Lightweight and Effective Preference Construction in PIBT for Large-Scale Multi-Agent Pathfinding

Keisuke Okumura[1,2], **Hiroki Nagai**[1,3]
[1]National Institute of Advanced Industrial Science and Technology (AIST), Japan
[2]University of Cambridge, UK
[3]Keio University, Japan
{okumura.k, nagai.hiroki39}@aist.go.jp

Abstract

PIBT is a computationally lightweight algorithm that can be applied to a variety of multi-agent pathfinding (MAPF) problems, generating the next collision-free locations of agents given another. Because of its simplicity and scalability, it is becoming a popular underlying scheme for recent large-scale MAPF methods involving several hundreds or thousands of agents. Vanilla PIBT makes agents behave greedily towards their assigned goals, while agents typically have multiple best actions, since the graph shortest path is not always unique. Consequently, tiebreaking about how to choose between these actions significantly affects resulting solutions. This paper studies two simple yet effective techniques for tiebreaking in PIBT, without compromising its computational advantage. The first technique allows an agent to intelligently dodge another, taking into account whether each action will hinder the progress of the next timestep. The second technique is to learn, through multiple PIBT runs, how an action causes regret in others and to use this information to minimise regret collectively. Our empirical results demonstrate that these techniques can reduce the solution cost of one-shot MAPF and improve the throughput of lifelong MAPF. For instance, in densely populated one-shot cases, the combined use of these tiebreaks achieves improvements of around 10-20% in sum-of-costs, without significantly compromising the speed of a PIBT-based planner.

Introduction

Multi-agent pathfinding (MAPF) (Stern et al. 2019) is a planning problem that assigns collision-free paths to each agent on a discrete workspace, and is seen as a future driving force of efficient logistics automation (Ma et al. 2017; Li et al. 2021). Although real-world applications often require planning algorithms to cope with swarms of hundreds or thousands of robots (Brown 2022), synthesising optimal coordination is computationally challenging (Yu and LaValle 2013; Banfi, Basilico, and Amigoni 2017). It is therefore important to develop suboptimal methods that are scalable, but can produce sufficiently plausible solutions with fewer redundant agent movements within tight time constraints. Indeed, developments in suboptimal MAPF methods over the last decade have been remarkable, allowing us to obtain collision-free paths for hundreds, sometimes thousands, of agents in sub-seconds (Li et al. 2022; Okumura 2023b).

Among advanced suboptimal methods, PIBT (Priority Inheritance with Backtracking) (Okumura et al. 2022) has gained notable popularity in recent years. It is a computationally lightweight function that generates a collision-free, transitionable *configuration*, i.e., locations for all agents, given another. Due to its simplicity and scalability, PIBT has been serving as a core component in state-of-the-art MAPF studies, such as massively scalable search-based algorithms with theoretical guarantees (Okumura 2023b,a), a winning strategy (Jiang et al. 2024b) of the first MAPF competition held in 2023 (Chan et al. 2024), a lifelong MAPF approach with hundreds of times faster runtime than a search-based method with comparable solution quality (Zhang et al. 2024), and machine learning approaches that can handle densely populated instances, which are traditionally difficult (Veerapaneni et al. 2024; Jiang et al. 2024a). In this sense, the improvement of the PIBT scheme has a significant impact on practical solutions for large-scale MAPF systems.

Implementation-wise, for each configuration generation PIBT uses a sorted list of candidate actions for each agent, herein termed *preference*. For an agent i within a graph $G = (V, E)$ at a vertex $u \in V$, an action corresponds to a neighbouring vertex v, including u itself, which represents 'staying' motion. These candidates are then typically sorted by the shortest path distance on G, denoted as dist, to a target vertex $g_i \in V$. Formally, they are sorted in a lexicographic and ascending order with

$$\langle \mathsf{dist}(v, g_i), \epsilon \rangle \tag{1}$$

where ϵ is a random number to break ties.

This preference construction ensures that agents move towards their respective destinations, but it remains a bare minimum implementation. In fact, it is well known in the research community that Eq. (1) leads to greedy and short-sighted suboptimal behaviour, as evidenced by recent studies (Okumura 2024; Chen et al. 2024; Veerapaneni et al. 2024; Zhang et al. 2024; Zang et al. 2025) that improve PIBT preference in terms of solution quality. These studies can improve on the original PIBT, but at a significant computational cost from the original, which impairs the advantage of being able to instantly generate motions for thousands of agents or more.

Instead, we are interested in computationally lightweight yet effective preference construction in PIBT. In particular, this paper proposes to add two terms before the random tiebreaker, as:

$$\langle \mathsf{dist}(v, g_i), hindrance, regret, \epsilon \rangle \quad (2)$$

These interim terms are efficiently computed with an overhead no greater than the linear time complexity of PIBT itself, and therefore do not compromise the algorithm scalability. Nevertheless, this enhanced tiebreaking significantly improves the solution quality in MAPF problems. For example, in densely populated one-shot MAPF scenarios, we observe that a PIBT-based planner (i.e., LaCAM) with Eq. (2) achieves solution cost reductions of around 10-20% over Eq. (1). Furthermore, in a specific lifelong case with a map size of 32×32, including 10% obstacles and 400 agents, Eq. (2) contributes to $\geq 40\%$ throughput improvement over vanilla PIBT. This empirical evidence suggests that the enhanced tiebreaking may improve many state-of-the-art MAPF implementations based on PIBT with little engineering effort.

Intuitively, the first term, $hindrance$, is a one-step later estimate of whether an action hinders the progress of a neighbouring agent towards its goal, computed by $O(\Delta(G))$, where $\Delta(G)$ is the maximum degree of a graph G. The second term, $regret$, represents how an action affects the subsequent action choices of nearby agents within that configuration generation, which is learned by running PIBT a few times. The overhead is thus the time complexity of PIBT itself, which is practically sub-millisecond for each configuration generation, even with thousands of agents. With this combination, we make a minimal but effective effort to improve PIBT. In addition, the completeness guarantees in the original PIBT and LaCAM are maintained, as the proposed adjustments are just tiebreaking.

In what follows, the paper continues with the preliminaries, related work, technical details, empirical results and discussions in order.

Preliminaries

Problem Definition

This paper addresses both one-shot and lifelong versions with a classical MAPF fashion. The system consists of a set of agents $A = \{1, 2, \ldots, n\}$ and a graph $G = (V, E)$. All agents act synchronously according to a discrete wall-clock time. At each timestep, each agent can stay at its current vertex or move to a neighbouring vertex. Vertex and edge conflicts are considered, i.e. two agents cannot occupy the same vertex simultaneously, and two agents cannot swap their occupied vertices within one timestep move. Then, a *one-shot MAPF problem* aims to find a finite action sequence for each agent $i \in A$ to its assigned goal $g_i \in V$ from a given start $s_i \in V$. The solution quality is assessed by *Sum-of-Costs (SoC)* (aka. flowtime), which sums the travel time of each agent until it stops at the target location and no longer moves. In a *lifelong MAPF problem*, once an agent has reached its goal, a new goal is immediately assigned by an external, black-box task assigner, so planners need to operate the agents continuously. The solution quality is assessed by *throughput*, the number of task completions normalised by the operation period.

Algorithm 1: Original PIBT

input: configuration $\mathcal{Q}^{\text{from}}$, agents A
output: configuration $\mathcal{Q}^{\text{to}}$ (each initialized with $\bot$)

```
1:  for i ∈ A do; if Q^to[i] = ⊥ then PIBT(i)
2:  return Q^to

3:  procedure PIBT(i)
4:      P ← neigh(Q^from[i]) ∪ {Q^from[i]}
5:      sort P by some means                ▷ preference construction
6:      for v ∈ P do
7:          if collisions in Q^to supposing Q^to[i] = v then continue
8:          Q^to[i] ← v
9:          if ∃j ∈ A s.t. j ≠ i ∧ Q^from[j] = v ∧ Q^to[j] = ⊥ then
10:             if PIBT(j) = INVALID then continue
11:         return VALID
12:     Q^to[i] ← Q^from[i]; return INVALID
```

PIBT

PIBT is a function that maps a collision-free configuration $\mathcal{Q}^{\text{from}} \in V^{|A|}$ to another $\mathcal{Q}^{\text{to}}$. These configurations are ensured to be transitionable in the sense that each agent i can move from the current location $\mathcal{Q}^{\text{from}}[i]$ to $\mathcal{Q}^{\text{to}}[i]$ within one timestep move, without colliding.

Algorithm 1 provides the pseudocode. The assignment process is performed by calling a submodule (Lines 3–12) sequentially following a prefixed order of agents A (Line 1). PIBT then tries to assign each agent a location in order of its preference $\mathcal{P}$, a sorted list of available actions (Line 5) consisting of the currently occupying vertex and its neighbouring vertices, denoted as neigh. Each assignment is accompanied by a collision check (Line 7), ensuring a collision-free outcome. The agent order is dynamically adjusted when one agent j blocks the desired location of another agent i, which is implemented by a recursive call of the submodule (Line 9). This scheme is called *priority inheritance* as assignment priority is inherited from i to j. The priority inheritance call is followed by *backtracking* (Line 10), which notifies i whether j secures a feasible action; otherwise, i needs the reassignment and thus continues the for-loop operation. With this simple procedure, PIBT serves as an instant configuration generator.

Applying PIBT to either one-shot or lifelong MAPF is straightforward; repeat one-timestep planning with a preference creation using Eq. (1) with the current goal assignment. In addition, there is a search-based method called LaCAM (Okumura 2023b) for one-shot MAPF, which uses PIBT as a successor generation. The experiments use LaCAM for one-shot MAPF, and bare PIBT for lifelong scenarios.

Related Work

The importance of tiebreaking in preference construction was briefly mentioned in the original PIBT paper (Okumura et al. 2022), but not investigated in depth. As PIBT

has grown in popularity, researchers have investigated how to optimise the preference.

An effective scheme to improve solution quality takes the form of preparing a *guidance* graph (Zhang et al. 2024), which is typically represented by a weighted directed graph G_w over the original graph G. Instead of the naive preference on G, such a study makes PIBT follow a preference based on G_w. With carefully designed weights, either by handcrafted (Cohen, Uras, and Koenig 2015; Li and Sun 2023), computationally-heavy heuristic search (Okumura 2024; Chen et al. 2024), or machine learning approaches (Yu and Wolf 2023; Zhang et al. 2024; Zang et al. 2025), the guidance is able to create *global* traffic flow and mitigate congestion. The techniques presented in this paper are orthogonal to this notion, which aims to minimise ineffective *local* collective behaviour, and thus can co-enhance PIBT with the global guiadance.

Another direction of preference optimisation focuses on overcoming narrow corridor scenarios where two agents cannot exchange their positions. Such situations require long-horizon coordinated behaviour between paired agents, which cannot be synthesised by greedy and shortsighted PIBT. Several studies have successfully migrated this problem using topology analysis on G (Okumura 2023a; Matsui 2024; Zhou, Zhao, and Ren 2025), sometimes called the *swap* technique. Meanwhile, this study aims to improve solution quality in broader situations, not limited to narrow corridors. Similar to the guidance, the swap can co-enhance PIBT with the proposed techniques.

Recent work has discovered direct preference optimisation using data-driven approaches, in particular imitation learning from expert (near-)optimal algorithms (Veerapaneni et al. 2024; Jiang et al. 2024a). Although this direction is attractive, it requires a significant amount of offline preparation. Another concern with data-driven artefacts is the notably slow inference (Skrynnik et al. 2024) compared to the heuristic search-based preference of Eq. (1), which is a bottleneck when incorporating PIBT into a search scheme. Rather, Monte-Carlo sampling over PIBT, i.e. running vanilla PIBT several times with random tiebreaking and retrieving the best one as a successor node, is fast and effective with the search (Okumura 2024).

Hindrance

A *hindrance* term aims to achieve smarter yet effective dodge behaviour when another agent is about to move to the currently occupying location. In Fig. 1(a,c), the lower priority agent must avoid the higher priority agent by either going right or down. If it goes right (Fig. 1b), the low-priority agent still has to avoid the high-priority agent at the next timestep, resulting in a minimum of five steps to reach its goal. Meanwhile, if it goes down (Fig. 1d), this agent can reach the goal faster, i.e., in three steps. The preference construction with Eq. (1) cannot distinguish these two situations.

Intelligent tiebreaking in preference construction is possible by estimating how such an avoidance behaviour by the low-priority agent might still disrupt the progress of the high-priority agent in the next timestep. Specifically, this estimate, called *hindrance*, is computed by examining only

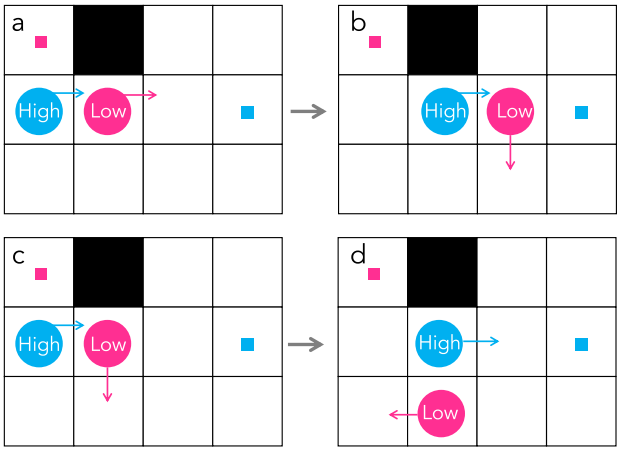

Figure 1: Motivating example for calculating the *hindrance* term. Goals for the agents are marked with coloured boxes.

Algorithm 2: Hindrance calculation

input: configuration $\mathcal{Q}$, agent $i \in A$, location $u \in V$

1: $hindrance = 0$
2: **for** $j \in \{k \in A \mid Q[k] \in \mathsf{neigh}(\mathcal{Q}[i])\}$ **do**
3: **if** $u \neq Q[j] \wedge \mathsf{dist}(u, g_j) < \mathsf{dist}(\mathcal{Q}[i], g_j)$ **then**
4: $hindrance \leftarrow hindrance + 1$
5: **return** $hindrance$

the distance relations between several vertices, as shown in Alg. 2. For a potential action $u \in V$ for an agent $i \in A$, the low-priority agent in Fig. 1, Alg. 2 checks whether u is heading towards a goal for a neighbouring agent $j \in A$, i.e., high-priority agent. Such an action is summed up as *hindrance*, enabling PIBT to distinguish between the cases of Fig. 1a and Fig. 1c. Note that the first condition in Line 3 does not penalise non-dodging behaviour, e.g. going left for the low-priority agent in Fig. 1a.

Time Complexity. Algorithm 2 is a computationally lightweight procedure. Under the assumption that $\mathsf{dist}(\cdot, g_j)$ is constant,[1] Alg. 2 has mere $O(\Delta)$ time complexity, where Δ is the maximum degree of a graph G, e.g., four in four-connected gridworld, on which MAPF methods are typically tested. Consequently, the total overhead for the PIBT step remains $O(|A| \cdot \Delta^2)$ because each agent computes *hindrance* once for each action.

Regret Learning

While *hindrance* improves how low-priority agents avoid high-priority ones, this section aims to improve how high-priority agents break ties among multiple actions. We calculate how each action affects *regret* of other agents, the cost gap between the best action and the action actually taken, and use it for tiebreaking. This term is expected to be effec-

[1]Most MAPF implementations first compute the distance table using backward Dijkstra, and then look up this table to retrieve the distance information in constant time during the search.

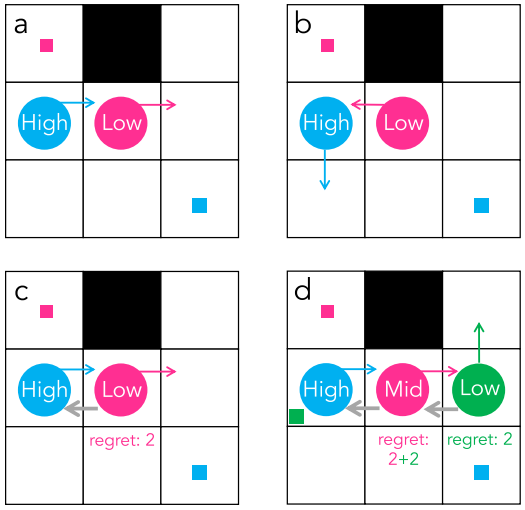

Figure 2: Motivating example for calculating the *regret* term. Grey arrows represent backtracking in PIBT.

Algorithm 3: PIBT with regret propagation

hyperparameters: $m \in \mathbb{N}_{>0}, 0 \leq w \leq 1$

1: **for** $i \in A$ **do** ▷ *initialise regret table*
2: **for** $v \in \mathsf{neigh}\left(\mathcal{Q}^{\mathrm{from}}[i]\right) \cup \left\{\mathcal{Q}^{\mathrm{from}}[i]\right\}$ **do**
3: $\mathcal{R}[i, v] \leftarrow 0$
4: **for** $1, 2, \ldots, m$ **do** ▷ *learning regret*
5: initialise $\mathcal{Q}^{\mathrm{to}}$
6: **for** $i \in A$ **do**; **if** $\mathcal{Q}^{\mathrm{to}}[i] = \perp$ **then** $\mathsf{PIBT}(i)$
7: **return** $\mathcal{Q}^{\mathrm{to}}$

8: **procedure** $\mathsf{PIBT}(i)$
9: $C \leftarrow \mathsf{neigh}\left(\mathcal{Q}^{\mathrm{from}}[i]\right) \cup \left\{\mathcal{Q}^{\mathrm{from}}[i]\right\}$
10: sort C with $\mathcal{R}[i, \cdot]$ as tiebreaker
11: **for** $v \in C$ **do**
12: **if** collisions in $\mathcal{Q}^{\mathrm{to}}$ supposing $\mathcal{Q}^{\mathrm{to}}[i] = v$ **then continue**
13: $\mathcal{Q}^{\mathrm{to}}[i] \leftarrow v$; $\mathit{regret} \leftarrow 0$
14: **if** $\exists j \in A$ s.t. $j \neq i \wedge \mathcal{Q}^{\mathrm{from}}[j] = v \wedge \mathcal{Q}^{\mathrm{to}}[j] = \perp$ **then**
15: $\mathit{validity}, \mathit{regret} \leftarrow \mathsf{PIBT}(j)$
16: $\mathcal{R}[i, v] \leftarrow (1 - w) * \mathcal{R}[i, v] + w * \mathit{regret}$
17: **if** $\mathit{validity} = \texttt{INVALID}$ **then continue**
18: $\mathit{regret}_i \leftarrow \mathsf{dist}(v, g_i) - \min_{u \in C} \mathsf{dist}(u, g_i)$
19: **return** VALID, $\mathit{regret} + \mathit{regret}_i$
20: $\mathit{regret}_i \leftarrow \mathsf{dist}(\mathcal{Q}^{\mathrm{from}}[i], g_i) - \min_{u \in C} \mathsf{dist}(u, g_i)$
21: $\mathcal{Q}^{\mathrm{to}}[i] \leftarrow \mathcal{Q}^{\mathrm{from}}[i]$; **return** INVALID, regret_i

tive in highly congested situations where the chain of priority inheritance is often triggered within PIBT.

Figure 2(a,b) describes example situations to see the importance of high-priority agent action selection. From the perspective of the high-priority agent, going either right or down brings the agent closer to its goal, while the low-priority agent cannot take the shortest path if the right action is chosen. Instead, the low-priority agent experiences *regret*, computed by $\mathsf{dist}(v, g_i) - \min_{u \in C} \mathsf{dist}(u, g_i)$, where v is the selected action and C is the set of candidate actions. With PIBT, it is easy to implement so that the high-priority agent can know *regret* of others through backtracking (Fig. 2c). The same scheme applies when priority inheritance occurs recursively (Fig. 2d). Therefore, when we run PIBT again, based on this information, the high-priority agent can choose the down action to avoid experiencing others with *regret*.

This regret learning is a non-stationary process, as in each PIBT run, each agent changes its action based on the current *regret* estimate, and then *regret* might be different even if one took the same action. Our simple solution to this is running PIBT several times, and over the iterations, we gradually update the estimates while incorporating the past estimates as a weighted average.

Algorithm 3 embodies the concept above, while greying out the same lines from the original PIBT (Alg. 1). Lines 1–3 first initialise the regret table $\mathcal{R}$, which records *regret* from others when an agent $i \in$ takes action $v \in V$. The regret table is learned through several PIBT runs (Lines 4–6), and the final run is used as output. The subprocedure requires minor modifications to include *regret* in backtracking, in addition to the validity of priority inheritance (Lines 19 and 21). For each priority inheritance call, the regret table is updated with a weighted sum of the previous and new values (Line 16).

Time Complexity. Regret learning is simply running the original PIBT several times, typically three times is sufficient to improve the solution quality. This does not significantly affect the speed advantage of PIBT, which runs $O(|A| \cdot \Delta)$ in its minimal form, empirically sub-millisecond procedure even with thousands of agents. Besides, regret learning is smarter than simply using the presence of another agent, which is used in the original PIBT paper (Okumura et al. 2022), because that ad-hoc rule does not necessarily involve regret. Monte-Carlo configuration generation (Okumura 2024) similarly runs PIBT several times and retrieves the best one according to the heuristic, but there is no guidance on sampling sequences to improve its quality over trials. Consequently, regret learning is more efficient in terms of sampling efficiency. We will empirically observe these claims in the next section.

Experiments

Through both one-shot and lifelong MAPF problems, we empirically evaluate several tiebreaking strategies on PIBT, including the proposed techniques, as listed below.

- We refer to `Original` as preference construction using the vanilla method, i.e. ascending order with Eq. (1).
- `MC`, which stands for Monte Carlo sampling, is a strategy introduced in the latest LaCAM implementation (Okumura 2024). From a given configuration $\mathcal{Q}$, it creates a batch $\mathcal{B}$ of k configurations using PIBT and Eq. (1) but with different random seeds. `MC` then selects the cost-minimising configuration as $\arg\min_{\mathcal{Q}' \in \mathcal{B}}(\mathsf{g}(\mathcal{Q}, \mathcal{Q}') + \mathsf{h}(\mathcal{Q}'))$, where g is a transition cost $|\{i \in A \mid \neg(\mathcal{Q}[i] = \mathcal{Q}'[i] = g_i)\}|$ and h is a heuristic $\sum_{i \in A} \mathsf{dist}(Q'[i], g_i)$. The experiments use $k = 10$ following the original work.
- `Vacancy` follows the PIBT paper (Okumura et al. 2022), which prioritises a vacant location if available. Formally, the preference is constructed with $\left\langle \mathsf{dist}(v, g_i), \mathrm{Ind}\left[\exists j \in A, v = \mathcal{Q}[j]\right], \epsilon \right\rangle$, where $\mathrm{Ind}[\cdot] = 1$ if the condition is true; zero otherwise.

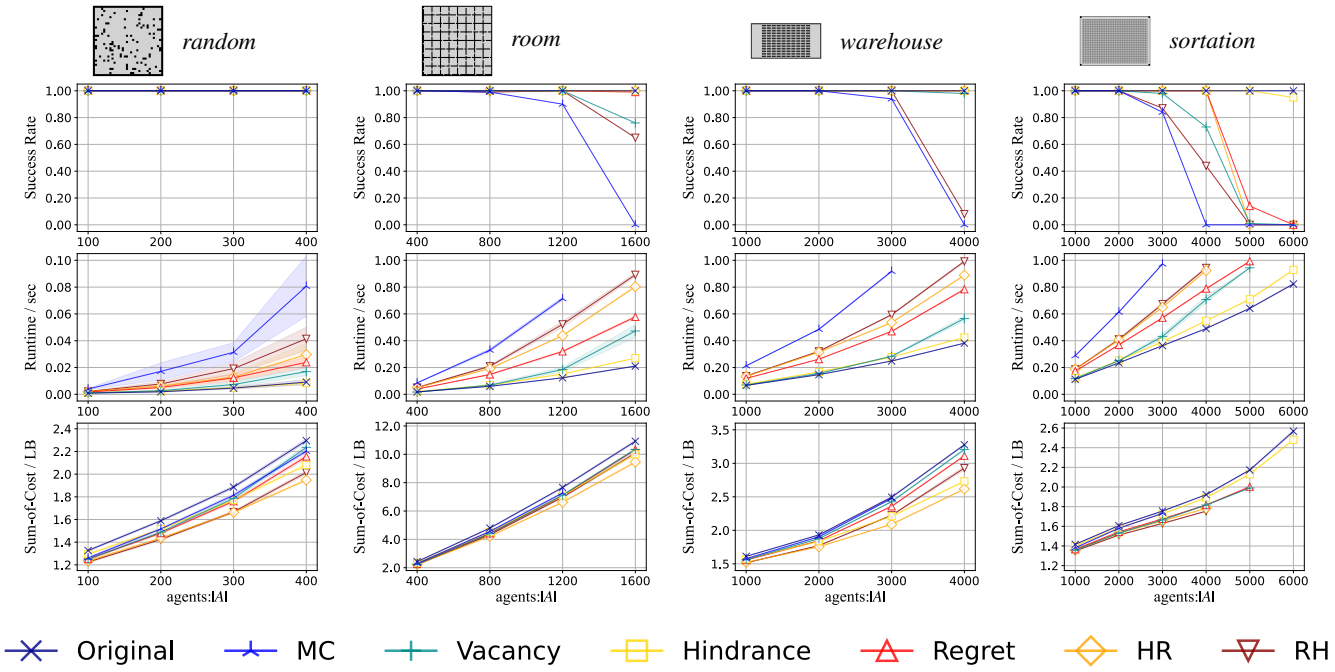

Figure 3: Results for one-shot MAPF. The success rate of planning within 1 s (top), average runtime (middle) and SoC normalised by lower bound (1.0 is minimum; bottom) for successful cases are shown.

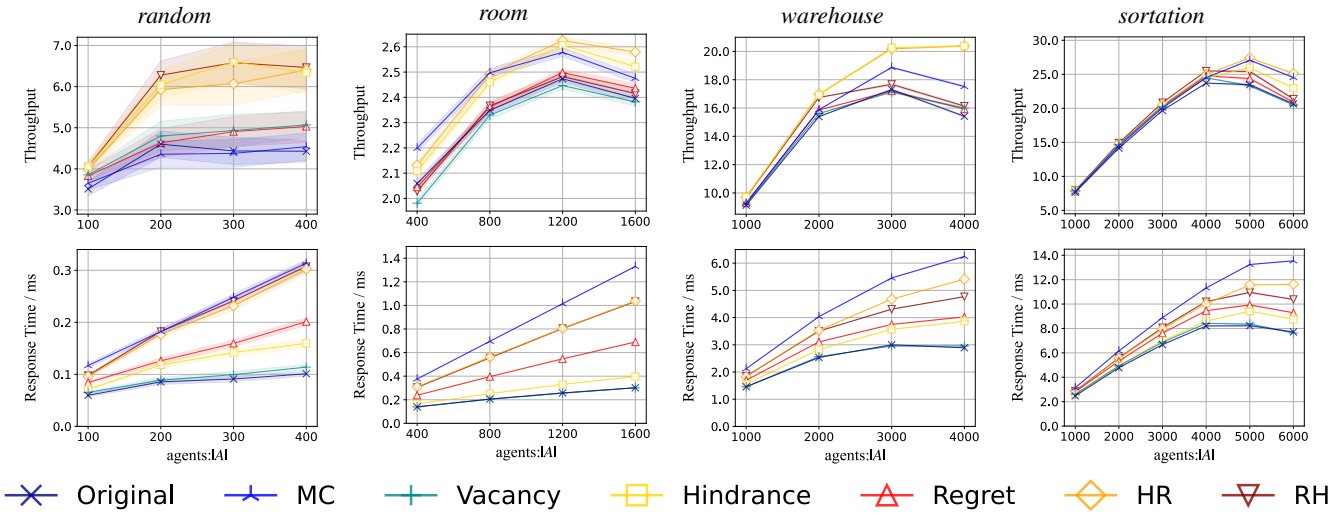

Figure 4: Results for lifelong MAPF. Average throughput (top) and response time (bottom) are shown.

- `Hindrance` uses $\langle \mathsf{dist}, \mathit{hindrance}, \epsilon \rangle$ for the preference construction.
- `Regret` uses $\langle \mathsf{dist}, \mathit{regret}, \epsilon \rangle$. Unless explicitly mentioned, the parameters are set to $m = 3$ and $w = 0.9$. These parameters have been adjusted following pilot studies to give reasonable results over a wide range of situations while minimising m.
- `HR` uses $\langle \mathsf{dist}, \mathit{hindrance}, \mathit{regret}, \epsilon \rangle$.
- `RH` uses $\langle \mathsf{dist}, \mathit{regret}, \mathit{hindrance}, \epsilon \rangle$.

The prioritisation scheme in PIBT follows the original paper.

Benchmarks. Experiments were conducted on a laptop equipped with an M3 Pro Apple Silicon chip and 18 GB of RAM, using the following four-connected grid maps:

- *Random* (*random-32-32-10*) is a 32×32 grid map with 10% obstacles and $|V| = 922$.
- *Room* (*room-64-64-8*) sizes 64×64 and $|V| = 3,232$.
- *Warehouse* (*warehouse-10-20-10-2-2*) represents a typical warehouse layout with 170×84 and $|V| = 9,766$.
- *Sortation*, with 200×140 and $|V| = 21,920$, has single cell obstacles evenly spaced for every two columns/rows.

The first three are from the MAPF benchmark (Stern et al. 2019), while the last is from (Chen et al. 2024). Their illustration is available in Fig. 3. The values reported were generated from 100 test cases, prepared for each map and each number of agents, with the start and goal positions randomised. The code is written in C++ and is available at https://github.com/HirokiNagai-39/pibt-tiebreaking.

One-shot MAPF

In the context of one-shot MAPF, there is a popular search algorithm called LaCAM (Okumura 2023b) on top of PIBT that outperforms naively running PIBT. We thus evaluate tiebreaking strategies through LaCAM. *Our implementation is based on the vanilla LaCAM from (Okumura 2023b), which does not use the advanced techniques introduced later in (Okumura 2023a, 2024) to isolate the effect of tiebreaking.* Note that the original LaCAM paper uses `Original` as its tiebreaking strategy.

We assess the tiebreaking effect with *(i)* planning success rate within 1 s of finding a feasible solution, *(ii)* wall clock time for finding a solution, and *(iii)* solution cost represented by SoC (sum-of-costs). The SoC value shown is normalised by the sum of the shortest path lengths for all agents between their starts and goals, i.e. the trivial lower bound, so the minimum is one. The 1 s timeout reflects the real-world demands of real-time planning in logistics systems, and is also used in the MAPF's League of Robot Runners competition (Chan et al. 2024).

Figure 3 summarises the results, showing that overall, both *hindrance* and *regret* tiebreakers improve the planning ability of LaCAM in dense and challenging MAPF scenarios involving several hundreds to thousands of agents, with little computational overhead. In other words, these tiebreaking strategies in PIBT serve to better guide the search towards the goal configuration with less redundant agent movements, compared to existing ones such as `MC` and `Vacancy`. The joint use of *hindrance* and *regret* further enhances the performance of LaCAM. Notably, `HR` achieves approximately 20% cost reduction from `Original` in *warehouse* with $4,000$ agents.

The *hindrance* term is particularly effective with minimal engineering. At the extreme end, `Hindrance` has a success rate of over 90% on *sortation* with $6,000$ agents as like `Original`, even with a strict time limit, still improving LaCAM's solution quality in *all* scenarios. Meanwhile, the quality of the resulting solution varies from map to map. In *random* and *warehouse*, LaCAM results in smaller SoC solutions with `Hindrance`, while `Regret` performs better in *sortation* in terms of the solution quality. We presume that this is because *sortation* has a structure that *regret* is effectively propagated due to its regular obstacle placements. These map-dependent results have a direct impact on the order in which these terms should be used, as we can see with `HR` and `RH`.

The *regret* term requires PIBT to be run several times, which does indeed affect the runtime and causes some timeout attempts in instances with massive agents. However, it should be noted that the absolute runtime difference remains within hundreds of milliseconds even with thousands of agents in the cases tested. Furthermore, `Regret` outperforms `MC` with a similar concept, which also requires multiple PIBT runs. This shows that *regret* captures complex interactions with agents better than 'blind' trials of `MC`, due to its improved backtracking process.

random, $|A| = 400$

m		$w = 0.5$	$w = 0.9$	$w = 0.95$
3	time	0.029±0.007	0.024±0.004	0.029±0.008
	SoC	2.161±0.017	2.155±0.017	2.157±0.016
10	time	0.136±0.037	0.110±0.025	0.112±0.019
	SoC	2.118±0.024	2.110±0.022	**2.101**±0.022
20	time	0.267±0.054	0.231±0.041	0.212±0.044
	SoC	**2.101**±0.021	2.114±0.028	**2.101**±0.023

warehouse, $|A| = 4,000$

m		$w = 0.5$	$w = 0.9$	$w = 0.95$
3	time	0.788±0.007	0.780±0.006	0.792±0.007
	SoC	3.122±0.010	3.112±0.010	3.112±0.010
10	time	2.876±0.084	2.871±0.115	2.817±0.107
	SoC	2.929±0.009	2.918±0.010	2.913±0.009
20	time	6.888±0.299	6.398±0.291	6.369±0.312
	SoC	**2.897**±0.011	2.904±0.010	2.901±0.010

Table 1: Effect on hyperparameters for `Regret` over 100 instances, resulting in all successful. The time unit is seconds. 10 s timeout is used.

Effect on Hyperparameters in Regret Learning. The construction of the *regret* term requires two hyperparameters: the number of learning iterations m and the weight w, to sum up the regret values from different iterations. Table 1 examines how these parameters affect the performance of `Regret` using two scenarios. `Regret` has m times the overhead of vanilla PIBT according to the time complexity analysis. The runtime results roughly follow this observation, with slight deviations due to LaCAM's search process. The SoC metric generally improves as m increases with more accurate regret learning, eventually reaching saturation. In contrast, the weight w is not particularly dominant.

Lifelong MAPF

Next, we evaluate tiebreaking strategies of PIBT through a popular variant of MAPF such that once an agent reaches the goal, a new goal is randomly assigned. There are two metrics of interest here: *(i)* throughput, the number of goals reached for each timestep, averaged over the operation period, $1,000$ in our case, and *(ii)* response time required to generate a plan for each timestep.

Figure 4 summarises the results. In all experimental conditions, the difference in response time between the different tiebreaking strategies is almost negligible ($\leq$10 ms) in absolute terms, even with thousands of agents. Meanwhile, the throughput is significantly improved with *hindrance* in all the maps tested; for example, `Hindrance`, `HR`, and `RH` all

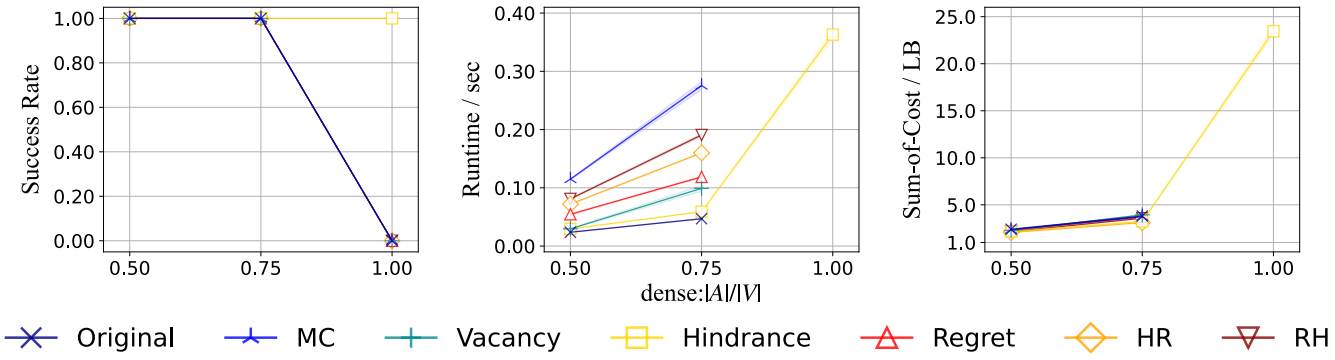

Figure 5: One-shot MAPF on *empty* in extremely dense situations.

empty, $\|A\| = \|V\| = 2,304$				
m		$w = 0.5$	$w = 0.9$	$w = 0.95$
20	time	3.837±0.067	3.657±0.057	3.649±0.047
	SoC	25.140±0.321	24.136±0.276	**24.129**±0.248
30	time	5.737±0.066	5.614±0.008	5.460±0.067
	SoC	26.234±0.196	25.145±0.195	24.874±0.201

Table 2: Performance of `Regret` with the extremely dense scenario where $|A|/|V| = 1$.

achieve $\geq 40\%$ throughput improvements with 400 agents in *random*. The *regret* term also has steady improvements over `Original`, but its effect is subtle compared to *hindrance*. We do not have solid interpretations for this; perhaps a one-step optimisation of the regret values would not correlate strongly with the throughput improvement.

Extremely Dense Situations

We further investigate the capability of newly developed tiebreaking strategies in extremely dense one-shot MAPF instances. Specifically, using the *empty* map (48×48; $|V| = 2,304$) taken from (Stern et al. 2019), we prepare instances with $|A|/|V| = \{0.5, 0.75, 1.0\}$, i.e., $1,152$, $1,728$, and $2,304$ agents, respectively. This time the time limit is set to 10 s, taking into account the difficulty.

Figure 5 presents the results. Remarkably, LaCAM with `Hindrance` was able to solve $|A|/|V| = 1$ instances with a success rate of 100%, within 1 s, while `Original` completely fails in the same setting. This observation provides further evidence that *hindrance* can serve as a stronger guide for the search than doing nothing to break a tie.

The failures in `HR` with high density imply that adding *regret* leads the search in the wrong direction. This is actually caused by immature regret learning with insufficient PIBT iteration specified by the parameter m, currently set to three. Table 2 presents this evidence, showing that increasing m allows LaCAM to solve extremely dense instances with `Regret` only. With $m = \{20, 30\}$, `Regret` results in a success rate of 100%, while lower m (e.g., 10) could not solve any instances regardless of w. Their solution quality is comparable to `Hindrance`, indicating that *regret* is a viable option in itself.

Discussions

Overall, adding *hindrance* consistently improves the performance of PIBT (`Original`) and outperforms the other strategies in terms of solution cost and computational overhead. The *regret* term is generally effective, but not as powerful as *hindrance*; nevertheless, in one-shot MAPF, it improves LaCAM compared to `MC`, which shares the concept of running PIBT several times. Regret learning is understood as a general scheme for `Vacancy`, and thus has a similar effect on planning. Meanwhile, it could have the potential to reflect complicated agent interactions thanks to regret propagation, as seen in Table 2. `MC` can enhance the vanilla PIBT and is especially promising in lifelong MAPF, but not stable in one-shot MAPF. We note that adding the so-called swap technique (Okumura 2023a) may clear out this instability, but still, *regret* could be a better alternative given the same amount of time budget.

Considering these observations, our suggestion is to use Eq. (2) as PIBT tiebreaking. In fact, `HR` consistently achieves superior solution quality among tested strategies in both one-shot and lifelong MAPF scenarios, with a slight runtime addition to `Original`.

Conclusion

This paper examined the technical details of PIBT that underpins modern large-scale MAPF studies. Our focus was tiebreaking of how each action is preferred by each agent. We proposed the hindrance metric and regret learning to easily improve the performance of PIBT with little additional computation. Empirical results in both one-shot and lifelong MAPF reveal significant impacts of tiebreaking, proving that the proposed strategy is an attractive replacement for leading MAPF implementations.

The future direction includes adaptive construction of the PIBT preference during the planning. This is motivated by our empirical results, which show that neither *hindrance* nor *regret* is always the best. There are several realisations, such as selecting the best strategies through in-search learning or updating the weight parameters online, as have been stud-

ied for large neighbourhood search (Phan et al. 2024).We also consider that directly optimising preferences with neural network policies is interesting (Veerapaneni et al. 2024; Jiang et al. 2024a). However, the inference with neural networks is notably slow compared to vanilla PIBT (Skrynnik et al. 2024), which impedes solving real-time and large-scale MAPF problems. Therefore, we believe that further developments of heuristic-based preferences, as presented in this paper, are of practical value.

Acknowledgement

This research was supported by a gift from Murata Machinery, Ltd.

References

Banfi, J.; Basilico, N.; and Amigoni, F. 2017. Intractability of time-optimal multirobot path planning on 2D grid graphs with holes. *IEEE Robotics and Automation Letters (RA-L)*.

Brown, A. S. 2022. How Amazon robots navigate congestion. Accessed: 2025-02-03.

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 *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*.

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

Cohen, L.; Uras, T.; and Koenig, S. 2015. Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In *Proceedings of Annual Symposium on Combinatorial Search (SoCS)*.

Jiang, H.; Wang, Y.; Veerapaneni, R.; Duhan, T.; Sartoretti, G.; and Li, J. 2024a. Deploying Ten Thousand Robots: Scalable Imitation Learning for Lifelong Multi-Agent Path Finding. *arXiv preprint arXiv:2410.21415*.

Jiang, H.; Zhang, Y.; Veerapaneni, R.; and Li, J. 2024b. Scaling Lifelong Multi-Agent Path Finding to More Realistic Settings: Research Challenges and Opportunities. In *Proceedings of Annual Symposium on Combinatorial Search (SoCS)*, volume 17.

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 AAAI Conference on Artificial Intelligence (AAAI)*.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. S.; and Koenig, S. 2021. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Li, M.-F.; and Sun, M. 2023. The Study of Highway for Lifelong Multi-Agent Path Finding. *arXiv preprint arXiv:2304.04217*.

Ma, H.; Li, J.; Kumar, T.; and Koenig, S. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*.

Matsui, T. 2024. Investigation of Heuristics for PIBT Solving Continuous MAPF Problem in Narrow Warehouse. In *Proceedings of International Conference on Agents and Artificial Intelligence (ICAART)*.

Okumura, K. 2023a. Improving LaCAM for Scalable Eventually Optimal Multi-Agent Pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Okumura, K. 2023b. LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Okumura, K. 2024. Engineering LaCAM*: Towards Real-Time, Large-Scale, and Near-Optimal Multi-Agent Pathfinding. In *Proceedings of International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. *Artificial Intelligence (AIJ)*.

Phan, T.; Huang, T.; Dilkina, B.; and Koenig, S. 2024. Adaptive anytime multi-agent path finding using bandit-based large neighborhood search. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

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 AAAI Conference on Artificial Intelligence (AAAI)*.

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 Annual Symposium on Combinatorial Search (SoCS)*.

Veerapaneni, R.; Wang, Q.; Ren, K.; Jakobsson, A.; Li, J.; and Likhachev, M. 2024. Improving Learnt Local MAPF Policies with Heuristic Search. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*.

Yu, G.; and Wolf, M. T. 2023. Congestion prediction for large fleets of mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Zang, H.; Zhang, Y.; Jiang, H.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Li, J. 2025. Online Guidance Graph Optimization for Lifelong Multi-Agent Path Finding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Zhang, Y.; Jiang, H.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2024. Guidance Graph Optimization for Lifelong Multi-Agent Path Finding. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Zhou, S.; Zhao, S.; and Ren, Z. 2025. Loosely Synchronized Rule-Based Planning for Multi-Agent Path Finding with Asynchronous Actions. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.