Enhancing PIBT via multi-action operations

Egor Yukhnevich, Anton Andreychuk

Abstract

The League of Robot Runners is an Amazon Robotics-sponsored competition focused on multi-robot coordination challenges (including dynamics, planning, task assignment, and execution) that aims to advance research in Multi-Agent Path Finding and Task Scheduling with applications in warehouse logistics and manufacturing. This report describes the solution that won all competition nominations. The core feature is an enhanced PIBT approach (EPIBT) that efficiently generates collision-free actions for thousands of agents in milliseconds while introducing multi-action operations for better agent cooperation in congested areas. Unlike windowed solvers planning for a specific depth, our solution focuses on single multi-action operations. Complementing EPIBT, our solution incorporates graph guidance techniques, parallelized Large Neighborhood Search for optimization, and a simple yet highly efficient scheduler.

Introduction

Multi-agent Pathfinding (MAPF) is a well-known and extensively studied problem in which a group of agents, starting from their initial locations, must reach designated goal locations while avoiding collisions. Numerous variations of this problem exist (Stern et al. 2019). The classical MAPF problem is known to be NP-hard (Geft and Halperin 2022). Consequently, existing MAPF solvers that guarantee finding an optimal solution – such as CBS (Sharon et al. 2015) and its variants (Li et al. 2019; Boyarski et al. 2015), BCP (Lam et al. 2022), ICTS (Sharon et al. 2013), and others – face significant challenges with runtime and scalability as the number of agents increases. To address these issues, suboptimal variants (Barer et al. 2014; Huang, Dilkina, and Koenig 2021; Li, Ruml, and Koenig 2021) and anytime approaches, such as MAPF-LNS (Li et al. 2022; Huang et al. 2022) and LaCAM* (Okumura 2023; 2024), have been developed. Anytime solvers often employ extremely fast rule-based techniques, such as PIBT (Okumura et al. 2022) or Push-and-Rotate (De Wilde, Ter Mors, and Witteveen 2014), to quickly generate initial solutions, which are then iteratively improved within a given time budget.

However, the aforementioned solvers are designed for the classical MAPF scenario, where all goal locations are known in advance. Some algorithms leverage this assumption; for example, LaCAM uses depth-first search to accelerate the search process. In contrast, there are several modified MAPF problem statements in which agents are assigned new goal locations each time they reach their current goal. The lack of information about future goals at the outset makes most traditional MAPF solvers unsuitable for these scenarios, which are typically referred to as Lifelong MAPF (LMAPF). A further variation of the LMAPF problem is Multi-agent Pickup and Delivery (MAPD), where each agent is tasked with picking up an item from one location and delivering it to another. This formulation is inspired by real-world applications, such as autonomous warehouses where thousands of robots transport goods. A related challenge is task assignment or task allocation, which involves distributing tasks among agents to maximize overall system efficiency. Both MAPD and task allocation are closely related, and considering them jointly can significantly enhance system performance.

The League of Robot Runners (Chan et al.) is a MAPF-related competition that was held for the second time in late 2024 – early 2025. Unlike the previous edition, this year's competition featured two combined tasks: multi-agent pathfinding and task assignment. There were three main tracks, requiring participants to solve either one of these problems individually or to develop an approach that addresses both simultaneously. One of the core challenges and distinguishing features of this competition is the extremely limited time allotted to the solver – just one second to decide the next action for each agent, controlling up to 10,000 agents. This stringent time constraint makes most existing approaches impractical. In addition, the agents' action model includes rotations: each agent has an orientation and can only move forward; to move in a different direction, it must first perform a rotation. This action model further increases the complexity of the task and restricts the applicability of many existing solvers.

The winner of the last year's competition introduced an approach called WPPL (Jiang et al. 2024), which combines Windowed PIBT with a Graph Guidance technique and parallelized Large Neighborhood Search (LNS). Windowed PIBT is employed to rapidly generate initial collision-free actions for several upcoming steps. The Graph Guidance component helps to reduce congestion among agents and promotes a more even distribution across the map. Finally,

the LNS module optimizes the solution produced by PIBT, making efficient use of the available computational time.

The proposed solution for the pathfinding problem employs similar components but differs in several key aspects. Next we will describe in details the overall solution, and EPIBT in particular.

Solution

The primary distinction of the designed solver lies in the planner responsible for generating rapid initial solutions. A major limitation of Windowed PIBT is that it is not designed for action models involving rotations. By planning for multiple steps ahead rather than just one, Windowed PIBT partially addresses the original PIBT's drawback – namely, the assumption that agents can vacate an occupied cell in a single action, which is often not possible when rotations are required. While multi-step planning makes PIBT applicable to such action models, it is not as efficient as it could be. To overcome these limitations, the proposed modification, called Enhanced PIBT (EPIBT), introduces operations composed of sequences of actions, enabling the resolution of complex collisions that require multiple moves.

Enhanced PIBT

One of the most crucial aspects of EPIBT is the concept of operations. Each operation consists of a sequence of actions performed by an agent. We considered operations of lengths 3, 4, and 5. Figure 1 illustrates all cells and states that are reachable with operations of different lengths. With an operation length of 1, as in basic PIBT, only two cells (marked in red) are reachable. This limited reachability significantly restricts the ability to efficiently resolve collisions between agents.

Although the maximum number of possible operations is 4^{length}, the actual number is much smaller because many operations can be discarded. First, operations containing multiple redundant rotation actions can be eliminated. Additionally, it is unnecessary to consider operations that end with rotations, since rotations do not change the agent's position. Instead, multiple operations such as FFW, FFR, and FFC can be merged into a single FFW operation[1], with the h-value of the successor state calculated based on the best heading among the reachable ones. However, the number of operations still exceeds the number of possible reachable states. Due to the presence of other agents in the workspace, it is necessary to consider the time dimension and include wait actions in operations, allowing an agent to reach a state later if needed to avoid collisions. The actual number of operations that must be considered corresponds to the number of possible sequences of cells that the agent occupies while executing the operation. Table 1 shows how many different cells and states can be reached depending on the operation length, as well as the number of unique cell sequences. For clarity, cells are positions with (i, j) coordinates, states additionally include orientation (i.e., are defined by the tuple (i, j, o)), while cell sequences are sequences of positions

[1]Note that F stands for the move forward action, W for wait, R for rotate, and C for counterclockwise rotate

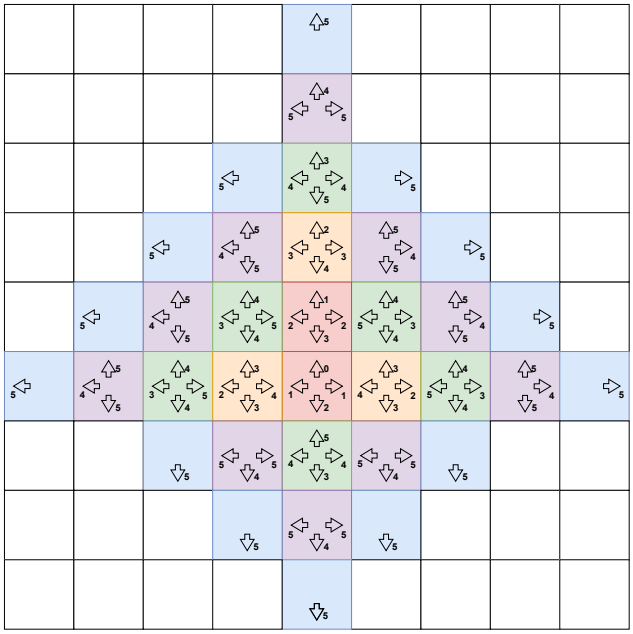

Figure 1: Cells and states reachable with different length of the operations. Different cell colors indicate the minimum required length of the operation to reach the corresponding cell. Arrows and numbers near them indicate the actual state and number of actions required to reach it.

Length	1	2	3	4	5
Cells	2	5	11	21	35
States	4	10	23	48	88
Cells sequences	2	6	17	48	136

Table 1: Number of possible reachable cells, states and unique sequences of occupied cells based on length of the operations.

with length corresponding to the column's length value – $\{(i_1, j_1), (i_2, j_2), (i_3, j_3)\}$ if the length is 3.

Another important aspect of operations is the order in which they are considered. In regular PIBT, actions are prioritized based on the distance to the goal. In EPIBT, this approach must be extended, as multiple states may have the same h-value. To address this, we implemented a tie-breaking mechanism that favors operations involving forward movement. Empirically, this mechanism produced the best results. We also limited the number of allowed collisions during the execution of a given operation. Since PIBT recursively invokes agents it interferes with to resolve collisions, excessive recursion can significantly slow down the process. Therefore, when evaluating possible operations, we skip those that would result in collisions with more than one agent. While the tie-breaking mechanism and collision limit significantly improved the speed of our approach, we believe there is still considerable room for improvement in how operations are selected.

Algorithm 1 presents the pseudocode for EPIBT. It largely

follows the logic of the original PIBT approach, with several key enhancements. By default, all agents are assigned a wait-in-place action (lines 3–5). The default operation has length 1, but it's still a sequence. The agents are then sorted based on their distance to their respective goals (lines 6–8). The main loop follows, where each agent that does not yet have a non-default set of actions executes the EPIBT procedure (lines 9–14).

During the EPIBT procedure, each agent attempts to select the most preferable set of actions. The order in which an agent's operations are considered is determined by the value $w_{op} = h(s_i, op, g_i) \cdot \alpha + \beta_{op}$, where α and β_{op} are weighting coefficients. In our implementation, α is set to a high value, making the second term, β_{op}, primarily a tie-breaker when two states have the same h-value. The β values for different operations are chosen so that rotation actions are preferred over wait actions, and movement actions are the most preferred. If an operation leads to an obstacle or outside the grid, it is skipped (lines 19–20).

The procedure $getPath(s_i, d_i)$ returns the sequence of states occupied by the agent, starting from state s_i and executing the sequence of actions d_i. If the operation is valid with respect to static obstacles, it is then checked for collisions with other agents. For this, the $getUsed(s_i, op, P)$ procedure is used, which returns the set of agent IDs that would collide with agent i if it attempted to perform the sequence of actions in operation op starting from state s_i. If the operation is collision-free, it is adopted as the agent's desired sequence of actions (lines 20–23). If it results in collisions with two or more agents, the operation is skipped (lines 24–25). If there is a collision with a single agent j, we attempt to rebuild agent j's sequence of actions, taking into account that agent i wants to perform operation op. If this attempt fails, the set of paths P is reverted to its previous state, and the loop continues with the next most preferable operation. If all possible sequences fail and none can be executed collision-free, the agent is assigned the default wait-in-place action $d'i$ (lines 13–14).

One more crucial difference from the original PIBT approach is that we allow agents to collide with high-priority agents and force them to rebuild their operations. According to our empirical results, EPIBT performs better this way. However, this change potentially violates the theoretical guarantee of reachability to all goal locations that the original approach provides.

Algorithm 1 EPIBT

```
1:  Input: graph G, starts {s_1, ..., s_n}, goals {g_1, ..., g_n}
2:  Output: selected actions {d_1, ..., d_n}
3:  Preface: d_i ← {wait} for i = 1, ..., n
4:  Preface: d'_i ← {wait} for i = 1, ..., n
5:  Preface: P ← getPath(s_i, d_i) for i = 1, ..., n
6:  p_i ← dist(s_i, g_i); for each agent i = 1, ..., n
7:  A ← {1, ..., n}
8:  sort A in ascending order of priorities p_i
9:  for i ∈ A do
10:     if d_i ≠ {wait} then
11:         continue
12:     P ← P \ getPath(s_i, d_i)
13:     if EPIBT(i) = failed then
14:         P ← P ∪ getPath(s_i, d'_i)
15: procedure EPIBT(i)
16:     C ← op ∈ Operations
17:     sort C in descending order of w_op
18:     for op ∈ C do
19:         if getPath(s_i, op) ⊄ G then
20:             continue
21:         if getUsed(s_i, op, P) = ∅ then
22:             d_i ← op
23:             P ← P ∪ getPath(s_i, op)
24:             return success
25:         if |getUsed(s_i, op, P)| ≥ 2 then
26:             continue
27:         j ∈ getUsed(s_i, op, P)
28:         P ← P \ getPath(s_j, d_j)
29:         P ← P ∪ getPath(s_i, op)
30:         d_i ← op
31:         if EPIBT(j) = success then
32:             return success
33:         P ← P \ getPath(s_i, d_i)
34:         P ← P ∪ getPath(s_j, d_j)
35:         d_i ← d'_i
36:     return failed
```

Large Neighborhood Search

The solution produced by EPIBT is valid, collision-free, and generated within milliseconds. However, it is constructed in a prioritized manner based on a specific ordering of the agents. To enhance solution quality and make efficient use of the remaining computational time, we incorporated a Large Neighborhood Search (LNS) algorithm based on simulated annealing (SA). The process begins by running EPIBT to obtain an initial solution, after which SA is applied to further refine it.

At each SA step, a random agent r is selected, its path is removed, and a path reconstruction algorithm – similar to the standard recursive EPIBT method – is invoked. The key difference is that when choosing an operation, there is a certain probability (0.3 in our implementation) of skipping it. This introduces randomness into the recursion, ensuring that repeated runs can yield different solutions. If the operations are successfully rebuilt using SA, we can either accept the new solution or revert to the original one. As the SA metric, we use the sum of $w_{op} \times p_r$ for each agent, where p_r is the priority of agent r (ranging from 0 to 1, indicating how close the agent is to its goal). This metric encourages agents to complete their tasks as quickly as possible and proceed to new ones – the higher the metric, the better the agents are progressing toward their targets. According to the logic of SA, we always accept changes that improve the metric, but also have a small chance to accept changes that actually make the solution worse. This is done to potentially escape local maxima and find a global maximum. For the acceptance probability of changes with negative improvement, we

utilized the following equation:

$$P(\text{accept}) = \exp\left(-\frac{\text{Score}_{\text{old}} - \text{Score}_{\text{new}}}{T \cdot \text{Score}_{\text{old}}}\right)$$

In our implementation, the initial temperature value was $T = 0.001$, and it was further reduced by a factor of 0.999 at every step. We utilized a fairly low initial temperature, as the initial solution found by EPIBT has good quality, and we wanted to avoid its degradation.

To leverage the multicore CPUs available in the competition, we ran multiple instances of EPIBT+LNS in parallel and selected the best result among them. While this parallelization approach is straightforward, it is not as efficient as it could be with process synchronization. Our attempts to develop a more advanced parallel EPIBT+LNS, where threads actively communicate and share improvements, did not yield better results. Nevertheless, we believe that further improvements are possible in this area.

Graph Guidance

Last but not least, a key component of our path planning solver is Graph Guidance. This is a relatively new and actively developing technique in LMAPF, used to reduce congestion among agents and distribute them more evenly across the grid. One of the first papers to employ this technique was Follower (Skrynnik et al. 2024), which introduced a hybrid approach combining search-based and learning-based components in a single LMAPF solver. However, in that work, graph guidance was just one part of the overall solver and was not the main focus. A more in-depth exploration of graph guidance and its construction methods was presented in (Zhang et al. 2024). The competition baseline used the PIBT+traffic flow approach (Chen et al. 2024), which also incorporates a form of graph guidance optimization. All these studies have demonstrated the benefits of modifying action costs to improve throughput and mitigate the short-sighted behavior typical of solvers like PIBT.

In our solution, we did not use any existing graph guidance method nor did we develop a new one from scratch. Instead, for the small random-32-32 map, we created a handcrafted grid with predefined costs. For other maps, we applied a simple rule-based approach, creating "highways" in various directions.

Scheduler

In addition to the pathfinding component, it was necessary to develop a module for distributing tasks among agents, i.e., a scheduler. Due to the very limited time available – which must be shared between both pathfinding and scheduling – all our attempts to implement a more complex solver, such as a parallelized version of the classical Hungarian method (Kuhn 1955), were unsuccessful.

Instead, we implemented a simple yet effective approach that is both fast and yields good results. For each agent r, we maintain $Pool(r) = \{\langle distance(r, task), task\rangle \mid task \in Tasks\}$ – an array of (distance, task) pairs, sorted by the distance from agent r to each task. Each agent r prefers to take the first task from its array $Pool(r)$. However, the same task may be the top choice for multiple agents, making it impossible for all of them to select it. To resolve such conflicts and quickly find the most preferable agent-task pairs, we insert the first element from each agent's pool into a heap and assign tasks by prioritizing agents with the shortest distance to their chosen task. If an agent cannot take its preferred task because it has already been assigned to another agent, the next most preferred task from its pool is added to the heap. This process continues until all agents have been assigned tasks.

This greedy procedure for task allocation proved to be both efficient and effective, allowing us to devote more computational time to the pathfinding component and its optimization via LNS. We believe that the lack of improvement from more complex procedures is due to the imperfect estimation of actual costs.

Disabling agents

Our solver, designed specifically for the competition, incorporated a technique tailored to improve performance on certain maps and under specific competition conditions. This technique, inspired by WPPL, involves selectively disabling agents. On the Game map, for example, the presence of too many agents led to excessive interference and crowding in narrow aisles. To reduce congestion and increase overall throughput, we implemented the following strategy: at each step, we selected the n agents closest to their targets to remain active, while the remaining agents were temporarily disabled – that is, they stayed in place to avoid obstructing others. After experimenting with several variants, we found that the best results were achieved with $n = 2500$.

Experimental Evaluation

Additionally to the results, obtained on the instances from the competition, we have run additional series of experiments on three maps from the competition (random-32-32-20, warehouse and brc202d) with varying number of agents to evaluate the performance of the designed approach and compare it with existing state-of-the-art approaches.

For this purpose, we have evaluated the following approaches:

- **PIBT±GG.** Our implementation of the basic algorithm.
- **PIBT+traffic flow.** Implementation of the basic planner from the LoRR competition.
- **EPIBT±GG.** Baseline method to estimate how EPIBT performs without LNS.
- **EPIBT+LNS±GG.** The final algorithm.
- **WPPL+GG.** The winner of LoRR 2023. Implementation was taken from code archive of the competition as the authors' public implementation is occasionally made for non-rotation model.

It is worth noting that in this experiment, we used our scheduler for all methods. The results are presented in Figure 2. The top row shows the map visualizations, the middle row displays the average throughput achieved by each method as a function of the number of agents, and the bottom row presents the average decision time, i.e., the time

Table 2: Here are our competition results (EPIBT+LNS with some specific optimizations and tricks) and the results after the competition of algorithms: PIBT, EPIBT and EPIBT+LNS. GG is enabled everywhere. During the competition, we have the best solution in 6 out of 10 tests.

Instance	Agents	Steps	Competition Results		PIBT		EPIBT		EPIBT+LNS	
			Tasks	Score	Tasks	Score	Tasks	Score	Tasks	Score
City-01	1500	3000	8420	0.997	7567	0.896	8378	0.992	8369	0.991
City-02	3000	3000	16787	0.987	7912	0.465	15899	0.935	16609	0.976
Game	6500	5000	23274	1	12342	0.530	21690	0.932	23303	1.001
Random-01	100	600	641	0.92	549	0.789	637	0.915	654	0.940
Random-02	200	600	1231	0.977	745	0.591	1066	0.846	1194	0.948
Random-03	400	800	2368	1	746	0.315	1619	0.684	2224	0.940
Random-04	700	1000	2585	1	551	0.213	1630	0.630	2281	0.882
Random-05	800	2000	3050	1	155	0.050	1651	0.541	2669	0.875
Sortation	10000	5000	152714	1	70798	0.464	144616	0.946	148828	0.975
Warehouse	10000	5000	154834	1	51881	0.335	141110	0.911	149526	0.966
Total			**365904**	**9.881**	**153246**	**4.648**	**338296**	**8.332**	**355657**	**9.494**

required to determine the next action. For the PIBT+traffic flow algorithm, EPIBT+LNS, and WPPL, the average time to compute one step is 1 second, as these algorithms continue to improve the solution as long as time permits. The graphs indicate that our solver, EPIBT+LNS+GG, outperforms all baselines, including last year's winner, WPPL. Comparing the results of the original PIBT approach and EPIBT, it is evident that EPIBT requires more time to make a decision, especially on random-32-32-20 and brc202d maps. This difference is explained by high congestion on these maps, which cannot be resolved by Graph Guidance. As a result, the approach runs many recursion calls to resolve collisions between agents. On the warehouse map, Graph Guidance is able to reduce congestion more efficiently. As a result, the decision time of EPIBT+GG is much closer to PIBT and also significantly lower than EPIBT, especially on instances with a high number of agents (8,000 and more). At the same time, Graph Guidance has no impact on PIBT in terms of decision time. This can be explained by the fact that PIBT makes single-action steps and has much fewer recursion calls and overall collisions than EPIBT. As a result, better distribution of agents on the map has an impact in terms of throughput, but not in terms of runtime.

Figures 3 and 4 show the heatmaps of waiting agents. The redder the cell, the more time any agent occupied this cell and performed a wait action in it. The absence of Graph Guidance results in high congestion in the middle of the map in the case of the warehouse and in some areas near narrow corridors in the case of the brc202d map. Agents cannot find collision-free actions and have to wait in place in such cases, which heavily impacts the overall performance of the system and reduces throughput. It is also worth noting that such behavior appears in most existing LMAPF and MAPD solvers that utilize windowed or reactive planning, especially when episodes are run not for 100-200 timesteps, but for thousands, as in the competition setting.

The last series of experiments were conducted on instances from the competition, after their public release. Here we evaluated the basic PIBT approach, EPIBT, and EPIBT+LNS. +GG is omitted as all approaches utilized it. The results of this experiment are shown in Table 2. For the competition results column, we took a compilation of the best achieved results from multiple attempts. Thus, the depicted results are slightly higher than the best attempt on the competition leaderboard. We disabled all tweaks and optimizations made for specific instances, such as various tie-breaking for operations, random length of operations, etc. Here EPIBT utilized operations of only length 4 as they showed the best results when considering operations of only one length. Still, the presented approach with "general" parameters for all instances is able to achieve a score of 9.494, which is higher than any other score achieved by other participants in the competition.

Conclusion

In this work, we have described the workings of our solver, which enabled us to win the League of Robot Runners 2024 competition. The core feature of our solution is an enhanced version of the PIBT algorithm that considers sequences of operations rather than single actions, allowing for efficient resolution of collisions between agents with action models that require rotations. Our solver is further complemented by a graph guidance technique and large neighborhood search optimization to enhance the solution even more. The external experimental evaluation demonstrates that our solver outperforms all existing state-of-the-art LMAPF/MAPD solvers. We believe there is still significant room for improvement in our approach and in MAPD solvers in general.

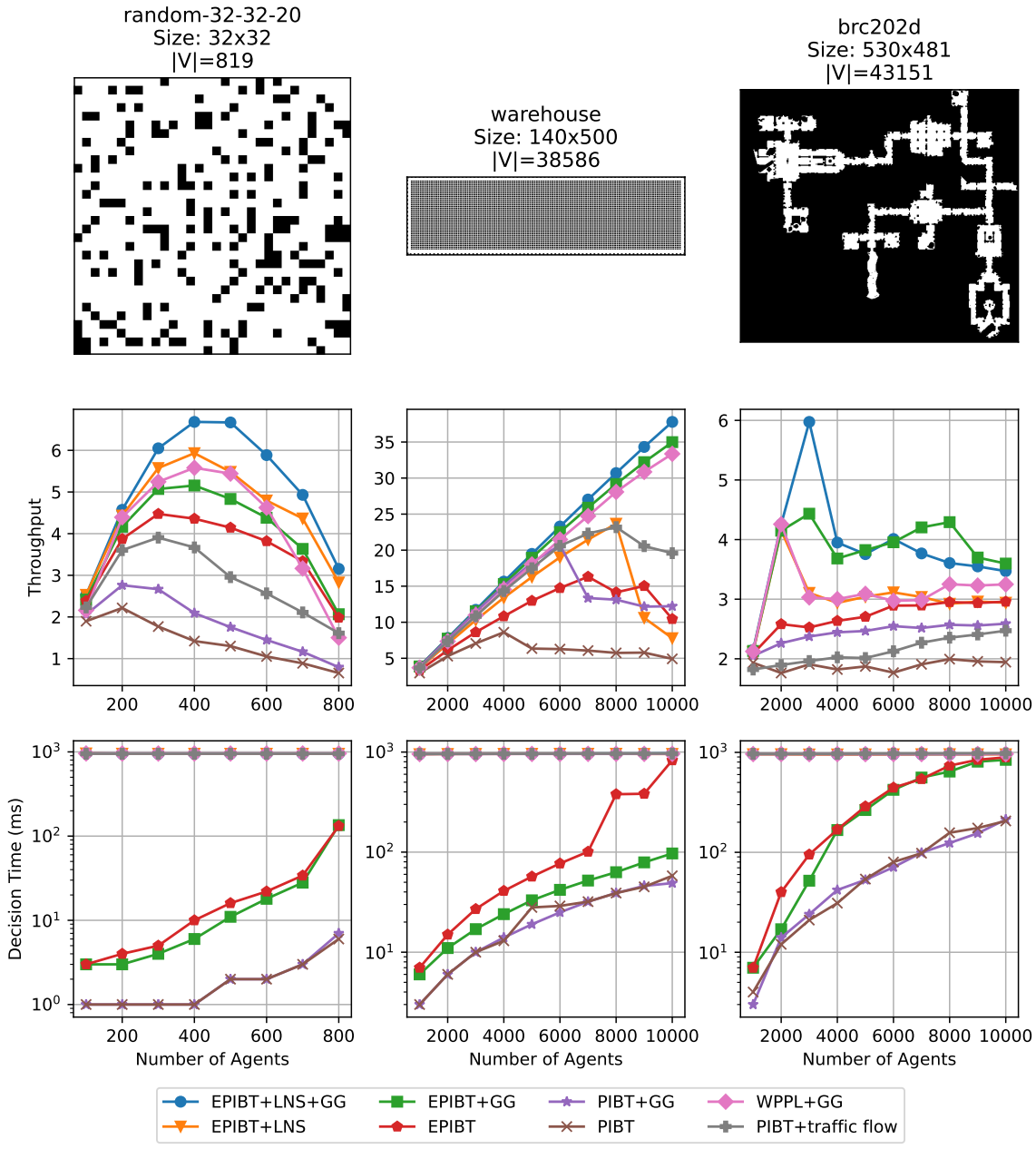

Figure 2: The X-axis graphs show the number of agents. Vertically, from top to bottom, there is information about the map, the map itself, then the bandwidth graph for each algorithm, then the average time to calculate one step.

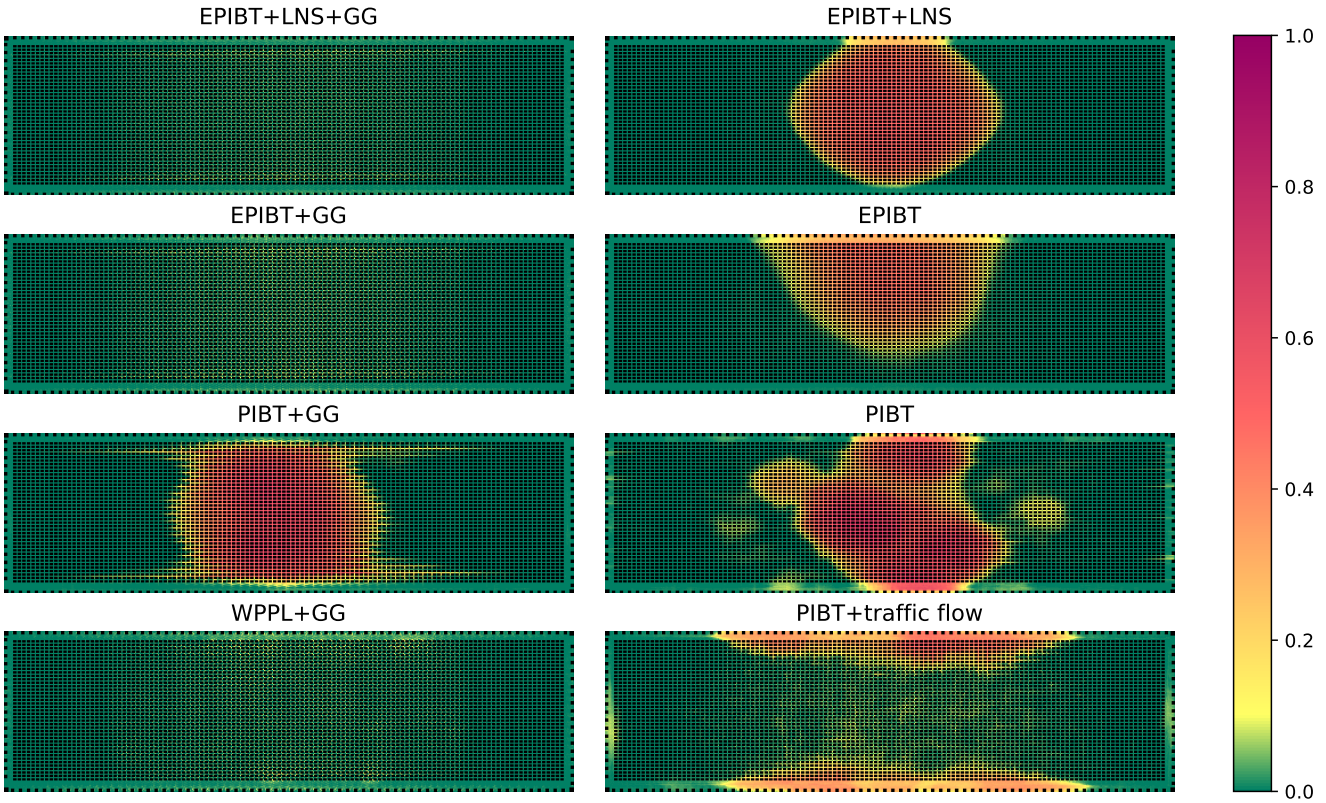

Figure 3: Comparison of the algorithms in the Warehouse with 10k agents. The heatmap shows the wait action usage (the number of steps agents wait in each vertex). The red color indicates areas with high traffic.

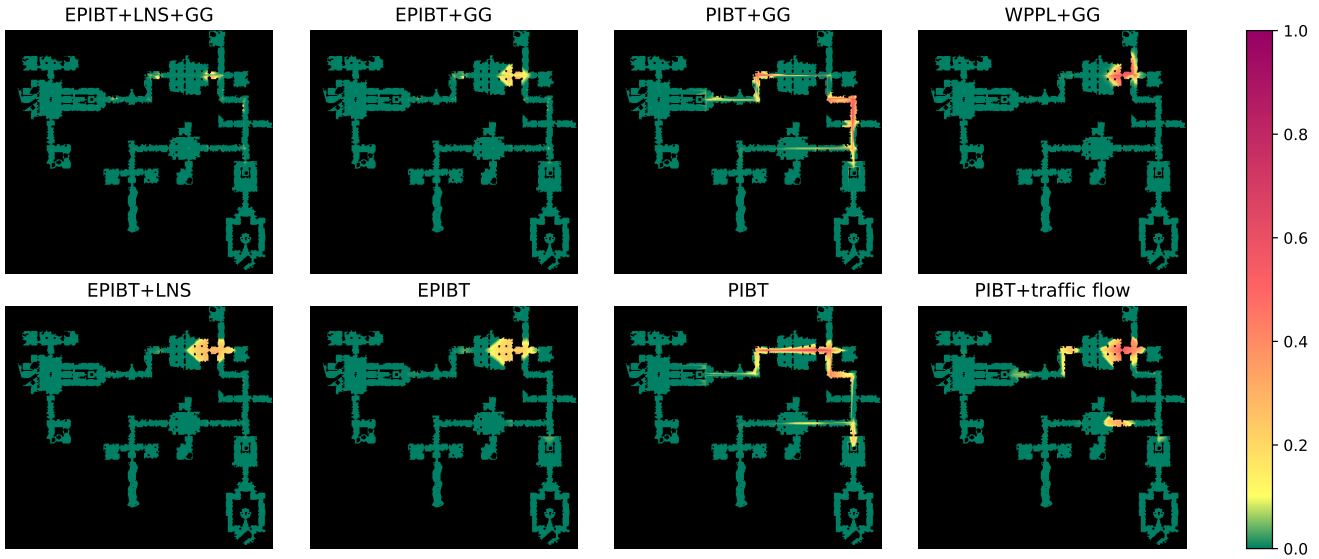

Figure 4: Comparison of the algorithms in the Game (brc202d) with 3k agents. The heatmap shows the wait action usage (the number of steps agents wait in each vertex). The red color indicates areas with high traffic.

References

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the international symposium on combinatorial Search*, volume 5, 19–27.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Betzalel, O.; Tolpin, D.; and Shimony, E. 2015. Icbs: The improved conflict-based search algorithm for multi-agent pathfinding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, 223–225.

Chan, S.-H.; Chen, Z.; Guo, T.; Zhang, H.; Zhang, Y.; Harabor, D.; Koenig, S.; Wu, C.; and Yu, J. The league of robot runners competition: Goals, designs, and implementation. In *ICAPS 2024 System's Demonstration track*.

Chen, Z.; Harabor, D.; Li, J.; and Stuckey, P. J. 2024. Traffic flow optimisation for lifelong multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20674–20682.

De Wilde, B.; Ter Mors, A. W.; and Witteveen, C. 2014. Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research* 51:443–492.

Geft, T., and Halperin, D. 2022. Refined hardness of distance-optimal multi-agent path finding. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 481–488.

Huang, T.; Li, J.; Koenig, S.; and Dilkina, B. 2022. Anytime multi-agent path finding via machine learning-guided large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 9368–9376.

Huang, T.; Dilkina, B.; and Koenig, S. 2021. Learning node-selection strategies in bounded suboptimal conflict-based search for multi-agent path finding. In *International joint conference on autonomous agents and multiagent systems (AAMAS)*.

Jiang, H.; Zhang, Y.; Veerapaneni, R.; and Li, J. 2024. Scaling lifelong multi-agent path finding to more realistic settings: Research challenges and opportunities. In *Proceedings of the International Symposium on Combinatorial Search*, volume 17, 234–242.

Kuhn, H. W. 1955. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2(1-2):83–97.

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2022. Branch-and-cut-and-price for multi-agent path finding. *Computers & Operations Research* 144:105809.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019. Improved heuristics for multi-agent path finding with conflict-based search. In *IJCAI*, volume 2019, 442–449.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. Mapf-lns2: Fast repairing for multi-agent path finding via large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 10256–10265.

Li, J.; Ruml, W.; and Koenig, S. 2021. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 12353–12362.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence* 310:103752.

Okumura, K. 2023. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 11655–11662.

Okumura, K. 2024. Engineering lacam*: Towards real-time, large-scale, and near-optimal multi-agent pathfinding. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, 1501–1509.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial intelligence* 195:470–495.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial intelligence* 219:40–66.

Skrynnik, A.; Andreychuk, A.; Nesterova, M.; Yakovlev, K.; and Panov, A. 2024. Learn to follow: Decentralized lifelong multi-agent pathfinding via planning and learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 17541–17549.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 151–158.

Zhang, Y.; Jiang, H.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2024. Guidance graph optimization for lifelong multi-agent path finding. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 311–320.