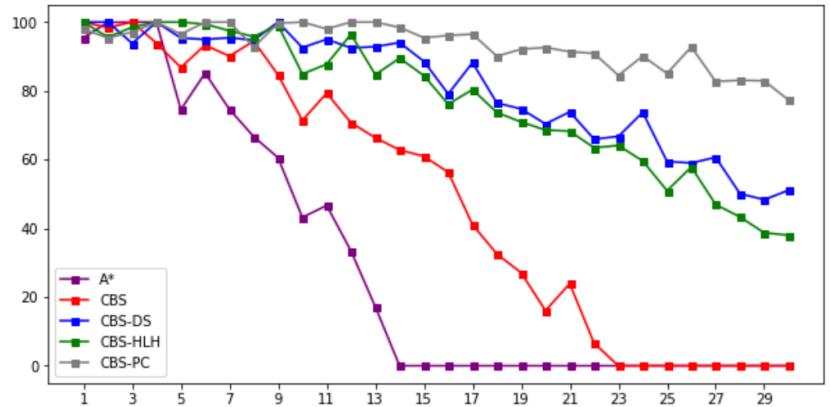
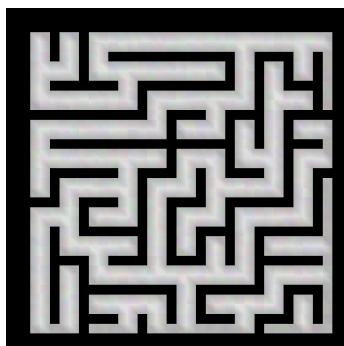


Первый тип карт: пустая карта. Как видно из графика для данной карты тяжело выделить явного фаворита, однако улучшенные версии CBS асимптотически превосходят в качестве baseline вариант. Это связано с тем, что для пустой карты улучшения не применяются так часто, как могли бы для других карт, поскольку в данной карте нет «бутылочных горлышек» и других труднопреодолимых мест.



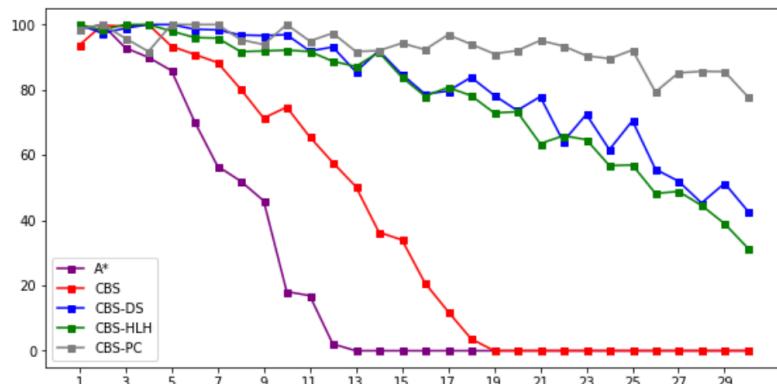
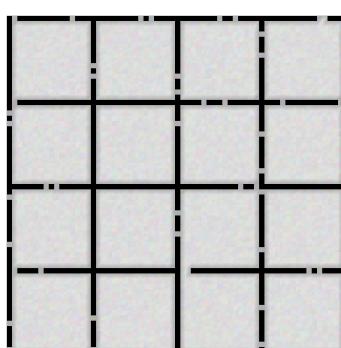
Второй тип карт: карта со случайными препятствиями: для данного типа карт уже явно прослеживается превосходство улучшенных вариантов алгоритма над базовым. С другой стороны тяжело выделить лучший алгоритм из дополненных версий, однако небольшой отрыв в производительности имеет алгоритм с

эвристиками на верхнем уровне. Это может быть связано с тем, что для такой карты все еще нет достаточного количества конфликтов, которые можно было бы эффективно приоритизировать, однако наличие эвристики помогает CBSправляться с заданиями за более быстрое время.



Третий тип карт: лабиринт: для данного типа карт превосходство улучшенных версий алгоритма становится очевидным. Среди всех версий лидирует CBS+PC, поскольку в данном типе карт огромное число столкновений, которые необходимо обрабатывать, хуже всего работает CBS+DS, поскольку приоритизация конфликтов и наличие эвристики позволяют добиться гораздо более хороших результатов при большом количестве агентов на «тесной» карте.

Четвертый тип карт: комната: для данного типа карт характерно наличие большого числа узких мест, в которых **часто** случаются конфликты, поэтому для данного типа карт приоритизация конфликтов показывает самые лучшие результаты.



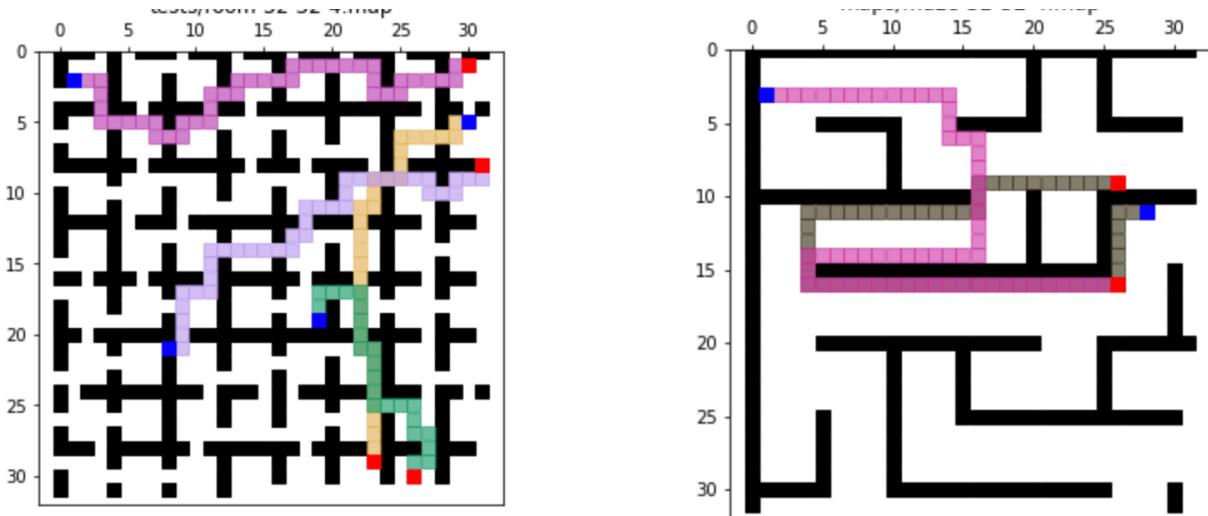
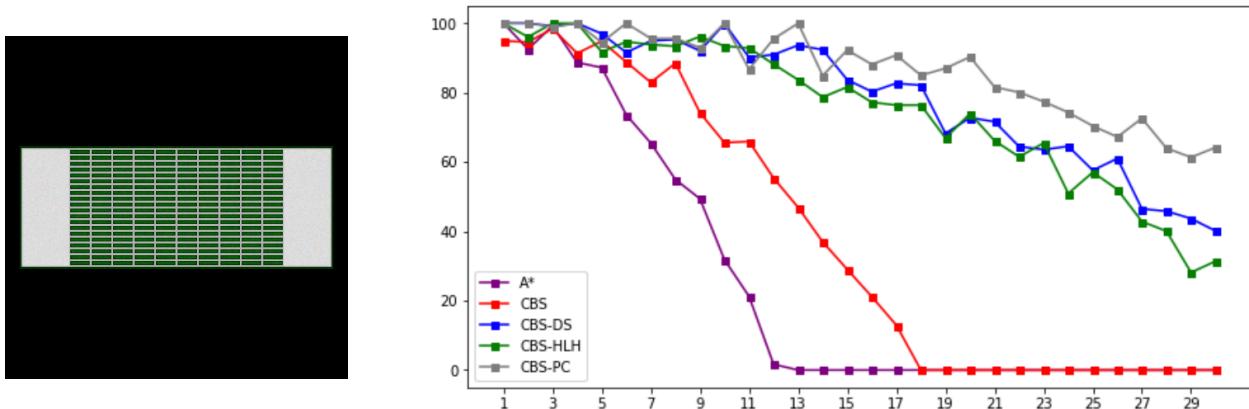


Рис 6. Примеры работы алгоритма на различных картах



Пятый тип карт: складской комплекс. Данный тип карт характеризуется наличием большого числа узких проходов, на которых возникают как реберные, так и вершинные конфликты, в связи с этим, необходима приоритизация как реберных так и вершинных конфликтов, поэтому, метод приоретизации снова показывает наилучшие результаты.

Несколько слов о тестировании алгоритмов только на карте типа «комната». Мы сравнивали различные характеристики алгоритмов для карты данного типа и получили следующие результаты: на графике по оси ОХ все так же откладывается число агентов, а по оси ОY число созданных вершин в ConflictTree.

Заключение

В работе была рассмотрена задача MAPF, и базовый алгоритм для ее решения — CBS. Также были рассмотрены различные улучшения данного алгоритма, такие как

добавление эвристик на верхний уровень СТ, которое строиться алгоритмом CBS, приоритизация конфликтов, Disjoint Splitting. Все перечисленные методы были реализованы в ходе проекта и протестирована эффективность каждого из них. Наборы данных для тестирования были взяты из открытой библиотеки MovingAI.com.