

УДК 004.896

doi: 10.15622/rcai.2025.047

## АЛГОРИТМЫ МНОГО-АГЕНТНОГО ПЛАНИРОВАНИЯ ТРАЕКТОРИИ С УЧЕТОМ ВРЕМЕНИ, ЗАТРАЧИВАЕМОГО НА ПОВОРОТЫ

А.И. Луценко (*ailutsenko@edu.hse.ru*)<sup>А</sup>

К.С. Яковлев (*yakovlev@isa.ru*)<sup>А,В</sup>

<sup>А</sup> Национальный исследовательский университет

«Высшая школа экономики», Москва

<sup>В</sup> Федеральный исследовательский центр

«Информатика и управление» РАН, Москва

В работе рассматривается задача много-агентного планирования путей на графе регулярной декомпозиции. В частности, исследуется один из возможных вариантов постановки этой задачи, а именно, вариант, когда агенты затрачивают на поворот время, равное времени, затрачиваемому на перемещение в смежную вершину. Предлагаются несколько оригинальных алгоритмов решения задачи в этой постановке, являющихся модификациями известного вычислительно-эффективного алгоритма много-агентного планирования РИВТ. Приводятся результаты экспериментальных исследований.

**Ключевые слова:** много-агентное планирование, граф, робот, кратчайший путь, поиск пути в графе.

### Введение

Задача много-агентного планирования путей – это задача, в которой по графу и множествам начальных и целевых вершин необходимо построить совокупность путей для нескольких агентов таким образом, что агенты во время следования вдоль этих путей не сталкиваются между собой.

Много-агентное планирование путей имеет множество приложений, например, в организации беспилотного транспорта, но ее самое частое применение сегодня – организация совместной работы роботов на автоматизированных складах, когда им надо перевозить товары из одного места в другое.

Задача уже широко изучена и существует множество алгоритмов, ее решающих. Так, например, популярным оптимальным алгоритмом является CBS [Sharon et al., 2015]. На его основе разработаны алгоритмы

ECBS [Barer et al., 2014] и EECBS [Li et al., 2021], которые находят решение, отличающееся от оптимального в фиксированное число раз, но при этом данное решение находится быстрее, чем в CBS.

Популярной идеей является построение алгоритмов на правилах: для агентов составляются правила, по которым они могут действовать (например, правило, обменивающее агентов местами) и с помощью этих правил задача решается. Такая идея применяется в статьях [De Wilde et al., 2014], [Sajid et al., 2012] и [Khorshid et al., 2011]. Решение находится быстро, но часто сильно отличается от оптимального в смысле качественных характеристик.

Предложенные алгоритмы хороши в своих областях, однако, они пренебрегают некоторыми физическими параметрами системы и решения получаются неприменимыми на практике. В данной работе рассматривается усложненная задача много-агентного планирования – много-агентное планирование с поворотами. Необходимость рассмотрения этой задачи диктуется практикой, т.к. в реальности большинство складских роботов затрачивают время на поворот на месте. Следовательно если пути для подобных роботов конструировались без учета этого обстоятельства, то безопасное следование по ним невозможно.

Для много-агентного планирования с поворотами уже был предложен оптимальный алгоритм CBS [Zhang et al., 2023], однако данный алгоритм из-за того, что является оптимальным, решает задачу за длительное время даже для небольшого числа агентов. В работе предлагаются алгоритмы PIBT<sub>T</sub>, Turned PIBT и WinPIBT<sub>T</sub>, основанные на алгоритме PIBT [Okumura et al., 2022], который пошагово решает задачу много-агентного планирования без поворотов. Он выбран за основу данных алгоритмов, так как решает задачу быстро, получая при этом не слишком плохие по качеству решения.

## 1. Постановка задачи

Определим задачу MAPF<sub>T</sub> – задачу много-агентного планирования с поворотами.

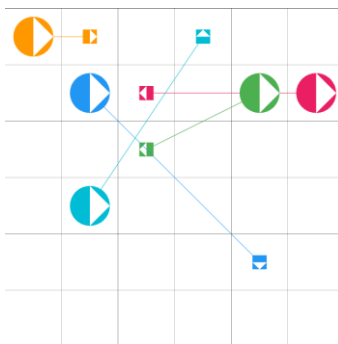


Рис. 1. Пример постановки задачи: круги – начальные положения агентов, квадраты – конечные их положения

Ее вход можно задать тройкой  $(x, y, d)$ , где  $(x, y)$  – неориентированный граф,  $(x, y)$  – стартовые позиции агентов, а  $d$  – конечные позиции агентов.

Граф в этой задаче задается на квадратной решетке, то есть  $G$ , а множество соседей вершины  $v$  является подмножеством  $N(v)$ , т.е. состоит из ребер вида  $(v, v')$  и  $(v', v)$ .

Определим множество возможных направлений  $D$ .  
 Ответ задается набором из путей.  $P$  – набор из пар  $(v, d)$  – текущая вершина и направление агента.

Правила для  $P$ :

1. Для любого  $(v, d) \in P$  и  $(v', d') \in P$  верно одно из:

1.1.  $(v, d) = (v', d')$  и переход происходит в направлении  $d$ , то есть если  $(v, d) \in P$ , то  $(v', d') \in P$ , а для других направлений правила аналогичны.

1.2.  $(v, d) \neq (v', d')$ , но при этом направления  $d$  и  $d'$  отличаются на  $90^\circ$ , то есть одно имеет вид  $(v, d)$ , а другое –  $(v, d')$ .

1.3.  $(v, d) \neq (v', d')$ , то есть агент не двигается.

2. Для любого  $(v, d) \in P$  верно, что  $(v, d) \in P$ , а  $(v, d)$  – любое направление из  $D$ , то есть задает начало пути  $i$ -ого агента, а  $(v, d)$  – конец его пути.

3. Для любых двух агентов  $i$  и  $j$ ,  $(v, d) \in P$  в любой момент времени  $t$  верно, что  $(v, d) \in P$ , то есть никакие два агента не находятся в одной вершине графа одновременно.

4. Для любых двух агентов  $i$  и  $j$ ,  $(v, d) \in P$  в любой момент времени  $t$  не могут одновременно выполняться условия  $(v, d) \in P$  и  $(v, d) \in P$ , то есть никакие два агента не могут одновременно проходить по одному ребру (даже в разных направлениях).

При этом задачей алгоритмов является минимизация следующих метрик:

- *Makespan* – минимальная длина пути, то есть  $\min_i \{L_i\}$ .
- *Sum-of-costs* – суммарная длина всех путей, то есть  $\sum_i L_i$ .

## 2. Алгоритмы

В основе предложенных алгоритмов лежит алгоритм PIBT [Okumura et al., 2022]. В ходе работы данного алгоритма происходит пошаговое планирование положений агентов. Агентам сопоставляются случайные приоритеты, распределенные равномерно на отрезке  $[0, 1]$ . Каждый ход

приоритет агента возрастает на 1, а когда он достигнет своей цели, целая часть приоритета отбрасывается (остается лишь изначально случайно сгенерированная величина). В его основе лежат техники Priority Inheritance и Backtracking. Ход выбирается следующим образом: Сначала с помощью алгоритма  $A^*$  для агента определяется следующая клетка на его кратчайшем пути до цели. Алгоритм пытается передвинуть в эту клетку. Если она уже занята каким-либо другим агентом, то происходит Priority Inheritance: этот агент временно получает приоритет агента и алгоритм также пытается найти ему клетку, в которую можно перейти, возможно, применяя Priority Inheritance. После этого, агент сообщает об успешности передвижения агенту. Такое обратное взаимодействие агента и агента называется Backtracking. В случае неуспешного завершения Priority Inheritance, начинает искать себе другую клетку, в которую он может перейти. Данный алгоритм хорошо показывает себя на практике: решает задачу быстро и предъявляет хорошие по качеству решения

## 2.1. Turned PIBT

Чтобы получить алгоритм, который решает задачу в хорошем числе случаев, предлагается взять задачу  $MARF_T$ , убрать из нее направления, получив тем самым задачу  $MARF$ , решить ее любым решателем, например, PIBT, и вернуть в полученное решение направления.

---

**Алгоритм 1** Преобразователь решения  $MARF$  в решение  $MARF_T$

---

**Вход:**  $\pi$  — решение  $MARF$  (набор путей,  $\pi_i[t]$  — положение агента  $i$  в момент  $t$ )

**Выход:**  $\pi_T$  — решение  $MARF_T$  ( $\pi_{T_i}[t].v$  — положение агента  $i$  в момент  $t$ ,  $\pi_{T_i}[t].d$  — его направление в этот момент)

---

```

1: for  $i \in 0, \dots, |\pi| - 1$  do
2:    $\pi_{T_i}[0].v \leftarrow \pi_i[0]$ 
3:    $\pi_{T_i}[0].d \leftarrow X_+$  ▷ Считаем, что агенты изначально ориентированы вдоль
   положительного направления оси  $X$ 
4: end for
5: for  $t \in 0, \dots, \max |\pi_i| - 1$  do
6:    $turns \leftarrow [\emptyset, \dots, \emptyset]$ 
7:   for  $i \in 0, \dots, |\pi| - 1$  do
8:      $next\_direction \leftarrow GET\_DIRECTION(\pi_i[t], \pi_i[t + 1])$ 
9:      $turns[i] \leftarrow ROTATE(\pi_{T_i}[t].d, next\_direction)$ 
10:  end for
11:   $m \leftarrow \max |turns[i]|$ 
12:  for  $i \in 0, \dots, |\pi| - 1$  do
13:    for  $j \in 0, \dots, m - 1$  do
14:       $\pi_{T_i} \leftarrow \pi_{T_i} \cup (\pi_i[t], turns[i][j])$ 
15:    end for
16:  end for
17: end for
```

---

Алгоритм PIBT получает на выходе набор путей, не учитывающих повороты агентов. Далее, для получения корректного решения этот набор путей необходимо преобразовать в набор путей для задачи  $MARF_T$ . Дан-

ную операцию выполняет Алгоритм 1. Он пошагово изменяет все пути следующим образом: берется каждый ход решения PIBT и для всех агентов, которые должны совершить поворот в этот ход, в ответ добавляется новое состояние с поворотом, а для всех остальных агентов добавляется последнее их состояние (агент остается на месте). Затем операция повторяется, если каким-либо агентам необходимо совершить разворот. После этого все агенты направлены в направлении из текущей клетки решения PIBT в следующую и в ответ добавляются состояния, переводящие нужных агентов вперед.

## 2.2. PIBT<sub>T</sub>

Данный алгоритм является простейшей доработкой PIBT до алгоритма для задачи MAPF<sub>T</sub>.

Агентам сопоставляются приоритеты, аналогичны приоритетам из алгоритма PIBT. Выбор следующей клетки происходит похожим образом: Сначала с помощью алгоритма A\* для агента определяется следующая клетка на его кратчайшем пути до цели. Если клетка находится не прямо перед агентом, то алгоритм поворачивает агента в направлении этой клетки. Иначе, алгоритм проверяет возможность продвижения агента вперед. Если клетка свободна, то PIBT<sub>T</sub> помечает клетку как зарезервированную для агента. Если же клетка уже была зарезервирована для другого агента, то алгоритм пропускает данную клетку, выбирает кратчайший путь, не проходящий через эту клетку, и поворачивает агента к этой клетке, либо, оставляет агента на месте, если кратчайший путь через другие клетки оказывается длиннее. Также, алгоритму необходимо учесть случай, в котором клетку в данный момент занимает агент с более низким приоритетом. В этом случае алгоритм использует техники Priority inheritance и Backtracking аналогично PIBT.

## 2.3. WinPIBT<sub>T</sub>

В ходе тестирования алгоритма PIBT<sub>T</sub> было выявлено много проблем. Ключевой проблемой стало то, что алгоритм в некоторых случаях алгоритм не способен обойти одним агентом другого. Из этого стало ясно, что алгоритму при планировании необходимо принимать в расчет не только предыдущий ход, но и несколько ходов, предшествующих предыдущему. WinPIBT<sub>T</sub> решает данную проблему, планируя пути агентов на несколько ходов вперед, как это делает алгоритм WinPIBT [Okumura et al., 2019]

### 2.3.1. Устройство алгоритма WinPIBT<sub>T</sub>

Алгоритм устроен аналогично алгоритму PIBT<sub>T</sub>. Агенты резервируют для себя положения вида — время и вершина в будущем. Если в текущий момент в вершине находится агент, то происходят Priority Inheritance и Backtracking.

### 2.3.2. Техника откатов в WinPIBT<sub>T</sub>

Из-за того, что алгоритм планирует пути на несколько ходов в будущее могут возникать особые конфликты (рис. 2)

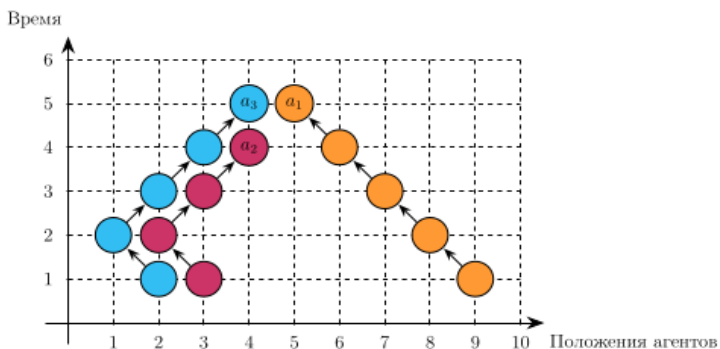


Рис. 2: Конфликт в алгоритме WinPIBT<sub>T</sub>

Так, на рис. 2 представлена гипотетическая ситуация, в которой агенты могут планировать ходы на 4 хода вперед, а граф – полоска . Агенты и успешно спланировали для себя пути, а агент не может продвинуться далее. Для обработки таких ситуаций в алгоритм добавлена система откатов: если алгоритм не может спланировать агенту ход, то он начинает постепенно удалять последние ходы этого агента до тех пор, пока не найдется клетка, в которой агент может простоять до конца планируемого периода. При нахождении такой клетки, агент остается в ней до конца планируемого периода. Если же такой клетки не нашлось, то агенту восстанавливаются все удаленные клетки из его пути, удаляются из остальных путей все клетки, которые зарезервированы в ходы, позже последнего хода рассматриваемого агента. Рассматриваемый агент получает самый высокий приоритет и алгоритм продолжается. Самый высокий приоритет позволяет данному агенту гарантированно пройти вперед на следующем ходу.

## 3. Эксперименты

Предложенные в работе алгоритмы реализованы на языке C++. Эксперименты проводились на нескольких картах из бенчмарков из известного набора MAPF-бенчмарков [Stern et al., 2019].

Каждый тест состоит из сценария– карты и набора положений агентов. Для сбора статистики по алгоритмам перебирается текущее число агентов ( ) и всем алгоритмам на вход подаются первые агентов из сценария. Алгоритмы запускаются на переданных данных, у них замеряется

время работы, собираются метрики полученных решений (процент успешных запусков – Success rate, сумма длин путей агентов – Sum of costs и максимальный пройденный путь – Makespan). Эксперименты проводятся на трех картах: Random (на квадратной карте препятствия расставлены случайным образом), Warehouse (складское помещение) и Den312d (карта из игры Dragon Age: Origins. На картах этой игры часто проводят замеры для анализа алгоритмов MAPF).

Запуск тестов происходит на процессоре Intel(R) Core(TM) i5-1038NG7 CPU @ 2.00GHz, 16 ГБ ОЗУ. Исходный код доступен в репозитории [https://github.com/LutsenkoAnton/pibt\\_t](https://github.com/LutsenkoAnton/pibt_t).

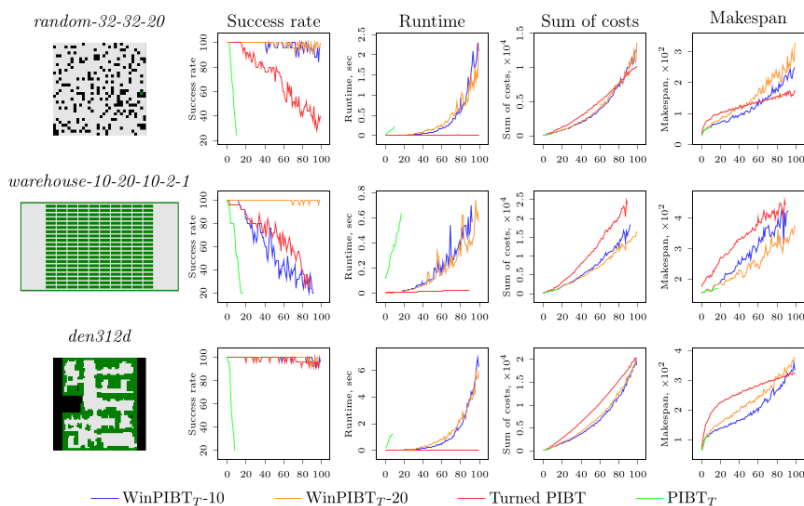


Рис. 3: Результаты экспериментов

На рис. 3 представлены результаты экспериментов. PIBT показывает неплохие Makespan и Sum-of-costs, но его success rate падает очень быстро.

Сравним теперь Turned PIBT и WinPIBT<sub>T</sub>. Графики времени работы алгоритмов выглядят одинаково на разных картах: с ростом числа агентов время работы WinPIBT<sub>T</sub> начинает резко возрастать. Действительно, это довольно предсказуемый результат: WinPIBT<sub>T</sub> пытается спланировать больше ходов, обработка конфликтов для него дороже, а с ростом числа агентов число конфликтов возрастает. Однако, для 100 агентов видим, что время выполнения занимает несколько секунд, что вполне приемлемо для применения на практике.

На всех предложенных графиках WinPIBT<sub>T-20</sub> получает success rate, близкий к 100%. Это является его несомненным преимуществом над Turned PIBT: как видим, Success rate у Turned PIBT на двух из трех графиков падает почти до 20% – границе, после которой запуск алгоритма в данных экспериментах заканчивался.

Как можно видеть, метрики решения на карте Random у WinPIBT<sub>T</sub> не очень хорошие: Sum of costs примерно такой же, как у Turned PIBT, а Makespan в итоге оказывается в несколько раз больше. Вероятно, такой рост в Makespan обусловлен стратегией откатов: агент, не нашедший дальнейшего пути может потерять 20 ходов, если не найдет выхода.

Графики для Den312d похожи на графики для Random. Для небольшого числа агентов WinPIBT<sub>T</sub> дает выигрыш, но в конце получается решение, схожее по характеристикам Turned PIBT.

Перейдем к рассмотрению ключевого случая: Warehouse. Алгоритм служит для планирования путей агентов на складах, поэтому этот тест особенно важен для данной работы. В этих тестах мы видим, что метрики у WinPIBT<sub>T</sub> получаются меньше, чем у Turned PIBT. Также легко видеть, что Success rate у WinPIBT<sub>T-20</sub> примерно равен 100%, а у Turned PIBT и WinPIBT<sub>T-10</sub> постепенно падает. Это связано с тем, что карта, на которой проходит тестирование имеет много узких коридоров. Агентам, чтобы в них разойтись, надо пройти большое расстояние (объехать полки с товарами). Очевидно, Turned PIBT с такой задачей не справляется. WinPIBT<sub>T-10</sub> тоже не хватает планирования для объезда таких препятствий, а WinPIBT<sub>T-20</sub> спокойно справляется с задачей.

## Заключение

В работе представлено три алгоритма, решающих задачу много-агентного планирования с поворотами: PIBT<sub>T</sub>, Turned PIBT и WinPIBT<sub>T</sub>. Проведено сравнение алгоритмов, в ходе которых показаны достоинства и недостатки этих алгоритмов. В ходе сравнения было подтверждено, что WinPIBT<sub>T</sub> работает лучше альтернатив на основной карте: карте складского помещения, при этом стабильно предоставляя решения для других карт, не сильно уступающие другим алгоритмам.

## Список литературы

- [Barer et al., 2014] Barer M., Sharon G.; Stern R., Felner A. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem // In: Proceedings of the Annual Symposium on Combinatorial Search (SoCS). – 2014. – Vol. 5(1). – P. 19-27. – doi: 10.1609/socs.v5i1.18315.
- [De Wilde et al., 2014] De Wilde B., Ter Mors A. W., Witteveen C. Push and rotate: a complete multi-agent pathfinding algorithm // In: Journal of Artificial Intelligence Research. – 2014. – Vol. 51. – P. 443-492. – doi: 10.1613/jair.4447.



- [Khorshid et al., 2011]** Khorshid M., Holte R., Sturtevant N. A polynomial-time algorithm for non-optimal multi-agent pathfinding // In: Proceedings of the International Symposium on Combinatorial Search. – 2011. – Vol. 2(1). – P. 76-83. – doi: 10.1609/socs.v2i1.18205.
- [Li et al., 2021]** Li J., Ruml W., Koenig S. Eecbs: A bounded-suboptimal search for multi-agent pathfinding // In: Proceedings of the AAAI conference on artificial intelligence. – 2021. – Vol. 35(14). – P. 12353-12362. – doi: 10.1609/aaai.v35i14.17466.
- [Sajid et al., 2012]** Sajid Q., Luna R., Bekris K. Multi-agent pathfinding with simultaneous execution of single-agent primitives // In: Proceedings of the International Symposium on Combinatorial Search. – 2021. – Vol. 3(1). – P. 88-96. – doi: 10.1609/socs.v3i1.18243.
- [Sharon et al., 2015]** Sharon G., Stern R., Felner A., Sturtevant N. R. Conflict-based search for optimal multi-agent pathfinding // In: Artificial intelligence. – 2015. – Vol. 219. – P. 40-66. – doi: 10.1016/j.artint.2014.11.006.
- [Stern et al., 2019]** Stern R., Sturtevant N.R., Felner A., Koenig S., Ma H., Walker T.T., Li J., Atzmon D., Cohen L., Kumar T.K.S., Boyarski E., Bartak R. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks // In: Symposium on Combinatorial Search (SoCS). – 2019. – Vol. 10(1). – P. 151-158. – doi: 10.1609/socs.v10i1.18510.
- [Okumura et al., 2019]** Okumura K., Tamura ., Défago X. winpibt: Extended prioritized algorithm for iterative multi-agent path finding // arXiv preprint arXiv:1905.10149. doi: 10.48550/arXiv.1905.10149. – 2019.
- [Okumura et al., 2022]** Okumura K., Machida M., Défago X., Tamura Y. Priority inheritance with backtracking for iterative multi-agent path finding // In: Artificial Intelligence. – 2022. – Vol. 310, 103752. – doi: 10.1016/j.artint.2022.103752.
- [Zhang et al., 2023]** Zhang Y., Harabor D., Le Bodic P., Stuckey P. J. Efficient Multi Agent Path Finding with Turn Actions // In: Proceedings of the International Symposium on Combinatorial Search. – 2023. – Vol. 16(1). – P. 119-127. – doi: 10.1609/socs.v16i1.27290.