

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

УДК 519.163

**Отчет об исследовательском проекте на тему:
Улучшение алгоритмов решения задачи LMAPF с помощью графа с
динамической стоимостью переходов**

Выполнил студент:

группы #БПМИ235, 2 курса Юхневич Егор Владимирович

Принял руководитель проекта:

Яковлев Константин Сергеевич
к.ф.-м.н., заведующий кафедрой, доцент
Факультет компьютерных наук НИУ ВШЭ

Соруководитель:

Андрейчук Антон Андреевич
к.ф.-м.н., научный сотрудник
АО "Институт искусственного интеллекта"

Москва 2025

Содержание

Аннотация	3
1 Введение	4
1.1 Описание предметной области	4
1.2 Постановка задачи	4
1.3 Соревнование	7
2 Обзор литературы	8
2.1 Priority Inheritance with Backtracking	8
2.2 Windowed Parallel PIBT-LNS	8
2.3 Graph Guidance	8
2.4 Lazy constraints addition search for MAPF	8
2.5 Вывод обзора литературы	9
3 Описание предлагаемого метода	10
3.1 Priority Inheritance with Backtracking	10
3.2 Enhanced Priority Inheritance with Backtracking	10
3.3 Enhanced Priority Inheritance with Backtracking Large Neighborhood Search	13
3.4 Graph Guidance	13
3.5 Эксперименты	13
4 Результаты	27
5 Выводы	28
Список литературы	29

Аннотация

Задача многоагентного планирования путей (MAPF) - это задача перемещения нескольких агентов от начальных позиций к конечным целям без коллизий. Lifelong MAPF (LMAPF) является вариацией задачи MAPF, в которой агентам постоянно назначаются новые цели. Данный проект содержит подход к решению задачи LMAPF. А в качестве побочной цели - применение данного решения в соревновании The League of Robot Runners 2024, который подводит нас к нескольким интересным исследовательским задачам и направлениям на будущее. В проекте описываются три основные исследовательские задачи. Первая задача заключается в поиске высококачественных решений LMAPF в рамках ограниченного времени планирования (1 секунда на шаг) для большого числа агентов (порядка 10000) или чрезвычайно высокая плотность агентов (почти 100%). Вторая задача состоит в том, чтобы уменьшить заторы и влияние близорукого поведения алгоритмов LMAPF. В проекте представлены такие методы как разработка указаний по движению и правил дорожного движения для уменьшения заторов, включение прогнозирования будущего и поиска в режиме реального времени. Третья задача заключается в устранении пробелов между используемыми моделями LMAPF в литературе и в реальных приложениях.

Ключевые слова

Алгоритмы и структуры данных, C/C++, Python, NP, LNS, MAPF, LMAPF, MAPF-R, LMAPF-R, PIBT, EPIBT, EPIBT+LNS, WPPL, GG

1 Введение

1.1 Описание предметной области

Multi-agent pathfinding problem (MAPF) - задача многоагентного планирования, суть которой заключается в том, чтобы построить совокупность неконфликтных траекторий для множества агентов, каждый из которых обладает собственным стартовым и целевым положениями. Пространство, в котором оперируют агенты, моделируется с помощью графа, который определяет возможные положения агентов, а также набор возможных действий. При этом планирование осуществляется с учетом времени, т.к. необходимо избегать конфликтных ситуаций, при которых 2 или более агента одновременно занимают одну и ту же вершину или ребро графа.

Поиск оптимального решения задачи многоагентного планирования относится к классу NP-полных задач, т.е. его невозможно найти за полиномиальное время. Однако, существуют алгоритмы, который могут найти решение очень быстро (за полиномиальное время), при этом качество найденного решения будет крайне низким. Данные алгоритмы используют эвристики и системы правил для устранения конфликтов между агентами. В данном проекте предлагается разработать подобный алгоритм, который будет превосходить существующие аналоги по качеству решений (хотя бы в определенном классе задач).

1.2 Постановка задачи

Multiagent pathfinding (MAPF) [2] - задача поиска путей для нескольких агентов. Для моделирования пространства, в котором оперируют агенты, используется граф $G = (V, E)$, где V - вершины и E - ребра, и n агентов с их начальными и конечными вершинами. На каждом шаге агент может переместиться в соседнюю вершину или подождать в своей текущей вершине. Коллизии возникают, когда два агента перемещаются в одну и ту же вершину или пересекают одно и то же ребро на одном и том же шаге. Цель состоит в том, чтобы найти для всех агентов пути к достижению их целей без коллизий, минимизируя при этом сумму затрат, а именно общее количество выполняемых действий. В типичных условиях используется карта в виде сетки, содержащая препятствия и дискретные временные интервалы.

Lifelong multiagent pathfinding (LMAPF) [2] расширяет MAPF, постоянно назначая новые цели агентам. Задача состоит в том, чтобы найти пути без столкновений для всех агентов, максимизируя при этом пропускную способность, а именно среднее количество целей, достигаемых всеми агентами за один шаг.

MAPF-R [2] - это вариант MAPF, где у агентов есть ориентация и они могут поворачиваться. MAPF-R моделирует сетчатую карту с четырьмя соседями, а состояние агента - в виде местоположения и ориентации (восток, юг, запад, север). На каждом шаге агент может переходить к соседней вершине по своему направлению, поворачиваясь на 90 градусов по часовой стрелке или против нее или ожидать в своей текущей вершине.

Задача, представленная здесь, представляет собой комбинацию LMAPF и MAPF-R и называется LMAPF-R.

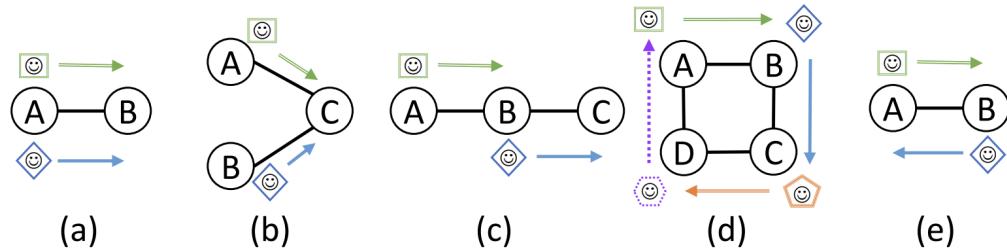


Рис. 1.1: Иллюстрация распространенных типов конфликтов. Слева направо: конфликт ребер, конфликт вершин, следующий конфликт, конфликт циклов и конфликт обмена. Источник: [6]

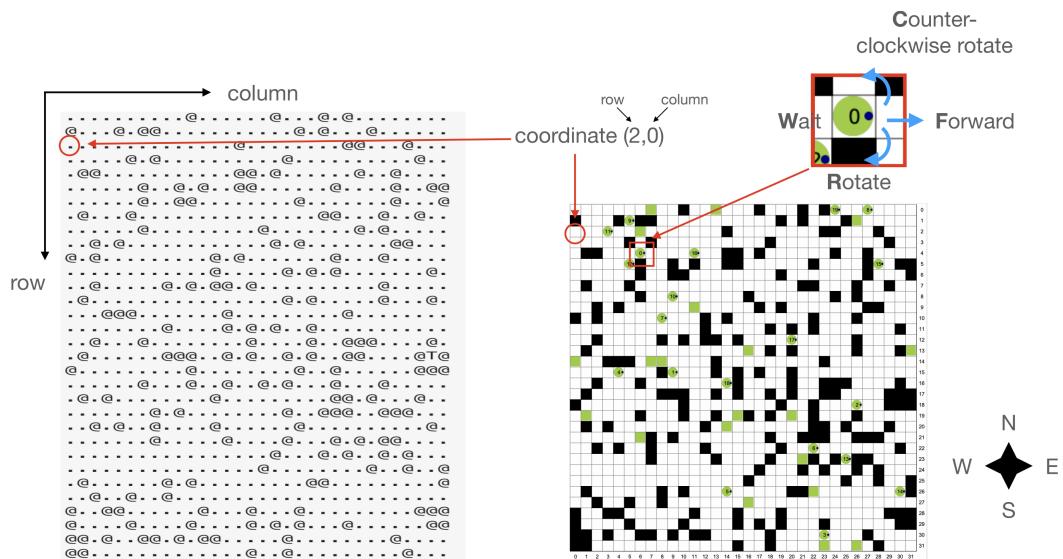


Рис. 1.2: Пример карты, система координат, действия агентов

В LMAPF проблеме поддерживается пул задач. Задача представляет собой последовательность точек на карте, которые нужно посетить. Мы можем назначить агенту одну задачу. Нельзя чтобы нескольким агентам была назначена одна задача.

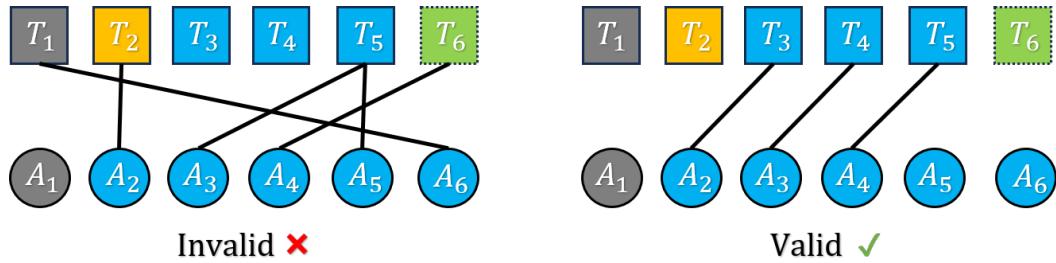


Рис. 1.3: Пример распределения задач по агентам. Задача T_5 назначена обоим агентам A_3 и A_5 , но каждая задача может быть назначена только одному агенту. Задача T_2 , которая уже назначена другому агенту, но еще не завершена, неправильно назначена агенту A_2 . Задача T_6 назначена агенту, что недопустимо, поскольку задачи, которые уже выполнены, не раскрыты или не существуют, не могут быть назначены.

Проблема заключается в назначении агентам задач и последовательности бесконфликтных операций для выполнения как можно большего количества задач. Проблема разбивается на шаги. Внутри каждого шага запускается планировщик задач, который назначает агентам задачи, затем планировщик путей, который назначает агентам их следующее действие. Необходимо написать планировщик задач и планировщик путей.

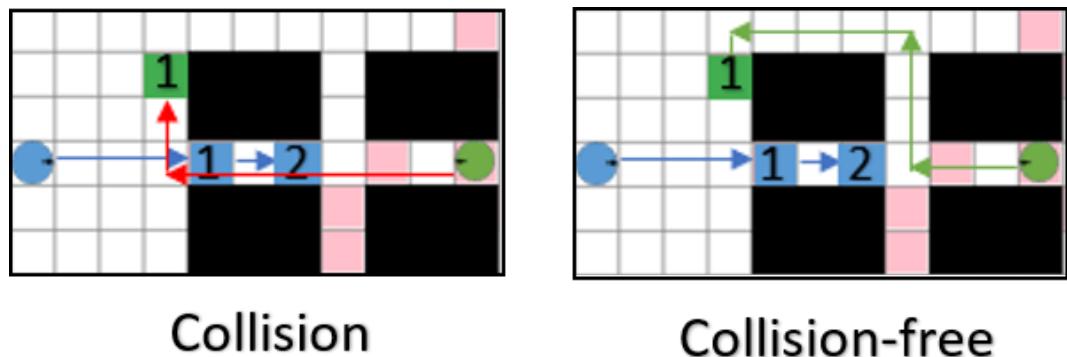


Рис. 1.4: Пример назначения последовательности операций агентам

1.3 Соревнование

С целью проверки эффективности разработанного подхода, руководителем проекта было предложено участие в соревновании The League of Robot Runners¹, так как оно идеально подходит для сравнения результатов с другими участниками и полностью вписывается в данный проект.

Призовой фонд соревнования составляет 11000\$. Длительность соревнования 4 месяца. Спецификация тестирующей системы: AMD EPYC 7R13 Processor with 32 vCPUs, 128 GiB Memory, Nvidia A10G GPU. Тестирование одной посылки занимает чуть больше 7 часов.

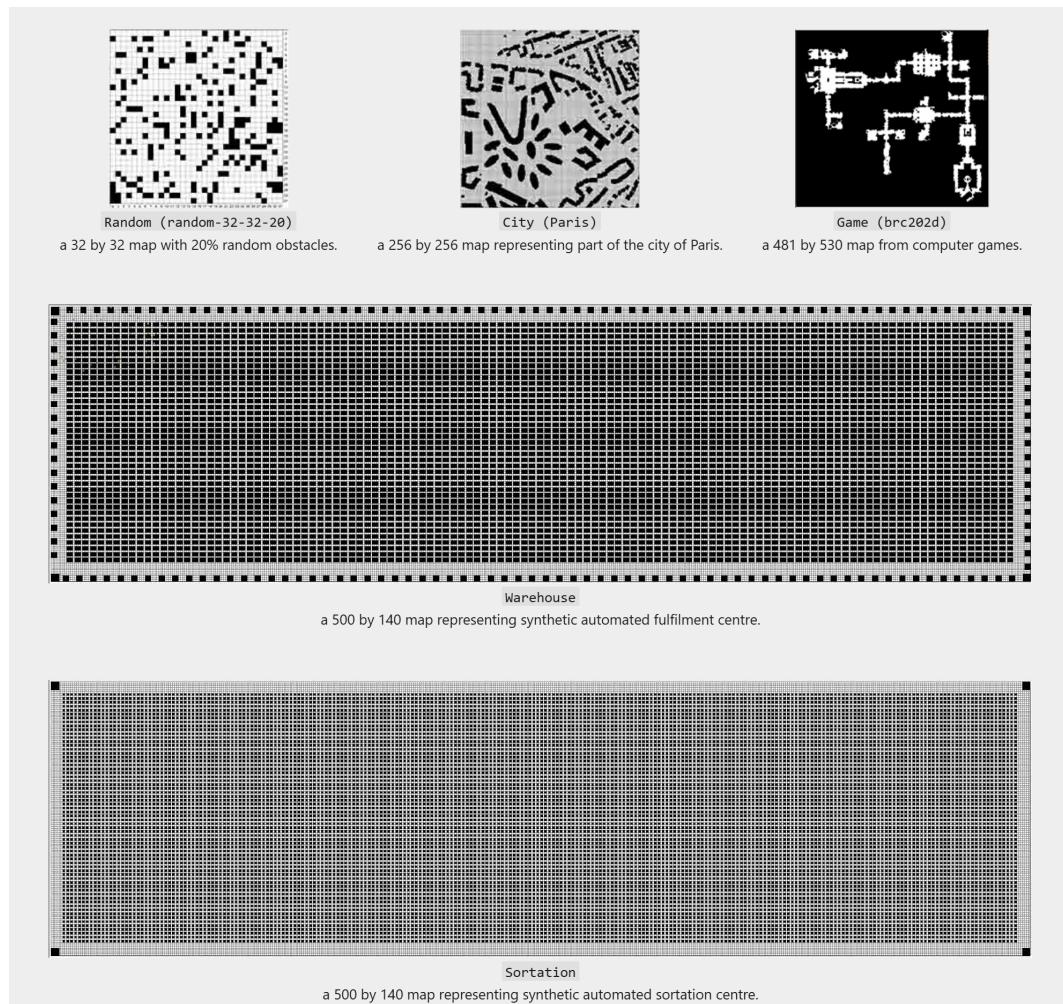


Рис. 1.5: Карты соревнования, на которых запускается решение

¹Сайт соревнования доступен по ссылке: <https://www.leagueofrobotrunners.org/>, дата обр. 11.03.2025

2 Обзор литературы

2.1 Priority Inheritance with Backtracking

Priority Inheritance with Backtracking (PIBT) [3], субоптимальный алгоритм для итеративного решения MAPF. PIBT использует адаптивную систему приоритетов, чтобы сосредоточиться на смежных движениях нескольких агентов. Данный алгоритм хорошо работает даже с тысячами агентов и дает приемлемые решения почти сразу и может решать большие экземпляры, которые не могут решить другие методы MAPF.

2.2 Windowed Parallel PIBT-LNS

Windowed Parallel PIBT (WPP) это улучшение алгоритма PIBT, в котором вместо планирования на глубину 1, используется оконное планирование на некоторую фиксированную глубину w , что позволяет быть агентам более умными и менее близорукими. Так, например, они заранее увидят столкновение в будущем и пойдут другим путем.

Дополнительное улучшение в виде Large Neighborhood Search (LNS) [5], которое решает главную проблему PIBT: приоритеты. Оно умным образом улучшает исходное состояние, полученное алгоритмом. Для сравнения состояний используется некоторая метрика.

Комбинируя эти алгоритмы получаем Windowed Parallel PIBT-LNS (WPPL) [2]. Данний алгоритм использовали прошлогодние победители соревнования.

2.3 Graph Guidance

Graph Guidance [7] грамотно задает веса ребрам графа так, чтобы агенты ходили в одну сторону хорошо, а в другую плохо. Это нужно, чтобы уменьшить столкновения с агентами. Особенно это заметно на картах Warehouse и Sortation. Где на одной линии склада хотелось бы идти только в одну сторону и чередовать их таким образом. Также есть похожий метод SUO [1] Space Utilization Optimization, но он обновляется онлайн, что будет тяжело для данной задачи из-за временных ограничений и большого количества агентов.

2.4 Lazy constraints addition search for MAPF

Этот алгоритм предназначен для MAPF задачи. LaCAM [4] - это двухуровневый поиск. На верхнем уровне он исследует последовательность конфигураций. Каждый узел поиска соответствует одной конфигурации. Для каждого узла высокого уровня он также выпол-

няет низкоуровневый поиск, который создает ограничения. Ограничение определяет, какой агент будет выбран в следующей конфигурации. Низкоуровневый поиск выполняется медленно, создавая минимального преемника каждый раз, когда вызывается соответствующий узел высокого уровня.

Высокоуровневый поиск:

Как и в обычных схемах поиска, в LaCAM выполняется обновление открытого списка, в котором хранятся узлы высокого уровня. Открытый список реализуется структурами данных stack, queue или priority queue. Для каждой итерации поиска LaCAM выбирает один узел из открытого списка. В отличие от обычных схем поиска, LaCAM не отбрасывает выбранный узел сразу.

Низкоуровневый поиск:

Каждый узел высокого уровня содержит конфигурацию и дерево ограничений. Дерево ограничений постепенно увеличивается при каждом обращении к узлу высокого уровня. Это низкоуровневый поиск в LaCAM. Каждый узел дерева ограничений имеет ограничение, за исключением корневого узла.

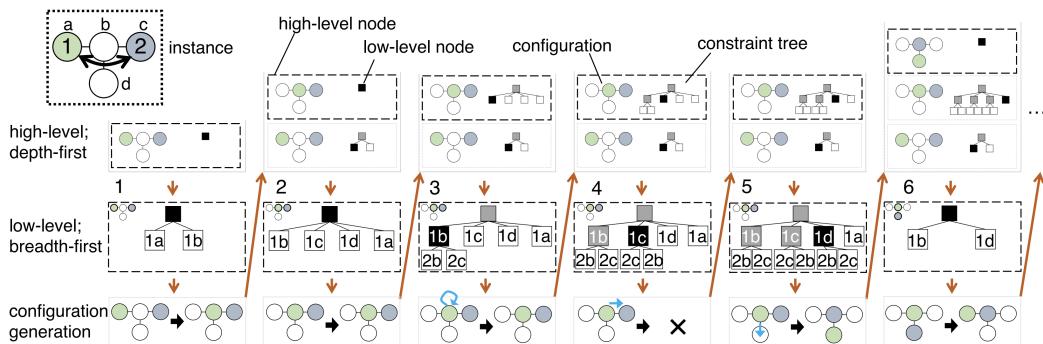


Рис. 2.1: Поясняющая картинка про LaCAM. Источник: [4]

2.5 Вывод обзора литературы

PIBT плох тем, что данный алгоритм изначально создавался для MAPF задачи (без поворотов), то есть PIBT не учитывает повороты, что плохо оказывается на качестве решений версии с поворотами. LaCAM выдает качественные результаты, но он медленный для большого числа агентов в силу огромности дерева состояний и вообще он не для lifelong версии задачи, хотя ему можно ограничить глубину планирования. WPPL хоть и планирует на некоторую глубину, что позволяет уменьшать эффект близорукости, но с увеличением глубины планирования время также возрастает. В связи со всеми этими проблемами, в данном проекте планируется разработать свой подход, который будет работать для огромного числа

агентов, будет менее близоруким и будет выдавать более качественные результаты.

3 Описание предлагаемого метода

3.1 Priority Inheritance with Backtracking

Принцип работы алгоритма PIBT заключается в том, чтобы назначить каждому агенту его желаемое направление для перехода: влево, вправо, вверх или вниз. Изначально мы задаем некоторый порядок на агентах. Например сортируем их по оценочному расстоянию до задачи. Затем проходим их в таком порядке и вызываем рекурсивное построение агента. Допустим мы вызвали рекурсию с агентом r . Мы хотим его построить, для этого каждому направлению для перехода мы назначаем приоритет, например расстояние до задачи после выполнения этого перехода. Затем обходим эти направления в отсортированном порядке. Назначаем агенту направление d . Если в этом направлении никто не стоит, то мы успешно построили всех на пути. Выходим из рекурсии. Иначе в этом направлении кто-то стоит и если он еще не построен, то рекурсивным вызовом попытаемся построить его. Если не получилось, значит идем дальше. Если получилось построить рекурсивно, то все хорошо, выходим из рекурсии.

У этого алгоритма есть один большой недостаток: он изначально был сделан для модели агентов где нет поворотов. То есть когда агент за один шаг может перейти влево, вправо, вверх или вниз. Наша модель задачи более сложная: у нас есть повороты.

3.2 Enhanced Priority Inheritance with Backtracking

В данной работе предлагается модификация алгоритма PIBT, а именно EPIBT, которая в отличие от оригинального подхода рассматривает не одно действие, а последовательность действий. Таким образом это должно было решить проблему PIBT, чтобы он также стал учитывать повороты агента и делать более качественные пути за счет глубины операции.

Закодируем операции: W это ожидание, F переход вперед по направлению, C поворот по часовой, R поворот против часовой. Зафиксируем глубину операций, в моем алгоритме это 5, для примера будет 3. Тогда, мы создаем пул операций вида WWW, CWF, RWF, WWF, ..., FRF, FCF, FFW, FFF. Тут записали все действия и отсортировали их в порядке полезности данной операции: WWW самая бесполезная (мы просто стоим), FFF самая полезная (3 раза идем вперед, желательно к цели). Также для каждой операции op зададим ее приоритет β_{op} .

У меня это номер этой операции в пуле операций, например $\beta_{WWF} = 4$. Допустим у нас есть агент r . Мы знаем его положение на карте, его цель, хотим задать метрику операции w_{op} , тогда $w_{op} = (prev - cur) * \alpha + \beta_{op}$. Где $prev$ = текущее расстояние от агента r до его цели, cur = расстояние от агента r до его цели после выполнения операции op , α = некоторый множитель, у меня он 50.

Тогда в ЕРІВТ изначально у всех агентов выбрана операция WWW. То есть они просто стоят на месте, далее все как в РІВТ: перебираем агентов в отсортированном по приоритету порядке. Взяли агента r , нужно его построить, вызываем рекурсивный вызов построения агента. Сортируем пул операций по w_{op} . Перебираем операции, взяли операцию op . Смотрим на путь, который образует эта операция. Если на этом пути никто не стоит (коллизия), тогда агенту r присваиваем операцию op и возвращаемся из рекурсии с успешным построением. Иначе, этот путь с кем-то коллизит, если это два и более агентов, то пропускаем эту операцию (иначе будет большая сложность рекурсии). Если же там мешает только один агент t , то сносим ему путь, присваиваем операцию op текущему агенту r , вызываем рекурсивное построение для агента t .

Algorithm 1 EPIBT

```
1: Input: graph  $G$ , starts  $\{s_1, \dots, s_n\}$ , goals  $\{g_1, \dots, g_n\}$ 
2: Output: selected actions  $\{d_1, \dots, d_n\}$ 
3: Preface:  $d_i = 0$  for  $i = 1, \dots, n$ .     $\triangleright$  Изначально, все роботы выбрали операцию, где они
   просто стоят
4: Preface:  $P \leftarrow \text{getPath}(s_i, d_i)$  for  $i = 1, \dots, n$   $\triangleright P$  это множество непересекающихся путей
   агентов,  $\text{getPath}(s_i, d_i)$  выдает путь из  $s_i$  с операцией  $d_i$ 
5:  $p_i \leftarrow \text{dist}(s_i, g_i)$ ; for each agent  $i = 1, \dots, n$ 
6:  $A \leftarrow \{1, \dots, n\}$ 
7: sort  $A$  in ascending order of priorities  $p_i$ 
8: for  $i \in A$  do
9:   if  $d_i \neq 0$  then
10:    continue                                 $\triangleright$  агент уже построен
11:    $P \leftarrow P \setminus \text{getPath}(s_i, d_i)$            $\triangleright$  удалим ему путь
12:   if  $\text{EPIBT}(i) = \text{failed}$  then
13:      $P \leftarrow P \cup \text{getPath}(s_i, d'_i)$             $\triangleright$  попытаемся построить новый
                                                                $\triangleright$  не получилось, вернем обратно
14: procedure  $\text{EPIBT}(i)$ 
15:    $C \leftarrow op \in \text{Operations}$ 
16:   sort  $C$  in descending order of  $w_{op}$ 
17:   for  $op \in C$  do
18:     if  $\text{getPath}(s_i, op) \not\subset G$  then  $\triangleright$  данная операция оп при выполнении из  $s_i$  выйдет за
      пределы графа (врежется в стену, выйдет за карту)
19:     continue
20:     if  $\text{getUsed}(s_i, op, P) = \emptyset$  then  $\triangleright$   $\text{getUsed}$  выдаст агентов, с которыми мы врежемся,
      если стартуем из  $s_i$  и выполним операцию  $op$ 
21:        $d_i \leftarrow op$                              $\triangleright$  присвоит агенту  $i$  операцию  $op$ 
22:        $P \leftarrow P \cup \text{getPath}(s_i, op)$          $\triangleright$  добавить этот путь
23:       return success                          $\triangleright$  мы ни с кем не врежемся
24:     if  $|\text{getUsed}(s_i, op, P)| \geq 2$  then
25:       continue                                 $\triangleright$  путь задевает большое число агентов
26:        $j \in \text{getUsed}(s_i, op, P)$             $\triangleright$  мы коллизим только с одним агентом. возьмем его
27:        $P \leftarrow P \setminus \text{getPath}(s_j, d_j)$          $\triangleright$  снесем путь, мешающему агенту
28:        $P \leftarrow P \cup \text{getPath}(s_i, op)$             $\triangleright$  поставим путь агенту  $i$ 
29:        $d_i \leftarrow op$ 
30:     if  $\text{EPIBT}(j) = \text{success}$  then
31:       return success                          $\triangleright$  мы смогли рекурсивно построить агентов
                                                                $\triangleright$  не смогли, вернем как было
32:        $P \leftarrow P \setminus \text{getPath}(s_i, d'_i)$          $\triangleright$  удалим путь агенту  $i$ 
33:        $P \leftarrow P \cup \text{getPath}(s_j, d_j)$             $\triangleright$  вернем старый путь агенту  $j$ 
34:        $d_i \leftarrow d'_i$ 
35:   return failed
```

3.3 Enhanced Priority Inheritance with Backtracking Large Neighborhood Search

Также в данной работе предлагается модификация EPIBT, которая решает главный недостаток алгоритма PIBT: приоритет. В таких алгоритмах от приоритетов зависит качество решения.

EPIBT+LNS задает некоторую метрику состоянию всех агентов. У меня это $\sum_r w_{opr}$, где $opr =$ операция, которую выбрал агент r . Чем больше это число, тем лучше агенты движутся к цели.

Изначально запускается EPIBT. Далее для улучшения метрики применяется LNS (алгоритм имитации отжига). На каждом шаге EPIBT+LNS мы берем рандомного агента и пытаемся рекурсивно его перестроить с помощью метода аналогичной EPIBT.

3.4 Graph Guidance

Были построены GG для всех карт соревнования, хоть и довольно простые. Так для карт Warehouse и Sortation линии складов были распределены так, чтобы на линии 1 агенты шли влево, на линии 2 агенты шли вправо, и так далее. А для random карты руками был подобран качественный Graph Guidance.

3.5 Эксперименты

Здесь представлены следующие алгоритмы: PIBT, PIBT+traffic flow (дефолтный планировщик с соревнования), EPIBT (мой новый алгоритм), EPIBT+LNS (моё итоговое решение соревнования), WPPL (алгоритм победителей прошлого года).

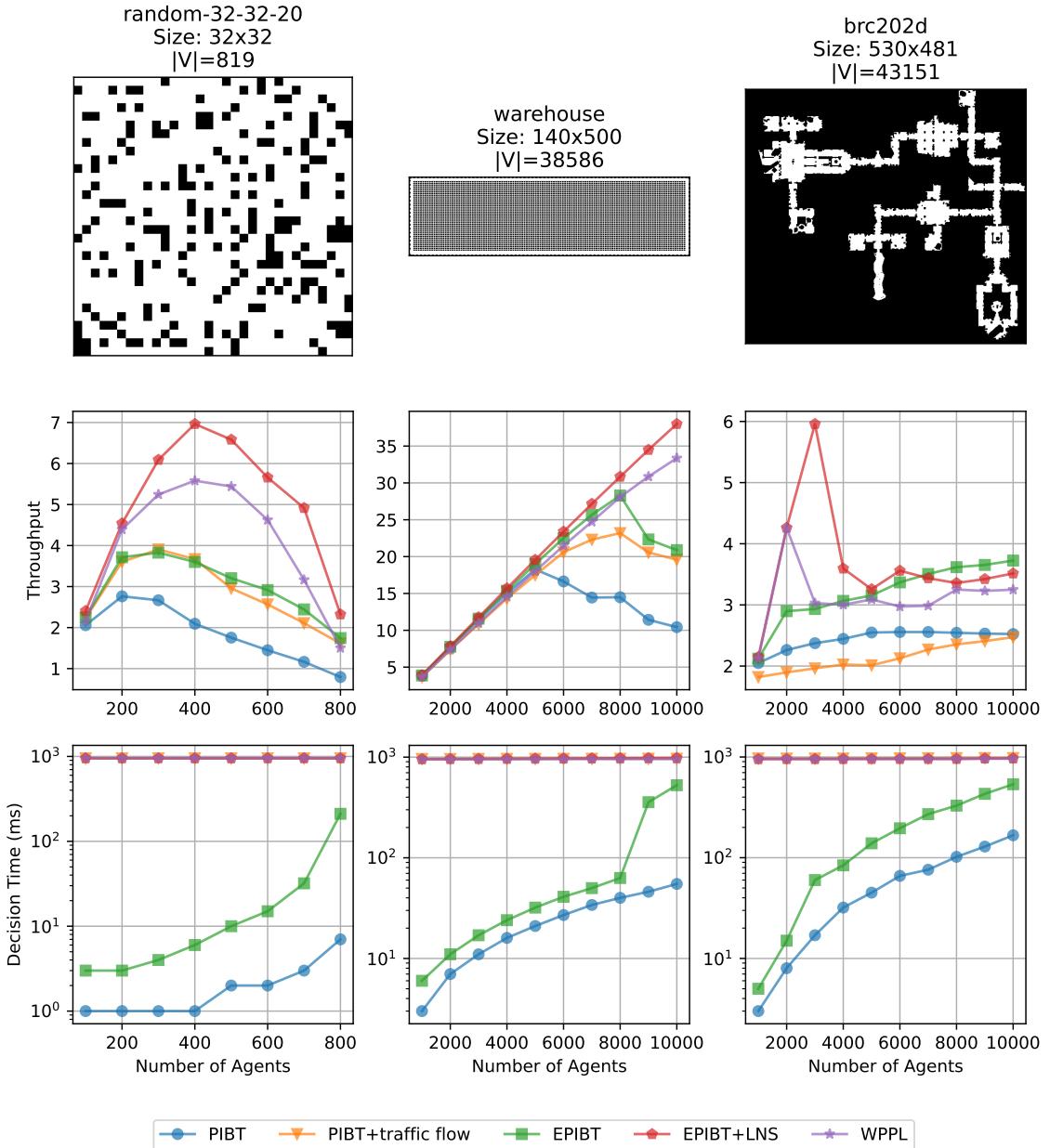


Рис. 3.1: На графиках по оси X расположено количество агентов. По вертикали сверху вниз идет информация о карте, сама карта, далее график пропускной способности для каждого алгоритма, затем среднее время на вычисление одного шага. У алгоритмов PIBT+traffic flow, EPIBT+LNS и WPPL среднее время на вычисление одного шага равно 1 секунде, потому что эти алгоритмы улучшают решение, пока у них есть на это время. На графиках видно, что EPIBT+LNS обходит WPPL

На рисунках ниже представлены результаты алгоритмов на карте random 32x32. Количество агентов: 400. На рисунках расположены карты. Каждая из карт имеет подпись: направление (East, South, West, North и All) и действие (ForWard, Rotate, Clockwise Rotate, Wait и All). Каждая клетка такой карты имеет значение равное нормализованному количеству событий, когда такое происходило: агент стоит в этой точке с этим направлением и делает такое действие или черный цвет, если это стена. Была подобрана хорошая цветовая палитра, которая выделяет интересные места и приятна глазу.

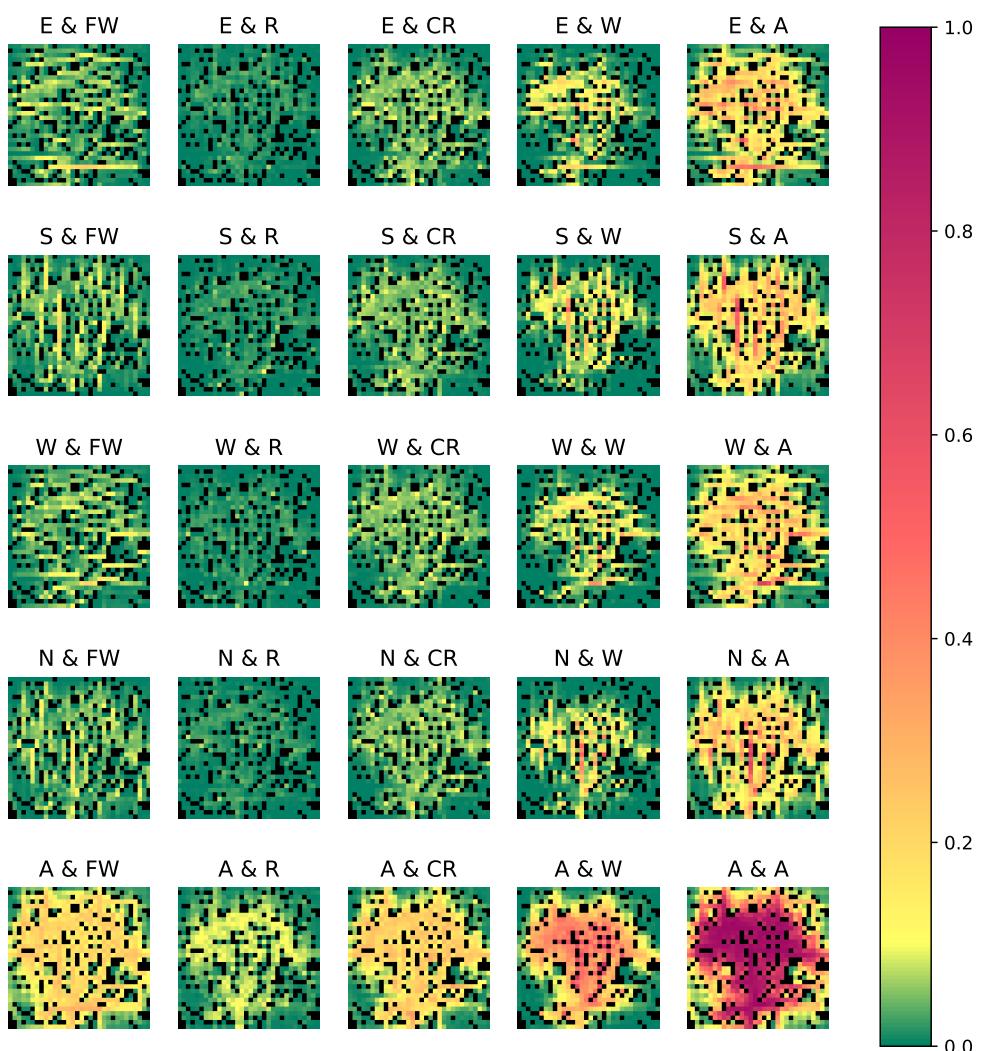


Рис. 3.2: Планировщик: PIBT, throughput: 2.093. На графике видно, что агенты столпились в центре карты: A&W желто-красное, то есть они там часто стояли и просто ждали, а также много поворачивались: A&R, A&CR. Результат очень плохой.

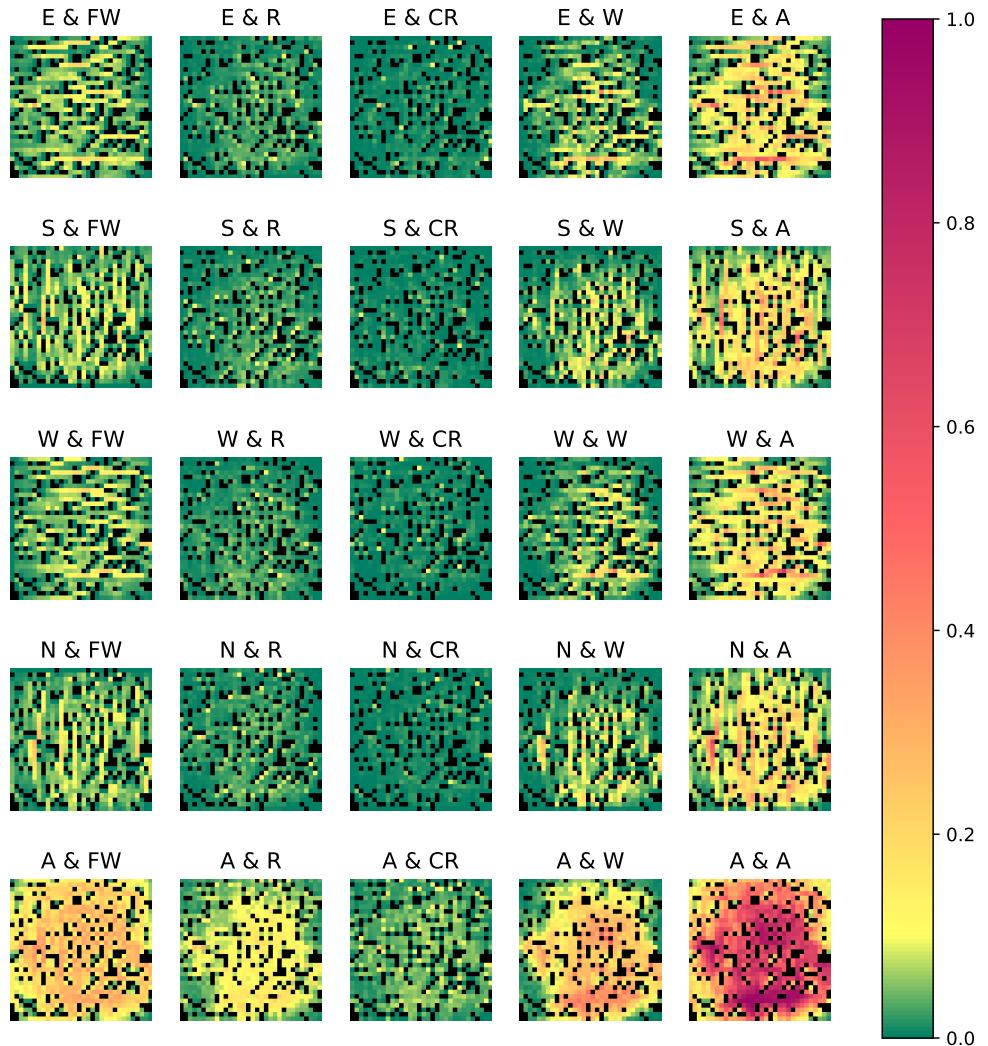


Рис. 3.3: Планировщик: PIBT+traffic flow, throughput: 3.669. На графике видно, что агенты все еще стоят, но уже поменьше и распределены от центра карты получше. А также меньше поворачиваются и побольше ходят вперед к цели.

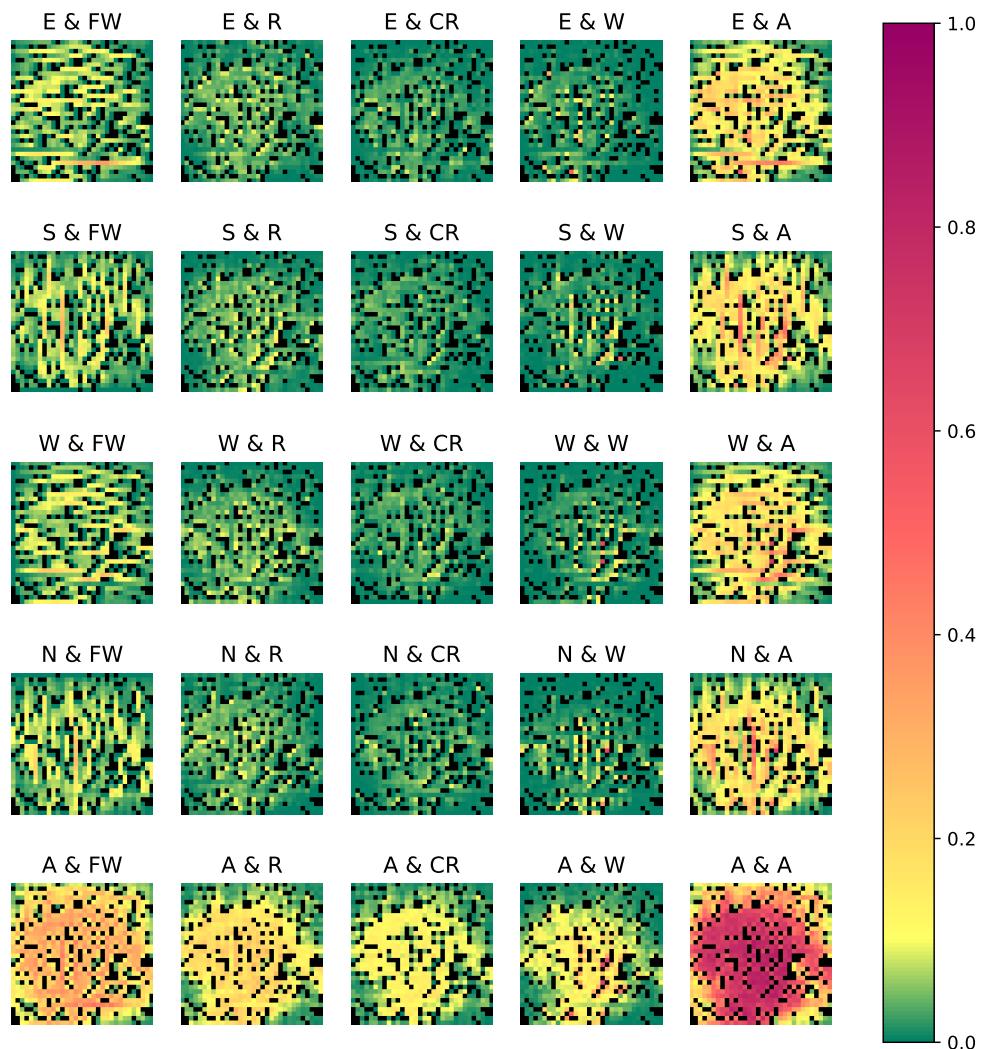


Рис. 3.4: Планировщик: ЕРІВТ, throughput: 3.6. На графике видно, что агенты уже практически не стоят на месте, но вот побольше поворачиваются, что все также плохо.

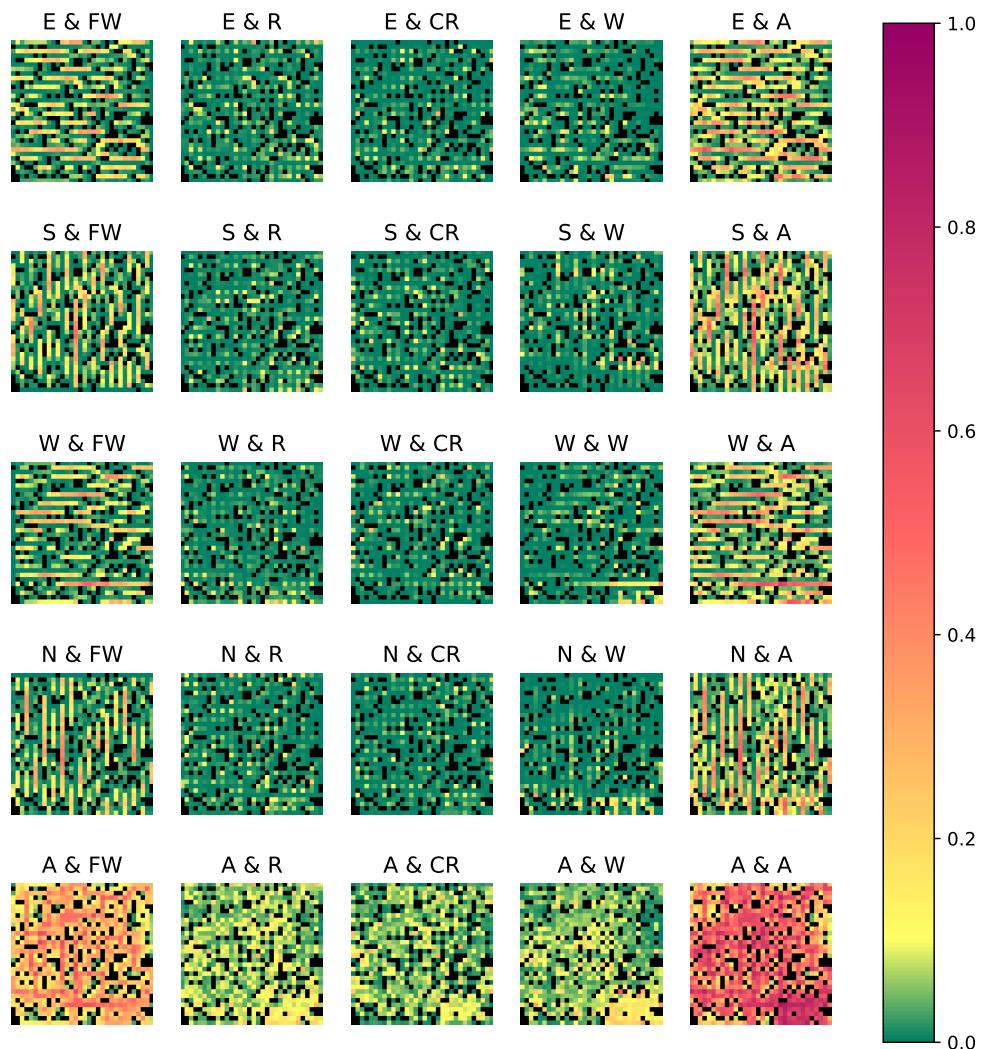


Рис. 3.5: Планировщик: WPPL, throughput: 5.581. На графике видно, что агенты равномерно распределились по карте, но застряли в правом-нижнем углу и там ждали и поворачивались.

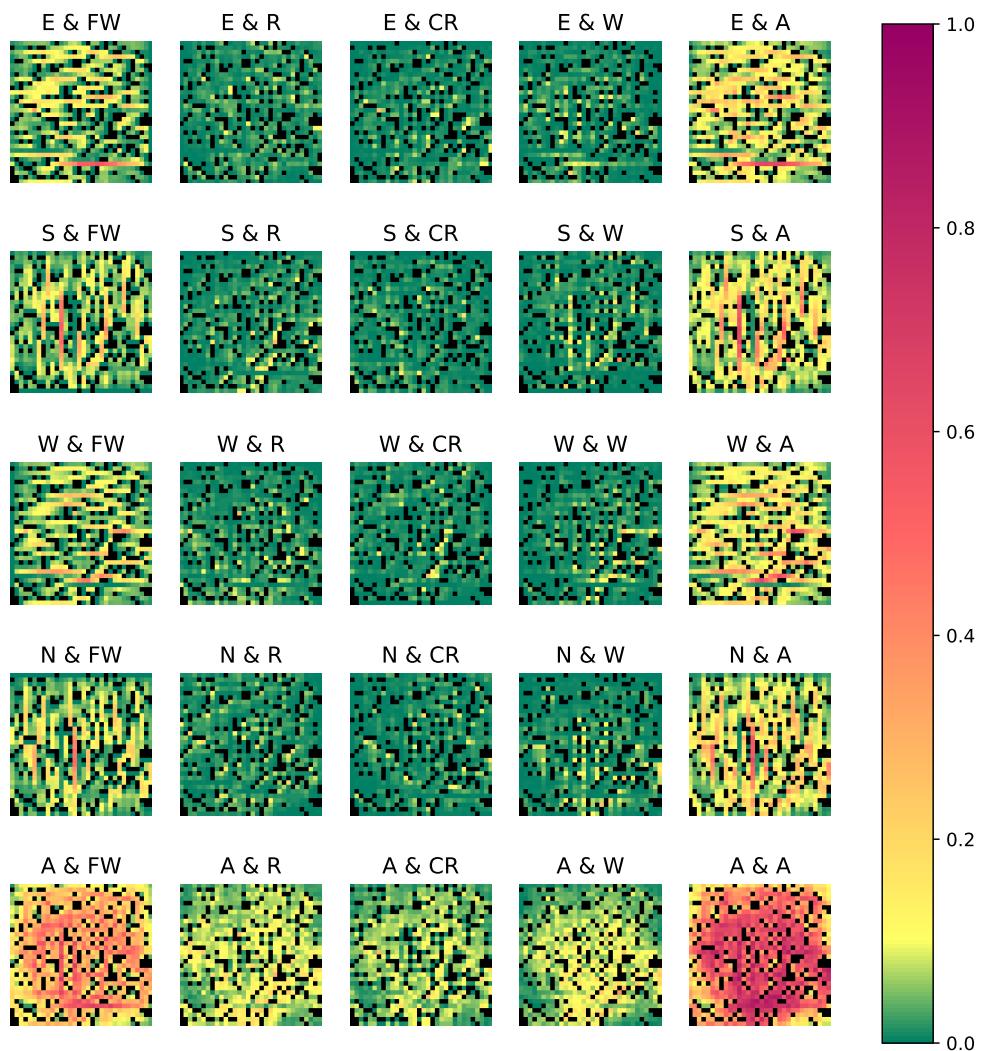


Рис. 3.6: Планировщик: EPIBT+LNS, throughput: 6.964. На графике видно, что агенты практически не стоят и не поворачиваются. Они распределились по карте. Результат очень хороший.

На рисунках ниже представлены результаты алгоритмов на карте warehouse 140x500.

Количество агентов: 10000. Это синтетический склад, много различных путей от пункта А, до Б.

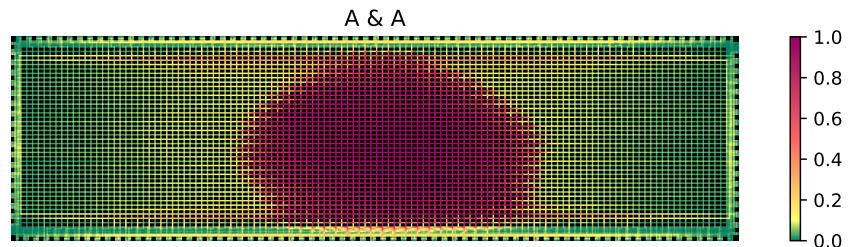


Рис. 3.7: Планировщик: PIBT, throughput: 10.4294. На графике видно огромное пятно из агентов, которые там столпились и плохо ходили. Результаты очень плохие.

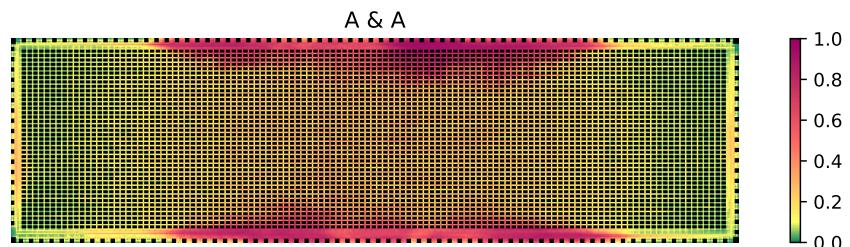


Рис. 3.8: Планировщик: PIBT+traffic flow, throughput: 19.585. На графике видно, что агенты неплохо распределились по складу, но все же есть пятна сверху и снизу.

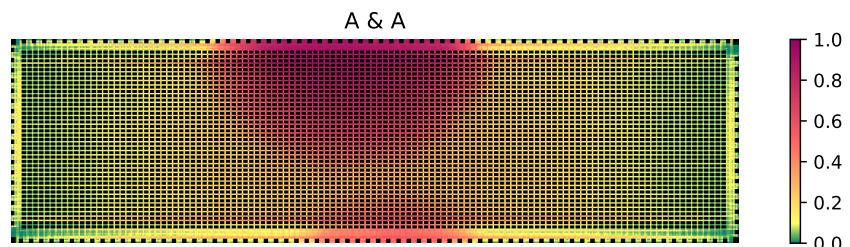


Рис. 3.9: Планировщик: EPIBT, throughput: 20.8818. На графике видно, что агенты также неплохо распределились по карте, но есть большое пятно.

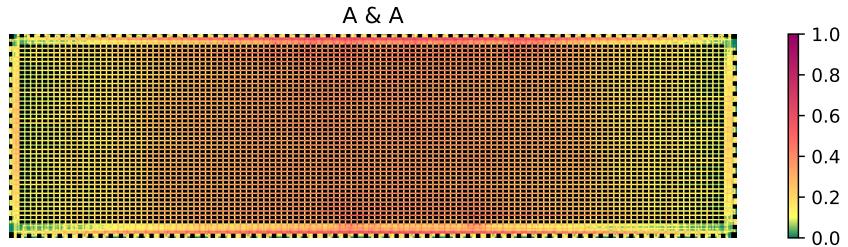


Рис. 3.10: Планировщик: WPPL, throughput: 33.3594. На графике не видно никаких пятен, агенты равномерно распределились по карте, хорошо выполняют задачи.

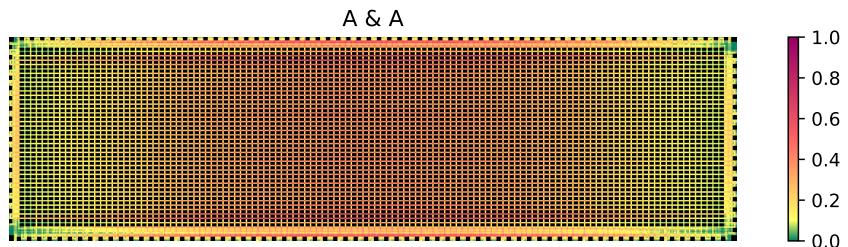


Рис. 3.11: Планировщик: EPIBT+LNS, throughput: 37.9986. Аналогично WPPL, но результаты еще лучше.

На рисунках ниже представлены результаты алгоритмов на карте brc202d 530x481.

Количество агентов: 3000. Это довольно специфичная карта: много агентов, но узкие проходы и куча коридоров.

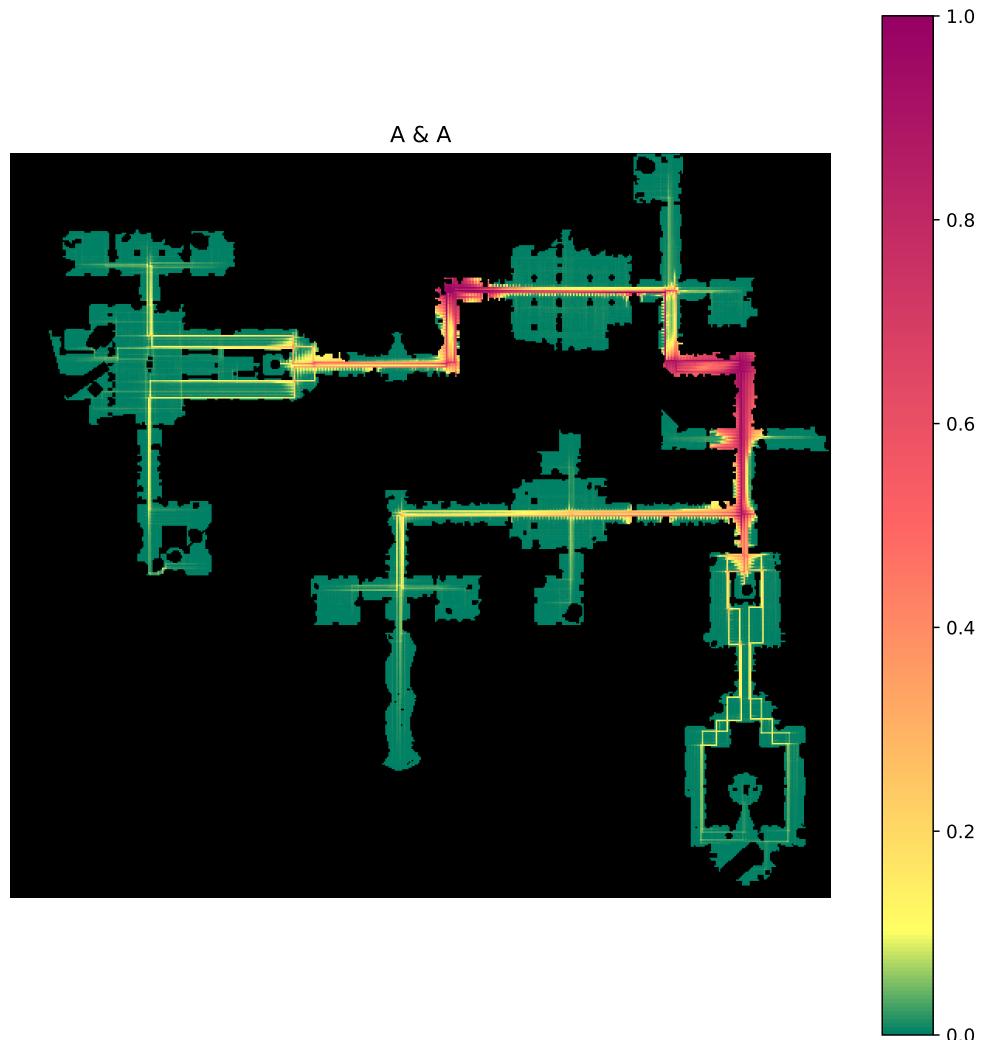


Рис. 3.12: Планировщик: PIBT, throughput: 2.3736. На графике видно, что агенты застряли в самом коридоре, результаты очень плохие.

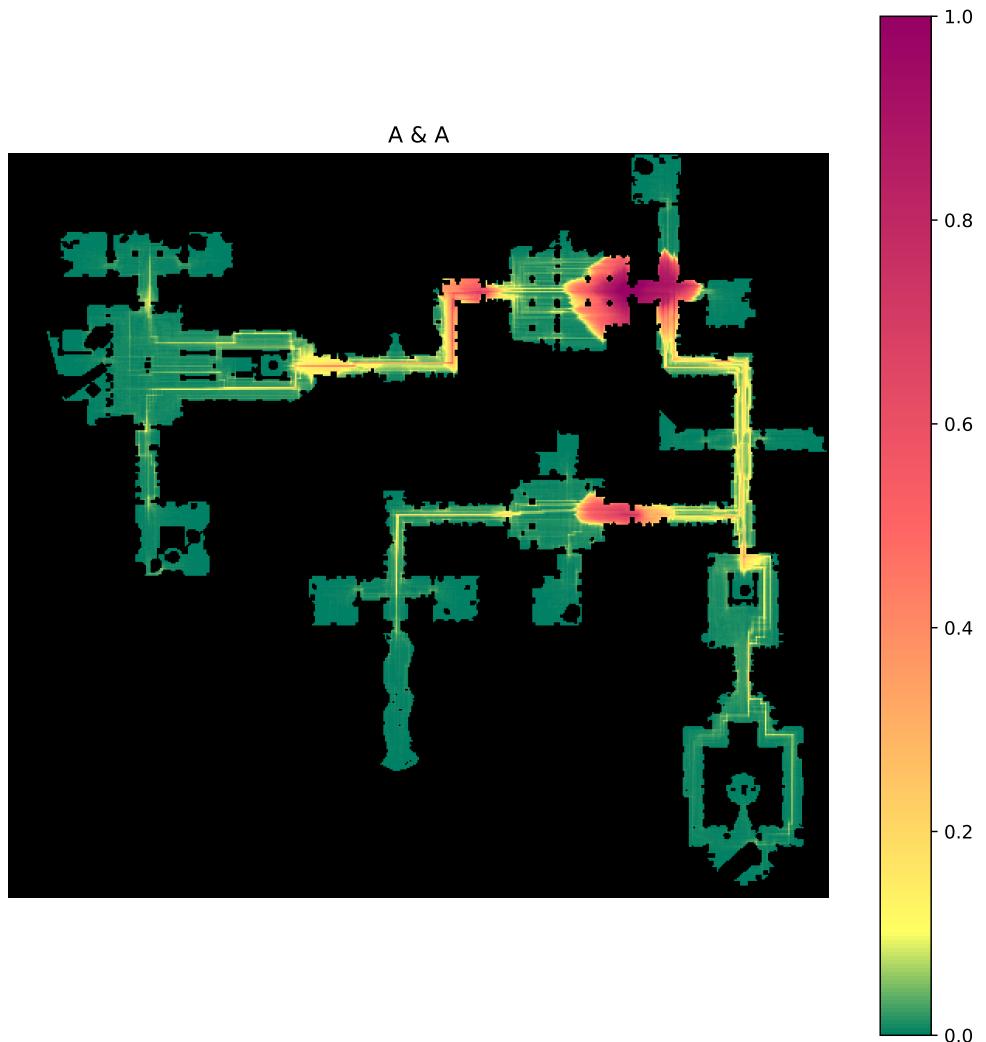


Рис. 3.13: Планировщик: PIBT+traffic flow, throughput: 1.9622. На графике видно, что агенты неплохо распределились по коридорам, но все еще застряли.

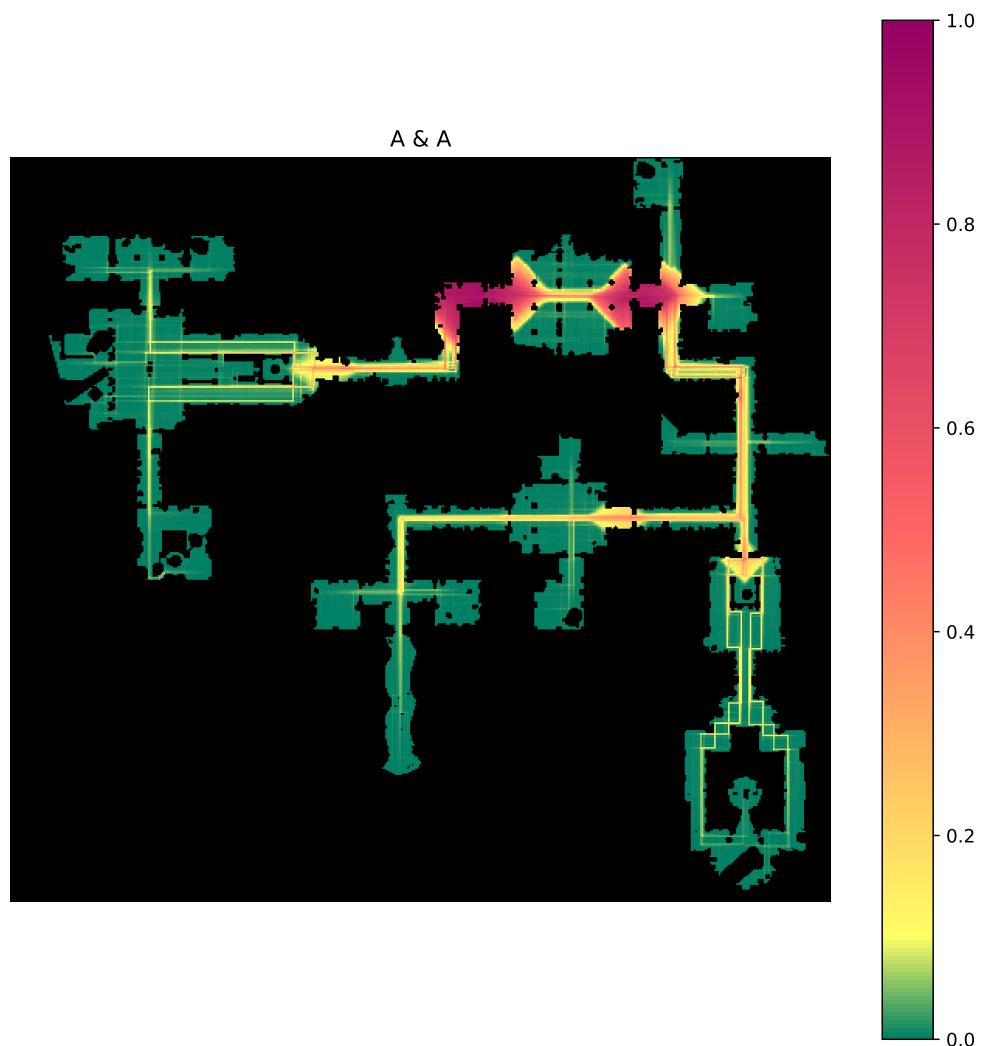


Рис. 3.14: Планировщик: EPIBT, throughput: 2.9328. На графике видно, что агенты довольно хорошо ходили, но застряли в двух узких проходах.

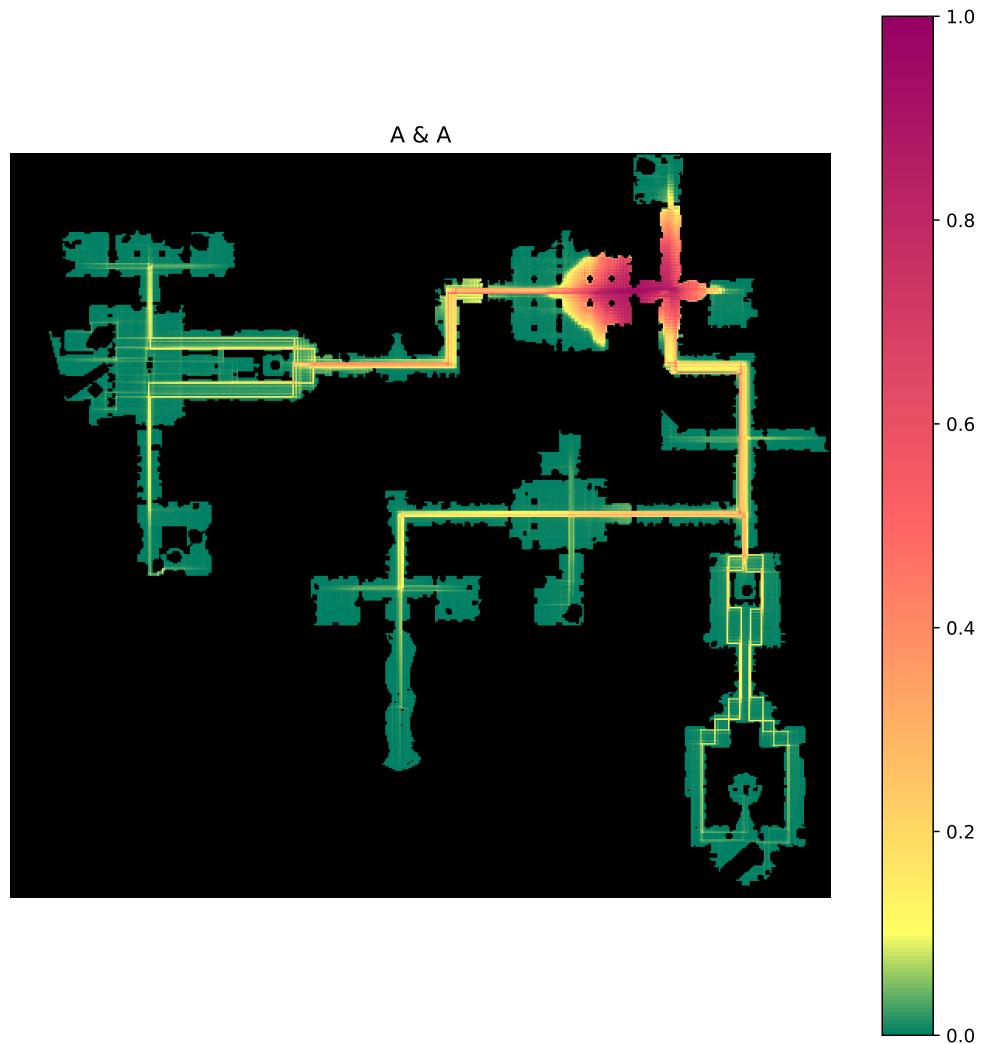


Рис. 3.15: Планировщик: WPPL, throughput: 3.03. На графике видно, что алгоритм не справился с управлением, много агентов столпились у узкого прохода. Результаты плохие.

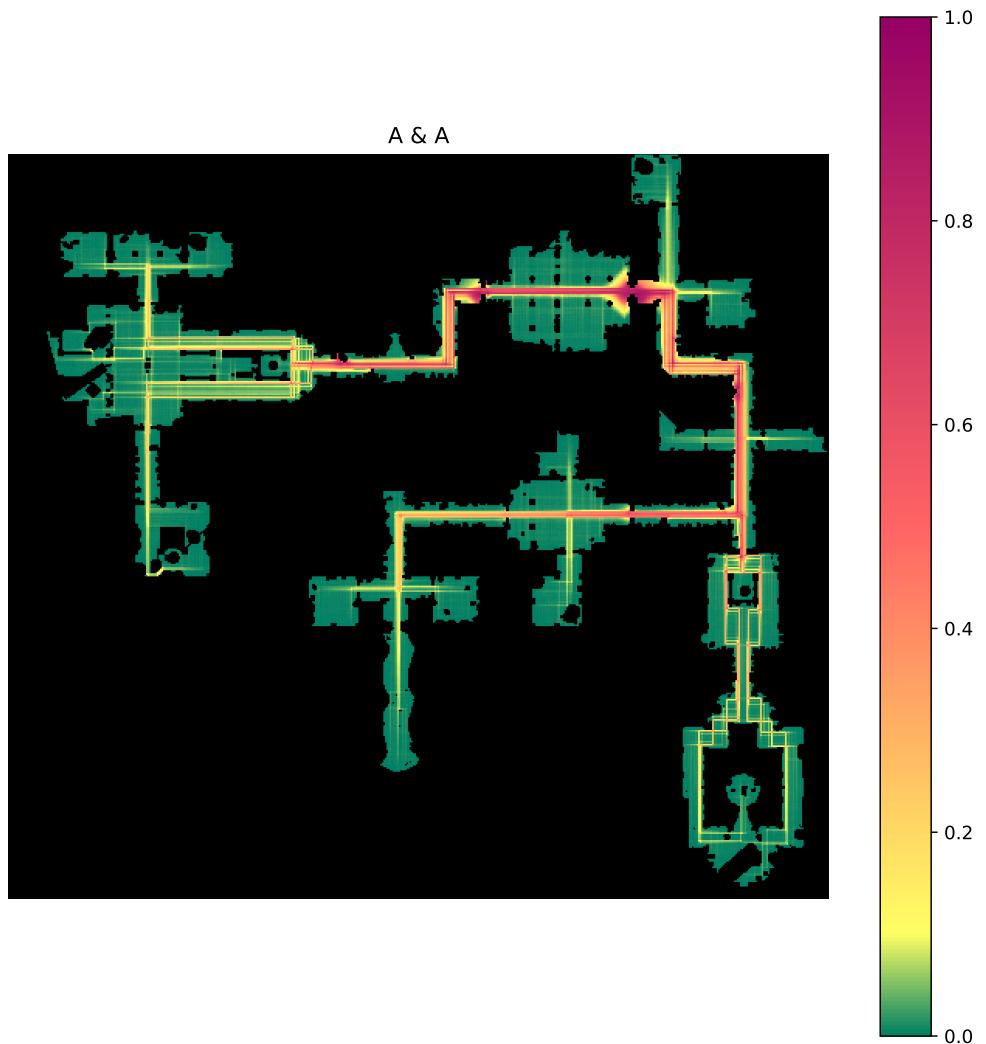


Рис. 3.16: Планировщик: EPIBT+LNS, throughput: 5.959. На графике ярко выражены трассы, по которым агенты много проходили. Здесь они практически не ждали. Результаты очень хорошие

4 Результаты

Здесь представлены результаты соревнования 1.3. Tasks это количество выполненных задач. Score это Tasks деленный на максимальный Tasks всех посылок. В Planner Track нужно было предоставить планировщик путей, а в качестве планировщика задач был стандартный. В Scheduler Track наоборот: нужно было предоставить планировщик задач и использовался стандартный планировщик путей. В Combined Track нужно было реализовать все вместе. И Line Honours это количество тестов, где у твоей команды лучшее решение.



Finally, we have the **Line Honours** award, which recognises the team with the largest number of best-known solutions across all problem instances. The winner of this award is: 🥇 Team No Man's Sky.

Рис. 4.1: Результаты соревнования

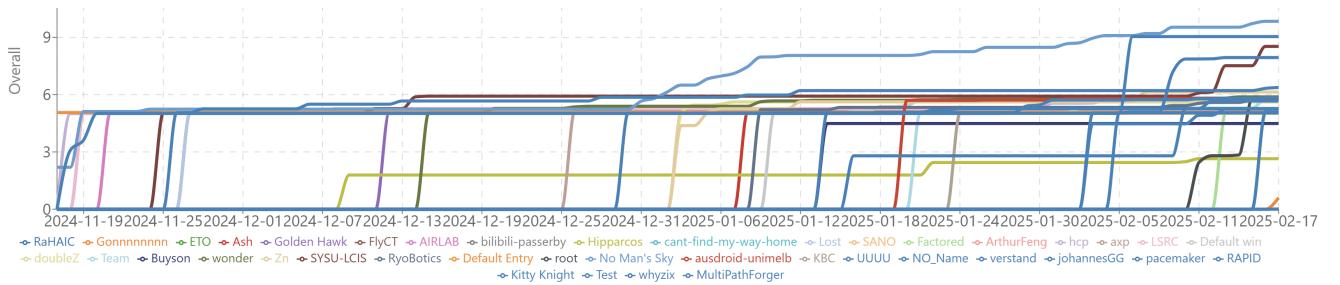


Рис. 4.2: График изменений результатов соревнования. Каждая линия это отдельная команда, на протяжении половины соревнования мы были лидером с большим отрывом

Таблица 4.1: Результаты нашей лучшей посылки. Мы получили лучшее решение в тестах GAME, RANDOM-05, SORTATION, WAREHOUSE, а также в другой посылки RANDOM-03 и RANDOM-04. Итого у нас в 6-и тестах из 10-и лучшее решение.

Instance	Tasks	Score
CITY-01	8420	0.997
CITY-02	16787	0.987
GAME	23274	1
RANDOM-01	639	0.918
RANDOM-02	1221	0.969
RANDOM-03	2334	0.986
RANDOM-04	2547	0.985
RANDOM-05	3050	1
SORTATION	152714	1
WAREHOUSE	154795	1
Total	365781	9.842

5 Выводы

Целью данного проекта было разобраться в LMAPF задаче, алгоритмах и методах ее решения, написать свой подход, улучшить существующие решения. Планировалось выиграть в соревновании, чтобы доказать, что решение данного проекта оказалось достаточно хорошим и у нас это вышло. Мы команда No Man's Sky и мы выиграли во всех четырех треках с большим отрывом и получили 11000\$. Также удалось создать новый алгоритм EPIBT и добавить в него LNS оптимизатор, итоговый алгоритм EPIBT+LNS, который хорошо себя показал как в соревновании, так и в экспериментах и побил решение победителей прошлого года WPPL [2].

Список литературы

- [1] Shuai D Han и Jingjin Yu. “Optimizing space utilization for more effective multi-robot path planning”. B: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, c. 10709—10715.
- [2] He Jiang, Yulun Zhang, Rishi Veerapaneni и Jiaoyang Li. “Scaling Lifelong Multi-Agent Path Finding to More Realistic Settings: Research Challenges and Opportunities”. B: *Proceedings of the International Symposium on Combinatorial Search*. T. 17. 2024, c. 234—242.
- [3] Toshihiro Matsui. “Investigation of Heuristics for PIBT Solving Continuous MAPF Problem in Narrow Warehouse.” B: *ICAART (1)*. 2024, c. 341—350.
- [4] Keisuke Okumura. “Lacam: Search-based algorithm for quick multi-agent pathfinding”. B: *Proceedings of the AAAI Conference on Artificial Intelligence*. T. 37. 10. 2023, c. 11655—11662.
- [5] Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov и Vinod Nair. “Learning a large neighborhood search algorithm for mixed integer programs”. B: *arXiv preprint arXiv:2107.10202* (2021).
- [6] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar и др. “Multi-agent pathfinding: Definitions, variants, and benchmarks”. B: *Proceedings of the International Symposium on Combinatorial Search*. T. 10. 1. 2019, c. 151—158.
- [7] Yulun Zhang, He Jiang, Varun Bhatt, Stefanos Nikolaidis и Jiaoyang Li. “Guidance Graph Optimization for Lifelong Multi-Agent Path Finding”. B: *arXiv preprint arXiv:2402.01446* (2024).