The Fortieth AAAI Conference on Artificial Intelligence (AAAI-26)

# Enhancing PIBT via Multi-Action Operations

**Egor Yukhnevich**[1], **Anton Andreychuk**[2]
[1]HSE University, Moscow, Russia
[2]Cognitive AI Systems Lab, Moscow, Russia
andreychuk@cogailab.com

## Abstract

PIBT is a rule-based Multi-Agent Path Finding (MAPF) solver, widely used as a low-level planner or action sampler in many state-of-the-art approaches. Its primary advantage lies in its exceptional speed, enabling action selection for thousands of agents within milliseconds by considering only the immediate next timestep. However, this short-horizon design leads to poor performance in scenarios where agents have orientation and must perform time-consuming rotation actions. In this work, we present an enhanced version of PIBT that addresses this limitation by incorporating multi-action operations. We detail the modifications introduced to improve PIBT's performance while preserving its hallmark efficiency. Furthermore, we demonstrate how our method, when combined with graph-guidance technique and large neighborhood search optimization, achieves state-of-the-art performance in the online LMAPF-T setting.

**Project page** — https://sites.google.com/view/epibt

## 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 search for an optimal solution to 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/density of agents increases. To address scalability problems, 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.

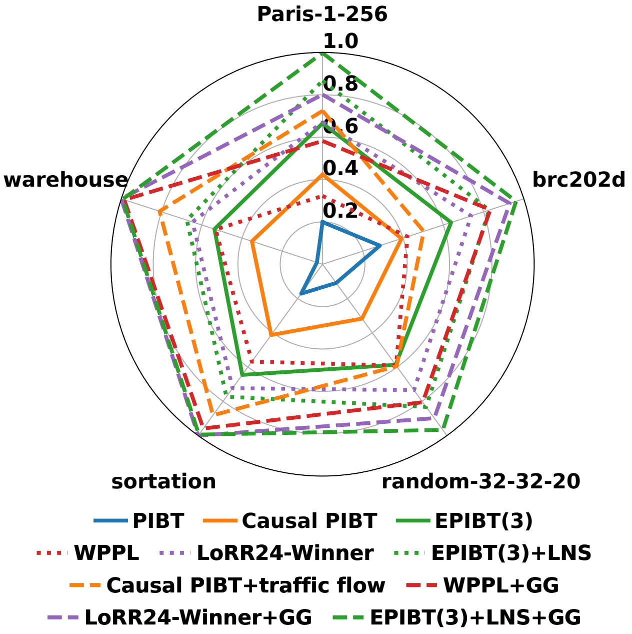

Figure 1: Spider plot demonstrating the relative performance of the evaluated approaches. Solid lines represent PIBT-like approaches without additional components, dotted lines indicate methods utilizing LNS, and dashed lines denote approaches that also incorporate GG. Related solvers are indicated by the same color.

However, the aforementioned solvers are tailored for the classical MAPF scenario, in which all goal locations are known in advance. Some algorithms explicitly leverage this assumption; for instance, LaCAM employs depth-first search to accelerate the search process. In contrast, there exists a variant of the MAPF problem where agents are assigned new goal locations each time they reach their current goal. This variant is commonly referred to as Lifelong MAPF (LMAPF). The absence of information about future goals at the outset makes most traditional MAPF solvers unsuitable for these scenarios, necessitating specific adaptations or entirely new approaches. One of the most com-

---

mon and effective strategies for addressing the challenge of continuously updating goals is the use of windowed search-planning only for a limited number of upcoming timesteps rather than the entire planning horizon. Consequently, when agents reach their current goals and receive new ones, the algorithm can be relaunched to generate the next segment of the plan, explicitly incorporating the new goals. The concept of windowed search for MAPF/LMAPF problems was first introduced in (Silver 2005) and has since been successfully adopted in approaches such as RHCR (Li et al. 2021b), WPPL (Jiang et al. 2024), WinC-MAPF (Veerapaneni et al. 2025) among others.

Another advantage of windowed-based solvers is their ability to quickly return the next actions, making them suitable for the online LMAPF problem. In this setting, the algorithm operates concurrently with real agents as they execute the planned actions. Consequently, the algorithm must provide the next set of actions within a strict time frame, often as short as one second. This requirement is imposed by real-world applications, such as autonomous warehouses (Dhaliwal 2020), logistics (Ferreira and Reis 2023), and others.

Another practical constraint, frequently encountered in real-world scenarios but often overlooked in MAPF/LMAPF literature, is the presence of agent orientations and the associated time required to change them. While most approaches can be adapted to action models that include rotations with relatively straightforward modifications, some methods require significant changes to maintain their performance and efficiency when applied to such models.

In this work, we present a modification of the rule-based iterative solver, Priority Inheritance with Back Tracking (PIBT) (Okumura et al. 2022), which is known for its excellent scalability. However, due to its design, PIBT struggles when applied to agents with a rotation action model, as agents may require more than one timestep to vacate an occupied cell if they must rotate first. The core idea of our proposed modification is the introduction of multi-action operations that combine all possible actions – rotations, moves, and waits. We also demonstrate that this modification can be beneficial not only for the rotation action model but also for the classic omnidirectional action model. Furthermore, our experimental evaluation shows that combining the modified PIBT with additional techniques, such as graph guidance and large neighborhood search, enables it to outperform existing state-of-the-art approaches on the considered problem statement: online LMAPF-T, i.e., Lifelong MAPF with a limited time budget and a rotation action model.

## Problem Statement

The environment is represented as a 4-connected grid of cells, where each cell is identified by its coordinates $(i, j)$. Each cell can be either traversable or blocked (representing obstacles). The set of all traversable cells is denoted by $V \subseteq \mathbb{Z}^2$, and the set of edges $E$ consists of pairs of adjacent traversable cells, i.e., $E = \{((i, j), (i', j')) \mid |i - i'| + |j - j'| = 1, (i, j) \in V, (i', j') \in V\}$.

We are given $n$ agents, each with a unique start location and orientation. The set of start locations is $\{(i_1^0, j_1^0), \dots, (i_n^0, j_n^0)\}$, where $(i_k^0, j_k^0) \in V$ for $k = 1, \dots, n$. Each agent also has an initial orientation $o_k^0 \in \{\text{north}, \text{east}, \text{south}, \text{west}\}$. The state of each agent at time $t$ is thus defined by both its location and orientation, i.e., $s_k^t = ((i_k^t, j_k^t), o_k^t)$.

An external task assignment module provides each agent with a current goal location $g_k = (i_{g_k}, j_{g_k}) \in V$. Whenever an agent reaches its current goal, it is immediately assigned a new goal location by the task assignment module. We assume that only the current goal location for each agent is known at any given time; future goals are revealed dynamically as agents reach their current goals.

We consider the *rotation action model*, in which each agent can perform one of four actions at each discrete timestep: $F$ – move forward, $R$ – rotate 90° clockwise, $C$ – rotate 90° counterclockwise, $W$ – wait in place. All actions have unit duration.

A collision occurs if, at any timestep, either (i) two agents occupy the same cell ($(i_k^t, j_k^t) = (i_l^t, j_l^t)$ for $k \neq l$), or (ii) two agents traverse the same edge in opposite directions simultaneously ($((i_k^{t-1}, j_k^{t-1}), (i_k^t, j_k^t)) = ((i_l^t, j_l^t), (i_l^{t-1}, j_l^{t-1}))$ for $k \neq l$).

The objective is to maximize the total number of goals reached by all agents within a fixed timestep horizon $T$, while avoiding collisions. This objective, commonly referred to as *throughput*, is defined as the average number of goals reached per timestep.

Additionally, we impose a strict time budget for the solver to compute the next actions for all agents at each timestep. This constraint is motivated by real-world applications, where the planner must react promptly to changes and operate in an online regime. In our experiments, this time limit is set to 1 second per timestep. If the solver exceeds this limit, all agents are delayed by one timestep for each additional second spent. The time budget is not cumulative: even if the solver uses less than the allotted time in one step, it cannot carry over the unused time to subsequent steps.

The problem statement considered in this work is identical to that of the League of Robot Runners (LoRR) (Chan et al. 2024) – a competition sponsored by Amazon Robotics, designed to bridge the gap between fundamental research and industrial applications, including warehouse logistics, transportation, and advanced manufacturing. Notably, the most recent edition of the competition introduced new challenges, such as dynamic task assignment and sequences of tasks. In this work, we focus exclusively on efficient agent routing, assuming that task assignment is handled by an external module beyond our control.

## Priority Inheritance with Backtracking

Before detailing the modifications introduced in the PIBT approach, it is useful to briefly describe its core concept. PIBT is a windowed solver with a single-step window, meaning that it provides collision-free actions for all agents for only one timestep. To achieve this, PIBT employs a prioritized approach: each agent is assigned a priority based on its distance to its goal, and agents plan their actions sequentially, avoiding collisions with higher-priority agents and selecting the most beneficial action – typically, the one that reduces their distance to the goal.

A key feature of PIBT is the use of priority inheritance with backtracking. If an agent $k$ encounters a collision with a higher-priority agent $l$, it inherits the priority $p_l$ and attempts to vacate the occupied cell, potentially pushing any other agents with lower priority than $p_l$. If such a push is unsuccessful, the agent receives a signal via backtracking and attempts to select an alternative action.

It was shown in (Okumura et al. 2022) that all agents will eventually reach their goal locations. However, simultaneous occupation of goal locations by all agents is not guaranteed. Additionally, a condition on the graph structure must be satisfied: for every pair of adjacent nodes, there must exist a simple cycle $C$ containing both nodes, with $|C| \geq 3$. This ensures that the agent with the highest priority can always push other agents from the desired location, guaranteeing that it will reach its goal within a bounded number of steps. After reaching its goal, its priority is decreased, enabling the next agent to reach its goal in a similarly bounded number of steps.

The original PIBT approach assumes that, at each timestep, at least one agent with the highest priority can move closer to its goal, as it can always vacate the desired location. However, under the rotation action model, this assumption does not hold: the agent occupying the desired location may first need to rotate before it can vacate the cell. This issue can be partially addressed by the Causal PIBT approach (Okumura, Tamura, and Défago 2021), which considers event-based causal dependencies between agents' actions. Causal PIBT was initially developed for a stochastic setting, where each action may be delayed with some probability; in the present context, rotation actions can be interpreted as deterministic delays.

Another way to address delays caused by rotation actions is to consider multiple timesteps, rather than just one. The idea of planning over multiple steps was explored in (Okumura, Tamura, and Défago 2019), where the winPIBT approach was introduced. Instead of planning only the next action, agents plan a path for the next $w$ steps. This algorithm is designed for classical MAPF and aims to resolve blockages caused by agents that have already reached their goal locations.

In contrast to winPIBT, which considers relatively long windows (up to 30 steps in the original paper), the approach proposed here uses a much shorter window (up to 5 steps). Rather than launching a path-planning algorithm, our method considers multi-action operations of limited length.

## Enhanced PIBT

In this section, we describe the modifications made to the original PIBT approach to efficiently support the rotation action model. In addition to introducing multi-action operations, we allow agents to be revisited during a single step; that is, an agent may change its chosen action even if a valid collision-free action has already been found. We also employ inheritance of operations, enabling the reuse of actions selected by agents in the previous step. The approach that integrates all these modifications is referred to as Enhanced PIBT, or simply EPIBT.

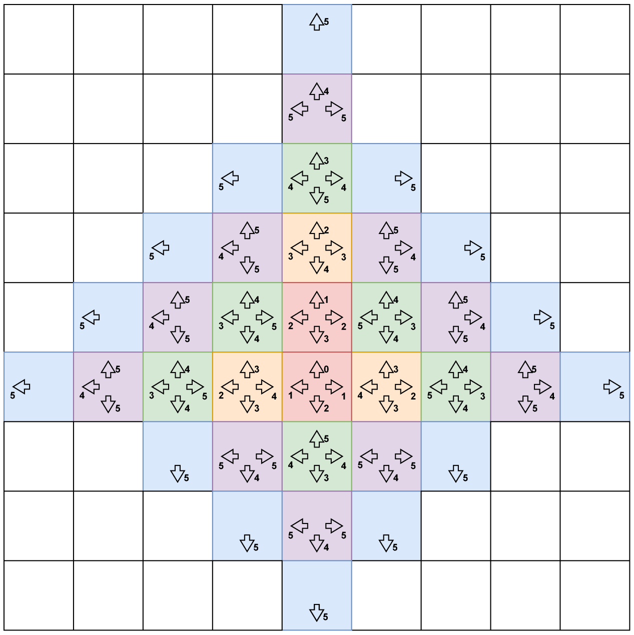

Figure 2: Cells and states reachable with different operation length. Different cell colors indicate the minimum required operation length to reach the corresponding cell. Arrows and numbers near them indicate the actual state and number of actions required to reach it.

| 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, reachable states and unique sequences of occupied cells depending on operation length.

## Multi-action Operations

A central aspect of EPIBT is the concept of operations. Each operation consists of a sequence of actions performed by an agent. Figure 2 illustrates all cells and states that are reachable with operations of different lengths. With an operation length of 1, as in regular 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 action combinations is $4^{\text{length}}$, the actual number of operations that must be considered is much smaller, as 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 location. Instead, multiple operations such as $FFW$[1], $FFR$, and $FFC$ can be

---

[1] The notation $FFW$ means that the agent consequently executes actions move forward, move forward and wait.

merged into a single $FFW$ operation, with the $h$-value (distance to the goal) 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 at a later time 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 refer to locations with $(i, j)$ coordinates, states additionally include orientation (i.e., are defined by the tuple $((i, j), o)$), while cell sequences are sequences of locations with length corresponding to the operation length – e.g., $\{(i_1, j_1), (i_2, j_2), (i_3, j_3)\}$ for length 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 forward movement over rotation, and rotation over wait. Empirically, this mechanism produced the best results.

## Revisiting Agents

Multi-action operations provide agents with a significantly greater set of options. While this is beneficial for performance, it also introduces new challenges. The first challenge is that considering multiple timesteps can lead to multi-agent collisions, i.e., situations where a chosen operation results in a collision with more than one agent. In such cases, recursively invoking the operation selection for each colliding agent leads to ambiguity, as these agents may mutually affect each other and all inherit the same priority due to priority inheritance. To address this, the proposed approach does not consider operations that would result in collisions with more than one agent.

The second challenge arising from the increased variety of options is the diversity of possible collisions between agents. In the original PIBT, each agent may be visited only once per timestep, and once its action is chosen, it cannot be changed. This restriction significantly limits the options available to agents. Moreover, if some agents fail to find collision-free operations, they will remain stationary for at least the next timestep. To overcome this limitation, we allow agents to be revisited, i.e., to reselect their operations in order to better resolve collisions and to find collision-free actions for at least some agents that have already been visited but failed to find valid operations. However, in high-density scenarios, unlimited revisiting can lead to a substantial increase in the algorithm's runtime. To mitigate this, we introduce a limit $L$ on the number of times each agent can be revisited during a single timestep.

## Inheritance of Operations

Another important feature implemented in EPIBT is the inheritance of operations. Like regular PIBT, EPIBT is executed at every timestep. When constructing operations from scratch, we assume that all agents will occupy their current locations for the next few steps. This assumption is necessary to detect potential collisions at any considered timestep and to allow EPIBT to recursively attempt to resolve them. However, in the previous step, we may have already constructed collision-free operations for multiple timesteps, of which only the first actions have been executed. Therefore, we can reuse the operations built in the previous step by removing the first action (as it has already been executed) and appending a wait action at the end. This preserves the operation length and ensures that these operations remain collision-free.

It is important to note that operation inheritance is used only for the initialization of operations. Agents are still free to select any operation, and there is no requirement that the newly chosen operation must match the previous one in its initial actions. This initialization helps to reduce the number of collisions among agents. Furthermore, if some agents fail to find collision-free operations in the current step, they continue to execute the operations found in the previous step, rather than simply waiting in place.

## Pseudocode

The pseudocode for EPIBT is shown in Algorithm 1 (Main Loop) and Algorithm 2 (Operation Selection Procedure), which together capture all EPIBT enhancements. The main loop takes as input the grid-graph $G$, current agent states $\{s_1, \ldots, s_n\}$, goal locations $\{g_1, \ldots, g_n\}$, inherited operations $\{a'_1, \ldots, a'_n\}$, and revisit limit $L$.

First, the chosen operations $\{a_1, \ldots, a_n\}$ and agent paths $P$ (the states visited while executing actions) are initialized (lines 2–4). Agents are then prioritized by their current distance to goal (lines 5–6). The main loop iterates over unvisited agents, allowing each to select an operation. If no collision-free operation is found, the agent reverts to its inherited operation (lines 8–12).

Algorithm 2 details the operation selection. Each agent considers operations in order of $w_{op} = h(s_k, op, g_k) \cdot \alpha +$

Algorithm 1: EPIBT - Main Loop

```
1: **Input:** $G$, $\{s_1, \ldots, s_n\}$, $\{g_1, \ldots, g_n\}$, $\{a'_1, \ldots, a'_n\}$, $L$
2: $a_k \leftarrow a'_k$ for $k = 1, \ldots, n$
3: $visited_k, hit_k \leftarrow \{0\}$ for $k = 1, \ldots, n$
4: $P \leftarrow$ getPath$(s_k, a_k)$ for $k = 1, \ldots, n$
5: $p_k \leftarrow dist(s_k, g_k)$ for each agent $k = 1, \ldots, n$
6: $Agents \leftarrow$ sort $\{1, \ldots, n\}$ by priorities $p_k$
7: **for** $k \in Agents$ **do**
8:     **if** $visited_k \neq 0$ **then continue**
9:     $P \leftarrow P \setminus$ getPath$(s_k, a_k)$
10:    **if** EPIBT$(k, p_k) = failed$ **then**
11:        $P \leftarrow P \cup$ getPath$(s_k, a'_k)$
12: **return** $\{a_1, \ldots, a_n\}$
```

Algorithm 2: EPIBT - Operation Selection Procedure

```
1: **Input:** agent $k$, priority $p$
2: $OP \leftarrow$ sort $op \in Operations$ by weights $w_{op}$
3: $visited_k \leftarrow visited_k + 1$; $hit_k \leftarrow 1$
4: for $op \in OP$ do
5:     if getPath$(s_k, op) \not\subset G$ then continue
6:     if getUsed$(s_k, op, P) = \emptyset$ then
7:         $a_k \leftarrow op$; $hit_k \leftarrow 0$
8:         $P \leftarrow P \cup$ getPath$(s_k, op)$
9:         return $success$
10:    if $|$getUsed$(s_k, op, P)| > 1$ then continue
11:    $l \leftarrow$ getUsed$(s_k, op, P)$
12:    if $hit_l = 1$ or $visited_l \geq L$ or $p_l \leq p$ then
13:        continue
14:    $P \leftarrow P \setminus$ getPath$(s_l, a_l) \cup$ getPath$(s_k, op)$
15:    $a_k \leftarrow op$
16:    if EPIBT$(l, p)$= $success$ then
17:        $hit_k \leftarrow 0$; return $success$
18:    $P \leftarrow P \setminus$ getPath$(s_k, op) \cup$ getPath$(s_l, a_l)$
19: $a_k \leftarrow a'_k$; $hit_k \leftarrow 0$
20: return $failed$
```

$\beta_{op}$, where $h(s_k, op, g_k)$ is a function returning the distance to goal $g_k$ after performing operation *op* from state $s_k$, and $\alpha$ and $\beta_{op}$ are weighting coefficients. In our implementation, $\alpha$ is set to a high value, making $\beta_{op}$ primarily a tie-breaker when two states have the same $h$-value. The $\beta$ values are chosen such that movement actions are most preferred, followed by rotations, and finally wait actions (line 2). The agent is then marked as visited for this call, and a flag is set to prevent multiple visits within the same recursion branch (line 3).

For each viable operation, `getPath`$(s_k, op)$ returns the sequence of states. Operations leading to obstacles or outside the grid are skipped (line 5). Valid operations are checked for agent collisions using `getUsed`$(s_k, op, P)$, which returns IDs of agents that would collide with $k$. Collision-free operations are adopted (lines 6–9). Operations causing collisions with two or more agents are skipped (line 10). For a single-agent collision ($l$), the operation is skipped if $l$ was already visited in this recursion branch, exceeded its revisit limit, or has higher priority (lines 12–13). Otherwise, the algorithm attempts to rebuild $l$'s actions, considering $k$'s operation. If successful, $k$ adopts the operation and returns success (lines 15–17); otherwise, $P$ is reverted (line 18) and the next operation is tried. If all options fail, the agent is assigned its inherited operation $a'_k$ (line 19), and failure is returned (line 20).

## Large Neighborhood Search

EPIBT produces valid, collision-free actions within milliseconds, but these are generated in a prioritized, order-dependent manner. To further improve solution quality and utilize the remaining computational time, we incorporate a Large Neighborhood Search (LNS) optimization. Similar approach, combining windowed PIBT with LNS, have been used in WPPL (Jiang et al. 2024), and LNS-based methods have also been proposed for classical MAPF (Li et al. 2021a, 2022). Our goal is not to advance LNS itself, but to show that combining EPIBT with LNS can substantially boost performance, even surpassing state-of-the-art solvers for online LMAPF-T.

At each LNS iteration, a random agent $k$ is selected, its operation and path are removed, and a the operation selection procedure is performed. To enable diverse solutions across runs, the agent $k$ gets the highest priority, allowing it to override the others. If a new solution is found, we accept it if it improves the LNS metric, defined as the sum of $w_{op} \cdot p_k$ for all agents, where $p_k$ reflects proximity to the goal. This metric encourages agents to complete tasks quickly and move on to new ones. The optimization process continues until the time budget is exhausted.

## Theoretical Analysis

In this section, we demonstrate and prove that none of the proposed modifications made in EPIBT violate the major properties of the original PIBT algorithm. Note that the properties of PIBT are valid only when all adjacent nodes in the graph have a simple cycle $C$ such that $|C| \geq 3$. The EPIBT approach requires this assumption as well.

First, we prove the following lemma, similar to Lemma 1 in (Okumura et al. 2022):

**Lemma 1**. Let $k$ denote the agent with the highest priority at timestep $t$, and let $(i, j)$ be the location nearest to $g_k$ among the neighbors of $s_k$. In the worst case, agent $k$ will reach location $(i, j)$ within $t + 3$ timesteps.

For simplicity, we consider only operations of length 3. The logic of the proof can be adapted to the higher lengths of operations by adding wait actions to the end of operations.

**Proof**. Consider the very first timestep or the case when no operation inheritance enhancement is applied. All agents have the initial operation WWW, which is guaranteed to be collision-free. Agent $k$ with the highest priority chooses its operation first, while all other agents still have predefined operations WWW.

The optimal operation might be FFF, FCF, or similar operations that involve more than one movement action, potentially causing collisions with multiple agents. Following the logic of Algorithm 2 (line 10), such operations are skipped. However, since location $(i, j)$ is an adjacent cell near the current state $s_k$, it can be reached by one of the following operations: WWF, CWF, RWF, or RRF. These four operations cover all adjacent locations, one of which is guaranteed to be closer to $g_k$ than state $s_k$. Moreover, all these operations have only one movement action performed as the final action, which means that: (i) they may result in collision with at most one agent, and (ii) the colliding agent has two extra timesteps to change orientation before freeing the location.

If the desired location $(i, j)$ is free, agent $k$ can obtain the corresponding operation (lines 6-9 of Algorithm 2) and will not be forced to change the chosen operation since it has the highest priority. Otherwise, a recursive call of the EPIBT procedure for agent $\ell$ (currently occupying location $(i, j)$) is made (line 16 of Algorithm 2).

The only restriction for agent $\ell$ is that it cannot collide with agent $k$. Since all locations have at least two traversable adjacent cells (due to our assumption about simple cycles with length $\geq 3$), agent $\ell$ can move to some location $(i', j')$ that differs from both location $(i, j)$ and the location of state $s_k$. Even if agent $\ell$ has a wrong orientation and cannot immediately move to location $(i', j')$, it has two extra timesteps to change orientation before movement.

If location $(i', j')$ is also occupied by some agent $m$, it can be pushed by agent $\ell$ out of this location, considering priority inheritance. Recursively applying the logic of procedure EPIBT, agent $m$ can free the occupied location within at most 3 timesteps, and agent $\ell$ can wait in its current location if necessary (by considering one of the four operations mentioned above). Thus, regardless of the number of affected agents, each can move out of the occupied location within 3 timesteps in the worst case. Therefore, agent $k$ with the highest priority will be able to occupy a location closer to $g_k$ than the current state $s_k$ within at most 3 timesteps.

Applying operation inheritance leads to changes in the initial operations that agents have. However, the inherited operations remain collision-free, meaning that even in the worst case, the agent with the highest priority can choose an operation that replicates the inherited operation and changes only the last action. Such operation may collide with at most one agent. If such a collision exists, it occurs at the end of the operation and involves a single agent $\ell$. Thus, it can be successfully resolved by pushing agent $\ell$ away. The same logic applies to the operation choices of the remaining agents, which can be recursively visited while freeing the cell for agent $k$. In the worst case, they may all replicate the inherited operations, changing only the last action. Therefore, operation inheritance does not violate the property that agent $k$ with the highest priority needs at most 3 timesteps to get closer to its goal location $g_k$, even in the worst case. $\square$

Second, we perform an analysis of the time complexity of EPIBT.

**Proposition 1**. The time complexity of EPIBT in one timestep is $O(n(logn + |OP| \cdot log|OP| + L \cdot |OP| \cdot op\_len))$, where $n$ stands for the number of agents, $|OP|$ – the number of operations considered, $op\_len$ – operations length, and $L$ – revisit limit.

**Proof**. Here we assume (and made it in our implementation) that all distance matrices are precalculated. Thus, the procedure of agents sorting in the main loop of EPIBT costs $O(n \cdot logn)$, and the sorting of operations costs $O(|OP| \cdot log|OP|)$. Procedures related to collision checks and reservation updates depend on the operation length, and thus cost $\Theta(op\_len)$. Each operation selection procedure call examines at most $|OP|$ operations that result in $O(|OP| \cdot log|OP| + |OP| \cdot op\_len)$. The sorting of operations is performed once for each agent, while operation selection is performed up to $L$ times, since each agent is visited at most $L$ times per timestep. Therefore, the per-timestep complexity is $O(n(logn + |OP| \cdot log|OP| + L \cdot |OP| \cdot op\_len))$. $\square$

Lastly, in (Okumura et al. 2022), a theorem was proved guaranteeing that all agents will reach their goal locations in a finite number of steps. However, it is not guaranteed that all goal locations will be occupied by the corresponding agents simultaneously, which is required by the classical one-shot MAPF problem. This property is achieved by modifying priorities: each agent that reaches its goal location receives a lower priority than other agents that have not yet reached their goals. Thus, the agent with the highest priority is always one that has not yet reached its goal location, until no such agent exists.

In case of LMAPF neither simultaneous nor eventual goal reaching is required. Nevertheless, EPIBT can utilize the same priority mechanism presented in the original PIBT approach and obtain the same property if required.

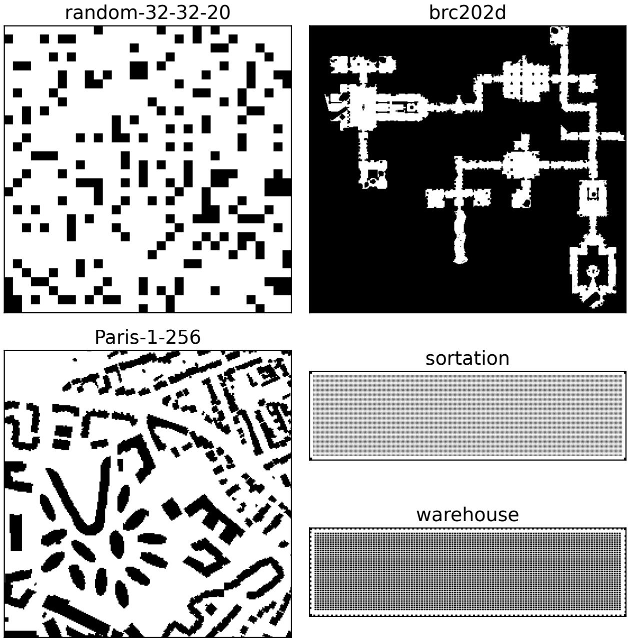

Figure 3: Visualization of maps used for the empirical evaluation.

| Map | Size | $\|V\|$ | Agents num |
|---|---|---|---|
| random-32-32-20 | 32x32 | 819 | 100, 200, ..., 800 |
| Paris-1-256 | 256x256 | 47240 | 1000, 2000, ..., 10000 |
| brc202d | 481x530 | 43151 | 500, 1000, ..., 5000 |
| sortation | 140x500 | 54320 | 1000, 2000, ..., 10000 |
| warehouse | 140x500 | 38586 | 1000, 2000, ..., 10000 |

Table 2: Detailed information about evaluated maps and instances.

## Empirical Evaluation

We evaluated our approach on the same set of five maps used in the League of Robot Runners competition (Chan et al. 2024): `random-32-32-20`, `brc202d`, `Paris-1-256`, `sortation`, and `warehouse`. Most maps originate from the well-known MAPF benchmark (Stern et al. 2019). Figure 3 shows their layouts. For each map, we ran multiple instances with varying agent counts, as detailed in Table 2. Goal sequences and their order were taken directly from the competition's public archive.

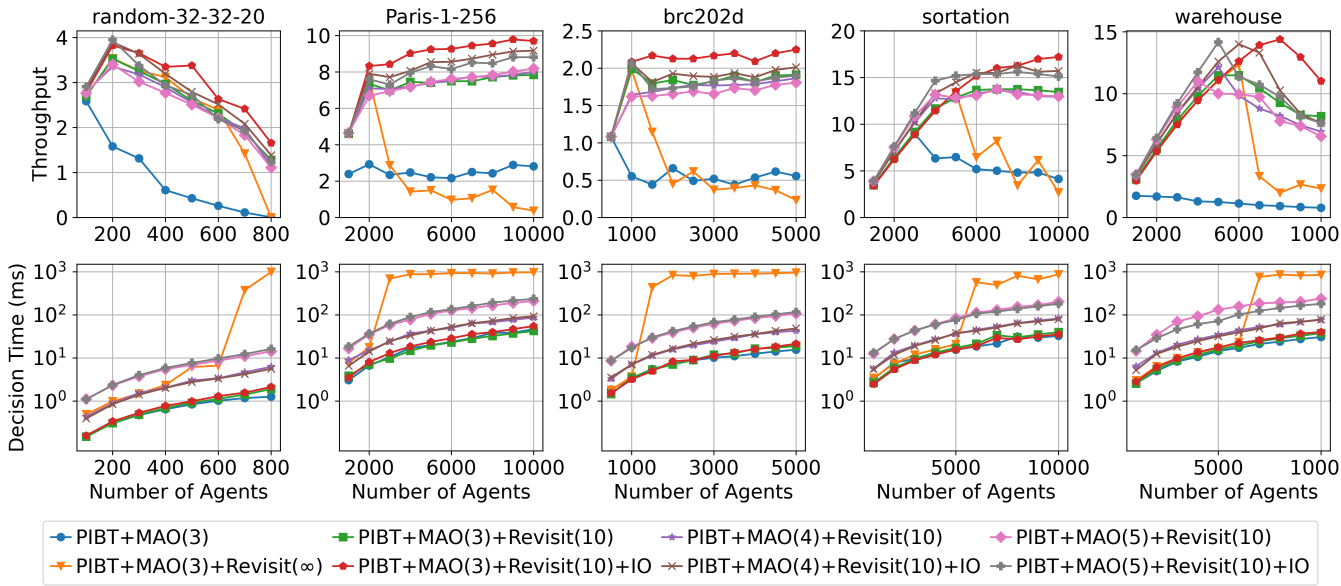

Figure 4: Ablation study of different enhancements incorporated into EPIBT and its evaluation with different operation lengths.

To eliminate any inter-agent influence caused by goal assignment, we used the following logic: agent $k$ receives tasks with indices $k$, $k + \text{total\_agents}$, $k + 2 \cdot \text{total\_agents}$, etc., cycling through the task pool as needed. The time budget was set to 1 second per timestep for all experiments. All methods were run single-threaded on an Intel Xeon Gold 6338 CPU with 256 GB RAM. Timestep horizon $T$ for each experiment was set to 5,000 timesteps, except for `random-32-32-20`, which used 1,000.

## Ablation Study

We first evaluated several EPIBT variants to assess the impact of each enhancement – multi-action operations (MAO), revisiting, and inheritance of operations (IO) – as well as the effect of operation length on runtime and performance. EPIBT was tested with operation lengths of 3, 4, and 5.

Results are shown in Figure 4. Here, MAO($x$) indicates operation length $x$, and Revisit($y$) denotes a revisit limit $L = y$. Adding MAO alone to PIBT yields poor performance, as agents cannot revise their choices. Introducing revisiting greatly improves results, but unlimited revisiting can cause the algorithm to hit the time budget, degrading performance. Limiting revisits stabilizes both throughput and runtime. Incorporating IO further boosts performance, regardless of operation length.

Longer operations increase runtime and only occasionally improve throughput (notably on large maps like `sortation` or `warehouse` with 5,000 agents). In most cases, length 3 yields the best throughput, likely because longer operations lead to more multi-agent collisions, which cannot be resolved by the current approach. Thus, agents may be forced to select suboptimal actions to avoid such collisions. Notably, even with 10,000 agents, EPIBT computes the next action in under 100 ms, with potential for further optimization.

In the next experiments, we use the fully enhanced version, denoted as EPIBT($x$), where $x$ is the operation length.

## EPIBT on LMAPF-T

We compared EPIBT to PIBT-like approaches applicable to LMAPF-T. Since original PIBT cannot be directly used, we adapted it to consider only five operations – FWW, RFW, CFW, RRF, and WWW – mimicking the omnidirectional action model, where moving to some adjacent cells requires 2–3 actions. Thus, we do not consider MAO with length less than 3 on LMAPF-T, as shorter operations are insufficient. We also evaluated Causal PIBT, an event-based variant that handles multi-timestep actions. Aggregated throughput results are shown in Figure 1. This spider plot averages normalized results across all instances for each map and includes additional approaches discussed later. The results clearly show that EPIBT consistently outperforms both PIBT and Causal PIBT across all maps.

## EPIBT on LMAPF

We also evaluated EPIBT under the omnidirectional action model. Here, the number of unique cell sequences is $5^{\text{length}}$ and cannot be reduced, as each action leads to a distinct location. In this setting, EPIBT(1) and EPIBT(2) are feasible, since all adjacent cells are reachable in one step. As baselines, we tested original PIBT and winPIBT (Okumura, Tamura, and Défago 2019), which plans for $w$ steps ahead. Following (Okumura, Tamura, and Défago 2019), we set $w = 10$, as this window size yields better results without runtime penalty.

Figure 6 presents the results. winPIBT performs well on the `random-32-32-20` map, but it reaches the time budget limit on the rest of the maps and as a result demonstrates poor performance. This likely occurs due to running A* for each agent, making it impractical for online LMAPF with

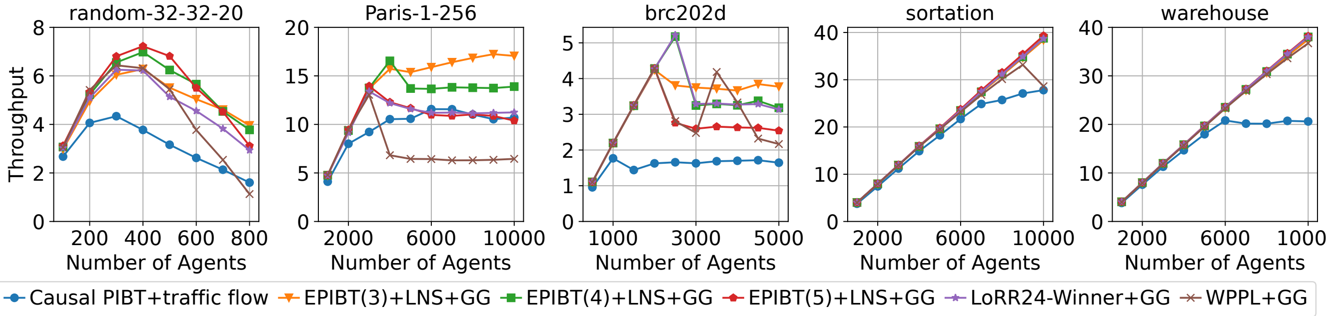

Figure 5: Comparison of EPIBT+LNS+GG with operation lengths 3, 4, and 5 with both winners of LoRR competition and Causal PIBT+traffic flow on the online LMAPF-T setting.

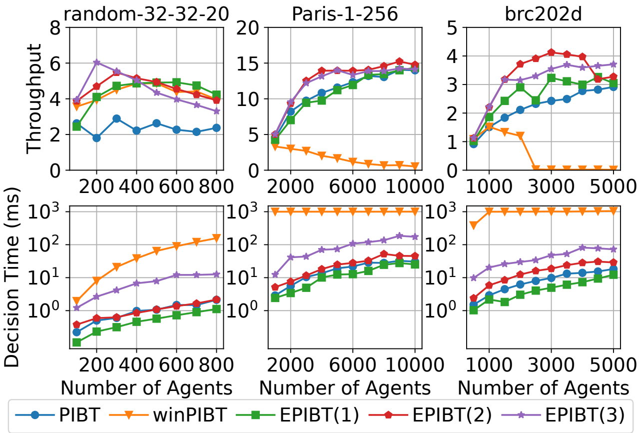

Figure 6: Comparison of EPIBT with original PIBT and winPIBT on the online LMAPF setting.

a 1-second time budget. The original PIBT approach underperforms on `random-32-32-20`, but is able to compete with EPIBT(1) on `Paris-1-256`. This is explained by the fact that the `random-32-32-20` map contains deadlocks, i.e., some of the cells have no cycle of 3+ size. As a result, some of the agents fail quite frequently and PIBT cannot rechoose their actions, as revisiting is not allowed. Such deadlocks are not present on large maps, which partially eliminates the advantage that EPIBT(1) gains from revisits. Utilizing operation length 2 for LMAPF is definitely beneficial. However, further increments in operation length result in a significant increase in runtime and do not always result in improvement in terms of throughput. Due to space limitations, the results on only 3 out of 5 maps are displayed. However, the results on the remaining 2 maps are similar to the ones obtained on `Paris-1-256` map.

## EPIBT+LNS

We finally compared EPIBT combined with LNS to state-of-the-art approaches for online LMAPF-T. The first baseline is WPPL (Jiang et al. 2024), the winning solution of LoRR23. The second is Causal PIBT with traffic flow (Chen et al. 2024), used as the default planner in LoRR24. The third, which we refer to as LoRR24-Winner, is the winning solution of LoRR24. The latter is a method created by us for the competition. It is also based on an EPIBT variant, but it has no operation inheritance and limits agent revisits in a different way. Both WPPL and LoRR24-Winner employ Graph Guidance (GG), which reduces congestion by weighting transition costs. We also incorporated GG into EPIBT+LNS, using the same weights as LoRR24-Winner. All methods were evaluated with and without GG.

Aggregated results are shown in Figure 1. EPIBT(3)+LNS+GG achieves score 1.0 on 3 out of 5 maps, demonstrating that none of the competitors is able to outperform it in those maps. On the remaining two maps (random-32-32-20 and brc202d), other methods occasionally achieve higher scores, particularly on low-agent instances. However, EPIBT(3)+LNS+GG still outperforms others in most cases. Detailed results, including EPIBT+LNS+GG with operation lengths 4 and 5, are provided in Figure 5. No runtime plot is shown, as all approaches used the full 1-second time budget per timestep.

Overall, these results demonstrate that EPIBT, when combined with LNS and GG, can outperform current state-of-the-art methods for online LMAPF-T.

## Conclusion

We introduced EPIBT, a new modification of the PIBT approach tailored for the rotation action model. The key innovation is the use of multi-action operations, enabling the algorithm to consider multiple timesteps at once without invoking a separate path-planning routine, unlike windowed methods such as winPIBT. Additional enhancements, including limited revisiting and operation inheritance, further improve performance and efficiency.

Empirical results show that these enhancements significantly boost throughput while maintaining fast runtime. Moreover, combining EPIBT with Large Neighborhood Search and Graph Guidance yields performance surpassing all existing methods, including both League of Robots Runners competition winners, establishing a new state-of-the-art for online LMAPF-T. Future work includes improving multi-agent collision handling and further optimizing the integration with LNS and GG.

# 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. 2024. 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.

Dhaliwal, A. 2020. The rise of automation and robotics in warehouse management. In *Transforming management using artificial intelligence techniques*, 63–72. CRC Press.

Ferreira, B.; and Reis, J. 2023. A systematic literature review on the application of automation in logistics. *Logistics*, 7: 80.

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.; 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)*.

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.

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.

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.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021a. Anytime multi-agent path finding via large neighborhood search. In *International Joint Conference on Artificial Intelligence 2021*, 4127–4135.

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.; 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.; 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.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. S.; and Koenig, S. 2021b. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11272–11281.

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.

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.; Tamura, Y.; and Défago, X. 2019. winpibt: Extended prioritized algorithm for iterative multi-agent path finding. *arXiv preprint arXiv:1905.10149*.

Okumura, K.; Tamura, Y.; and Défago, X. 2021. Time-independent planning for multiple moving agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11299–11307.

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

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.

Silver, D. 2005. Cooperative pathfinding. In *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, volume 1, 117–122.

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.

Veerapaneni, R.; Saleem, M. S.; Li, J.; and Likhachev, M. 2025. Windowed MAPF with Completeness Guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 23323–23332.