Lifelong LaCAM with Local Guidance for Lifelong MAPF

Tomoki Arita[1,2], **Keisuke Okumura**[2]

[1]Keio University, Japan
[2]National Institute of Advanced Industrial Science and Technology (AIST), Japan
{arita.tomoki, okumura.k}@aist.go.jp

Abstract

Local guidance has recently proven to be a powerful driver of empirical performance in real-time, suboptimal multi-agent pathfinding (MAPF), improving the scalable configuration-based solver *LaCAM*. By injecting informative spatiotemporal cues around each agent, local guidance mitigates congestion, reduces waiting, and remains scalable enough even with tight time budgets, yielding state-of-the-art performance for *one-shot* MAPF. This study asks whether the same benefits can be lifted to the *lifelong* setting (*LMAPF*), where tasks arrive continuously and improvements in per-step plans can increase task completion throughput over long horizons. We propose *LLLG*, *a Lifelong version of LaCAM enhanced with Local Guidance*, which employs a receding-horizon windowed planning framework and warm-starts guidance from the previous solution at each timestep. Our method scales effectively, maintains high throughput even in compact, dense environments, and surpasses existing planners, thereby pushing the frontier of real-time, lifelong MAPF.

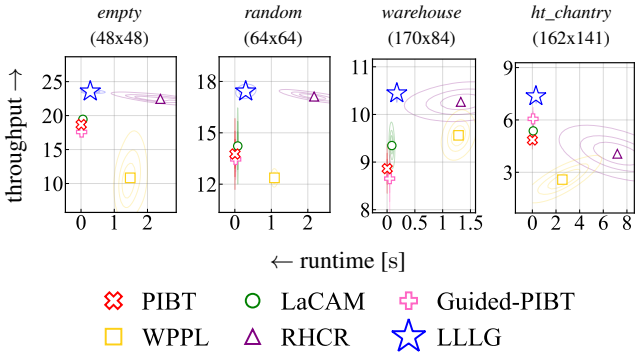

Figure 1: Performance of lifelong MAPF methods in representative scenarios, evaluated by throughput and runtime. All methods are evaluated over 500 steps with 800 agents across all maps.

1 Introduction

A central ambition in *multi-agent pathfinding (MAPF)* research is to realize capable multi-robot coordination, which in practice requires *lifelong* operation known as *LMAPF* (Ma et al. 2017; Chan et al. 2024). In this setting, agents do not stop after executing a single plan; rather, they continue to receive new goals (i.e., tasks) once they finish the current ones. Then, the objective is not only to maintain collision-free motion, but also to sustain high *throughput* over long horizons. Besides, deploying algorithms in the real world demands *real-time* planning: at each timestep, the solver must compute the next action in less wall-clock time than the robots need to execute that timestep.

Practical MAPF often unfolds in compact, dense environments with tens to hundreds of agents. In such settings, the planner frequently needs to resolve potential collisions around congested spaces within each planning window. A leading framework, *RHCR (Rolling Horizon Collision Resolution)* (Li et al. 2021b), is designed to produce near-optimal plans, yet its effectiveness remains in sparse environments. Indeed, RHCR can degrade sharply in dense settings. This is because it typically assumes CBS/PBS-style solvers (Sharon et al. 2015; Ma et al. 2018) and conflicts are handled through high-level branching and constrained replanning within each planning window. When conflicts are frequent, these repeated branch-and-replan steps can accumulate, causing severe slowdowns or failure.

In parallel, over the past several years, fast, suboptimal solvers have emerged that reliably maintain feasibility even under high density. Notable examples include *PIBT (Priority Inheritance with Backtracking)* (Okumura et al. 2022) and *LaCAM (Lazy Constraints Addition Search)* (Okumura 2023b), both of which scale well and deliver real-time decisions in scenarios where many other methods falter. PIBT functions as a fast one-step planner that synthesizes collective collision-free motions. Then, to avoid local deadlock or livelock, LaCAM solves long-horizon planning by leveraging this generator and performing a systematic search over configurations. While PIBT and LaCAM bring feasible solutions rapidly, they remain highly suboptimal in dense settings, where purely goal-directed action repeatedly induces *congestion* and degrades solution quality.

In summary, RHCR produces efficient solutions, but struggles in dense environments. In contrast, PIBT and LaCAM are stable, but highly suboptimal. To this end, this work proposes *Lifelong LaCAM with Local Guidance (LLLG)*, which quickly produces high-throughput solutions even in dense settings. Figure 1 highlights results across several compact, high-density LMAPF scenarios, demon-

strating that LLLG carves out a new performance frontier for LMAPF. For example, on *empty-48-48*, *random-64-64-10*, and *warehouse-10-20-10-2-2*, LLLG consistently attains higher throughput than RHCR while reducing runtime by approximately 90%, with only a modest runtime overhead relative to PIBT and LaCAM. Furthermore, on instances of *ht_chantry* with bottlenecks, where RHCR degrades sharply, LLLG achieves 81% higher throughput while reducing runtime by 96%.

This superior performance of LLLG is driven by *guidance*, which biases the underlying solver with costs designed to mitigate emerging congestion—for example, penalizing moves into predicted high-traffic cells/edges and excessive waiting, while rewarding detours or time shifts that distribute agents across space and time. Whereas many prior methods rely on *global* guidance that is derived from a map-wide view of the environment and typically time-independent (Chen et al. 2024; Zhang et al. 2024; Kato et al. 2025), recent work suggests that *local* guidance can be more effective by providing agent-centric spatiotemporal cues for imminent conflicts for *one-shot MAPF* (Arita and Okumura 2026). Following this insight, our LMAPF planner embeds local guidance into a windowed planning, receding horizon framework. In what follows, we present problem formulation, technical background, the proposed method, experimental evaluation, and discussion in order. The code is publicly available at https://github.com/allegorywrite/lllg.

2 Preliminaries

This section defines LMAPF, reviews key planning concepts, and summarizes the technical basis of our method.

A Problem Definition

We study lifelong multi-agent pathfinding (LMAPF) on an undirected graph $G = (V, E)$ with a team of agents $A = \{1, \dots, n\}$. Time is discrete, and agents move synchronously. At each timestep, each agent either waits or moves to an adjacent vertex. Collisions are interpreted in the standard MAPF sense (Stern et al. 2019): two agents may not occupy the same vertex at the same timestep, nor traverse the same edge in opposite directions simultaneously.

A *configuration* is the tuple of all agents' locations $\mathcal{Q} \in V^n$. Let $\mathcal{Q}^0 = (s_1, \dots, s_n)$ be the initial configuration, and let $g_i^t \in V$ denote agent i's current goal at time t. After initialization, each agent is assigned a new goal whenever it reaches its current goal. In our experiments, the goal is assigned randomly from all the vertices. A *solution* is an infinite collision-free sequence of configurations in which each agent moves to an adjacent vertex or waits at every timestep. We evaluate solution quality by *throughput*, measured as the average number of completed tasks per timestep.

B Planning Concepts for LMAPF

LMAPF admits several design choices in how planning and execution are interleaved. We summarize common concepts below (e.g., windowed planning) and use this taxonomy to position our method.

Naive Approach. The most straightforward strategy to LMAPF is to solve a one-shot MAPF problem at every timestep from the current configuration $\mathcal{Q}^t \in V^n$ to a goal configuration $\mathcal{G} = (g_1^t, \dots, g_n^t) \in V^n$, yielding a solution Π^t, and then execute only its first joint action $\Pi^t[1]$ to obtain $\mathcal{Q}^{t+1}$, e.g., (Shankar, Okumura, and Prorok 2025). This keeps the policy responsive to congestion and online goal updates, but it can be computationally demanding under strict real-time budgets.

Windowed Planning. To amortize computation, *windowed planning* solves a finite-horizon MAPF problem over a window of length $w_\Pi \in \mathbb{N}_{\geq 1}$ and optimizes only behavior within this window (Veerapaneni et al. 2025). Classic online solvers such as WHCA* (Silver 2005; Bnaya and Felner 2014) reserve paths only for the next w_Π timesteps, and modern lifelong methods such as WPPL (Jiang et al. 2024) likewise build coordination around finite-window optimization and refinement. One extreme is $w_\Pi{=}1$, which reduces planning to a purely myopic, one-step decision rule (e.g., PIBT; Okumura et al. 2022). At the other extreme, the naive approach described above corresponds to *full-horizon* planning with $w_\Pi{=}\infty$, whereas in lifelong settings the planning window typically spans only up to the agents' currently assigned goals.

Receding Horizon. The above discussion has focused on how to compute a plan. For real-time lifelong operation, however, such planning can be coupled with execution in a *receding horizon* manner, like model predictive control (MPC). With this fashion, at each timestep, the solver computes a plan from the current configuration, executes the first $h \in \mathbb{N}_{\geq 1}$ steps of that plan, and then replans from the updated configuration. Depending on the underlying planner, the computed plan may target the currently assigned goals over the full remaining horizon or only over a finite planning window (e.g., RHCR; Li et al. 2021b). One appealing choice is $h{=}1$, which replans every timestep and is often reasonable in LMAPF because it is highly responsive and does not require goal lookahead.

Incremental Planning. In receding horizon planning, much of the intermediate information computed in one replanning cycle is discarded before the next begins. Since consecutive cycles in LMAPF are often highly correlated, especially when $h{=}1$, it is beneficial to carry information across cycles rather than restart from scratch. This idea echoes incremental heuristic search, where prior search effort is reused across a sequence of related problems such as D* Lite (Koenig and Likhachev 2002), and also recent one-shot MAPF frameworks such as RT-LaCAM (Liang et al. 2025). However, unlike the one-shot setting, LMAPF frequently updates agents' goals online, making the previously constructed search tree difficult to exploit effectively. In receding horizon settings, this same intuition naturally appears as warm starting, i.e., initializing the current planning solution with the path computed in the previous cycle, as like standard implementation of MPC.

Anytime Planning. In LMAPF, each executed action is derived from a one-shot (potentially windowed) solution and

can thus benefit from *anytime* planning, where an initial feasible solution is refined as more computation time becomes available. In one-shot MAPF, such a strategy is common, as fast suboptimal solvers are well established (Li et al. 2022). Since a lifelong solution consists of a sequence of such one-shot solutions, improving each windowed plan can be expected to increase lifelong throughput (Jiang et al. 2024).

Our Approach. Our method follows a receding horizon scheme with h=1, replanning at every timestep while executing only the first step of the current plan. At each timestep, we warm start the local guidance by reusing the planning result at the previous timestep. This produces an initial, feasible, and high-quality windowed plan quickly and also allows further anytime refinement. To compute each one-shot plan over a horizon of length w_Π, we utilize LaCAM with local guidance, which we describe next.

C LaCAM and PIBT

A natural way to solve *one-shot* MAPF is to search over *joint configurations*, where each node represents the locations of all agents and each edge corresponds to a one-step synchronous transition. Namely, each search node N is associated with a joint configuration $N.\mathcal{Q} \in V^n$. However, a vanilla search in this space can suffer from a prohibitive branching factor, which can be as large as $O(5^n)$ in a grid world, since enumerating all joint successors requires considering combinations of individual agents' moves.

LaCAM (Okumura 2023b) is a configuration-based solver that addresses this difficulty by combining a high-level search over joint configurations with *lazy successor generation*. Given $N.\mathcal{Q}$, a fast configuration generator proposes a promising collision-free next configuration $\mathcal{Q}'$ (i.e., one move for each agent) on demand, yielding a successor node N' with $N'.\mathcal{Q} = \mathcal{Q}'$.

A standard choice of configuration generator in LaCAM is *PIBT* (Okumura et al. 2022). PIBT determines each agent's next move by ranking candidate vertices $v \in \mathsf{neigh}(\mathcal{Q}[i]) \cup \{\mathcal{Q}[i]\}$, including wait. The ranking is based on the lexicographic cost $\langle \mathsf{dist}(v, g_i), \epsilon \rangle$, with smaller costs preferred, and PIBT selects the first collision-free option in that order. Here, $\mathsf{dist}(v, g_i)$ is the shortest-path distance from v to the current goal g_i, while ϵ is a random tiebreak term.

D LG-LaCAM

In high-density environments, purely goal-directed local decisions based on $\mathsf{dist}(v, g_i)$ can concentrate many agents onto moves that appear individually favorable, thereby creating *congestion* even when each local choice is reasonable. This, in turn, increases the need for collision resolution, degrading solution quality. *LG-LaCAM* (Arita and Okumura 2026) addresses this issue by augmenting configuration generation with *local guidance*: for each planning step, it constructs agent-wise guidance paths Φ over a w_Φ-step window so as to mitigate anticipated local congestion. Intuitively, the objective is to find, for each agent, a short path fragment that minimizes collisions within the window without becoming overly conservative. More precisely, with a hyperparameter $\alpha \in \mathbb{R}_{\geq 0}$, a single-agent guidance path for agent $i \in A$ is constructed by solving

$$\begin{aligned} \arg\min_{\pi \in \Omega} \ \mathsf{cost}_i(\pi) = \langle \mathsf{dist}(\pi[w_\Phi], g_i), 0 \rangle \\ + \sum_{t=0}^{w_\Phi - 1} \langle 1 + \alpha \cdot \mathsf{Ind}[\chi > 0], \chi \rangle. \end{aligned} \tag{1}$$

where Ω denotes the set of paths of length w_Φ+1 starting from the current location $Q[i]$, and χ is the number of collisions of the transition $(\pi[t], \pi[t+1])$ with other paths currently stored in Φ. Here, Ind is an indicator function that returns one only if the condition is true, and zero otherwise. These guidance paths can be computed by space-time A*.

Algorithm 1 outlines how to construct the guidance paths Φ, which is initialized as empty at the initial timestep, whereas at later timesteps it is warm-started from the guidance of the parent node. Then, the guidance is refined sequentially for each agent, and this procedure is repeated m times to reduce bias induced by the agent update order. The resulting guidance is then injected into PIBT through the lexicographic cost $\langle \mathsf{Ind}[\Phi[i][1] \neq v], \mathsf{dist}(v, g_i), \epsilon \rangle$, which first prefers moves consistent with the guidance and then preserves the original goal-directed preference.

Algorithm 1: Guidance Construction

input: configuration $\mathcal{Q} \in V^n$, goals $\mathcal{G} = (g_1, g_2, \ldots, g_n) \in V^n$
output: $\Phi \in V^{n \times (w_\Phi + 1)}$ **params:** $w_\Phi, m \in \mathbb{N}_{>0}$

```
initialize Φ
for r = 1, 2, ..., m do                          ▷ guidance refinement
    for i ∈ A do
        update Φ[i] by solving Equation (1)
return Φ
```

E LaCAM*

LaCAM* (Okumura 2023a) is an anytime variant of LaCAM that improves an initial feasible plan over time and can converge to an optimal solution under cumulative transition-cost objectives. After obtaining an initial solution, it continues searching for lower-cost ones using branch-and-bound; it maintains the current best plan and prunes nodes that cannot improve it. Specifically, each node N is scored by $f(N) = g(N) + h(N.\mathcal{Q})$, where $g(N)$ is the accumulated transition cost to N and $h(N.\mathcal{Q})$ is a heuristic estimate of the remaining cost to reach the goal configuration.

3 Method

Our idea is to leverage local guidance from one-shot MAPF to LMAPF. This adaptation, however, is nontrivial because goals change upon completion, preventing direct reuse of one-shot plans. Meanwhile, because many agents retain the same goals across consecutive timesteps, much of the previous plan can be reused with a simple time shift, making such reuse crucial for performance. To address this, we propose *Lifelong LaCAM with Local Guidance (LLLG)*, a receding-horizon framework that replans over a finite window while *warm-starting local guidance* from the previous timestep, rather than directly warm-starting the current solution.

Structure. As discussed preliminarily, a natural starting point for extending local guidance to LMAPF is the naive approach: at every timestep, solve a one-shot MAPF instance from the current configuration using LG-LaCAM and execute only the first joint action. This choice is attractive because LG-LaCAM is a powerful one-shot solver in dense settings, but replanning the full-horizon problem from scratch at every timestep can be computationally expensive under real-time constraints.

To reduce this burden, we introduce *windowed planning*. Instead of solving the full-horizon planning at each timestep, we run LG-LaCAM with its high-level search truncated at depth w_Π. When the search reaches depth w_Π, we stop expanding the node and regard the backtracked sequence of configurations from the root to that node as a w_Π-step windowed plan. This reduces the cost of replanning while still allowing the solver to mitigate short-term congestion, which is important in lifelong settings where goals continue to change over time, making long-term planning uncertain. While windowed LaCAM looks similar to rolling out PIBT w_Φ steps, we argue that employing LaCAM search provides advanced search techniques, e.g., preventing local livelock, LaCAM*, and deadlock detection (Jain et al. 2026). We then embed this windowed solver into a *receding horizon* framework. At each timestep, the planner computes a windowed plan from the current configuration, executes only the first timestep, and then replans after one timestep is executed.

A key consideration is how to exploit information from the previous timestep, given the substantial overlap between consecutive replanning cycles, as such reuse can reduce per-step computation cost. As illustrated in Fig. 2, in the configuration generation of LG-LaCAM, LLLG uses warm-started guidance associated with each configuration. At the root configuration, we initialize the guidance paths Φ from the suffix of the previous timestep's solution $\Pi^{t-1}[2{:}w_\Phi]$. For successor nodes during high-level search, we initialize the guidance from one computed at the parent node. In this way, each planning cycle leverages the information from the previous cycle, instead of completely discarding the previous planning effort.

Finally, because the resulting windowed plan can be improved sometimes within the same per-step time budget, we can optionally incorporate *anytime refinement*. After obtaining an initial feasible w_Π-step plan, the planner continues improving it using either LaCAM* or LNS. We defer the details of these refinements to later. These components together yield LLLG, summarized in Alg. 2.

Algorithm 2: Lifelong LaCAM with Local Guidance

```
while not terminated do
    update goals G
    initialize Φ^t with previous solution Π^{t-1}[2:w_Φ]
    Π^t ← windowed LG-LaCAM for (Q^t, G)
    optionally refine Π^t by {LaCAM*, LNS}
    execute Π^t[1]
```

We now detail the key design choices of our method.

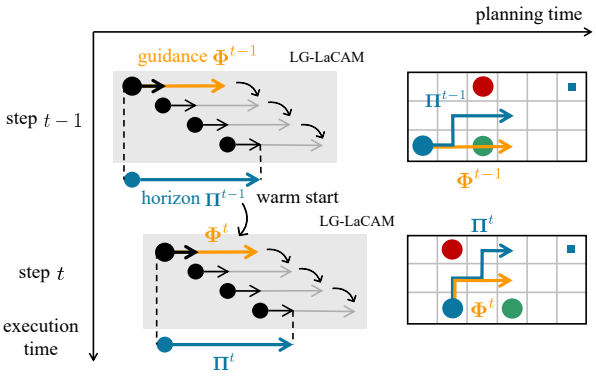

Figure 2: Concept of LLLG. The gray-shaded regions correspond to configuration generation by LaCAM. At each timestep, the initial guidance $\mathbf{\Phi}$ is warm-started from the previous step's solution $\mathbf{\Pi}$.

Leveraging LG in Receding Horizon. Under receding horizon execution, the guidance Φ is recomputed at every timestep, and performance depends on how we initialize each replanning cycle by reusing information from the previous step's planning result. Two warm-start schemes of LLLG are possible:

(i) Initializing Φ^t with Φ^{t-1} at the root configuration of the previous timestep, denoted $\text{LLLG}_{\mathbf{\Phi}}$.

(ii) Initializing the next guidance from the suffix of the previous finite-horizon solution Π^{t-1}, denoted $\text{LLLG}_{\mathbf{\Pi}}$.

For (ii), the guidance is initialized from the suffix of Π^{t-1}, using $\Pi^{t-1}[2{:}w_\Phi]$ when $w_\Phi \leq w_\Pi$; otherwise, we use $\Pi^{t-1}[2{:}w_\Pi]$ and pad the remaining steps up to length w_Φ with the terminal configuration. Later, we will evaluate these two warm-start schemes and show that (ii) is more effective in our lifelong receding horizon setting. Therefore, unless otherwise noted, LLLG refers to $\text{LLLG}_{\mathbf{\Pi}}$.

Refinement of local guidance. In Alg. 1, local guidance construction updates agent-wise paths sequentially; the resulting guidance can be biased by the update order within a single pass. Following LG-LaCAM (Arita and Okumura 2026), we perform m rounds of guidance refinement within each timestep. This reduces order-induced bias and improves solution quality. In the original full-horizon one-shot setting, LG-LaCAM uses $m{=}1$ because repeated refinement is costly when guidance must be refined at every step along the path from start to goal. By contrast, LLLG replans only over a finite window, making additional refinement practical. Accordingly, we typically use $m{=}2$ in LLLG.

Refinement of planning path. At each timestep, after LG-LaCAM returns an initial w_Π-step windowed plan, the planner can continue refining this plan until the next wall-clock timestep arrives. Because the windowed plan quality affects both the executed first action and the guidance carried over to the next timestep, improvements in one-shot windowed solution quality may serve as a promising indicator for higher lifelong throughput. We consider two refine-

ment options, each representing a design choice for allocating computation within a receding horizon planner.

- **LaCAM*** (Okumura 2023a) continues the horizon-limited LaCAM search after a first feasible plan is found. The planner keeps expanding its search within the same w_Π-step window and returns the best plan found under the time limit. In our implementation, the g-value is accumulated by a per-step transition cost $\mathsf{cost}(\mathcal{Q}^-, \mathcal{Q}) = \sum_{i \in A} \mathsf{Ind}[\mathcal{Q}[i] \neq \mathcal{Q}^-[i] \ \vee \ \mathcal{Q}^-[i] \neq g_i]$, and the heuristic is $h(\mathcal{Q}) = \sum_{i \in A} \mathsf{dist}(\mathcal{Q}[i], g_i)$.
- **Large Neighborhood Search (LNS)** (Li et al. 2021a) refines the current w_Π-step joint plan by repeatedly selecting a subset of agents, removing their paths, and replanning only for that subset while treating the remaining paths as fixed obstacles in space time (Jiang et al. 2024). Replanning is performed using a finite-horizon space-time A* search[1] over states (v, t) with evaluation function $f = g + h$. The path cost is accumulated using the per-step cost $\mathsf{cost}(v, t)$, defined as $\mathsf{cost}(v, t) = 1$ if the agent has not yet reached its goal and $\mathsf{cost}(v, t) = 0$ otherwise. The heuristic is defined as $h(v, t) = \mathsf{dist}(v, g_i)$ before the goal is reached and $h(v, t) = 0$ afterward.

Integration with Hindrance. We further refine PIBT's preference construction by integrating *hindrance* (Okumura and Nagai 2025), a lightweight one-step estimate of how an agent's move may hinder nearby agents at the next timestep, thereby mitigating short-term blocking. We combine hindrance with local guidance by ranking candidate moves $u \in \mathsf{neigh}(\mathcal{Q}[i]) \cup \{\mathcal{Q}[i]\}$ using a cost $\langle \mathsf{Ind}[\Phi[i][1] \neq u], \mathsf{dist}(u, g_i), \mathsf{hindrance}, \epsilon \rangle$. An empirical result of this effect is available in the appendix.

4 Evaluation

Experiments were conducted on a Mac Studio with M1 Ultra 64 GB of RAM. We use maps from the MAPF benchmark (Stern et al. 2019), and use the provided initial configurations as the start states. For evaluation, we report throughput and runtime, defined as the average number of completed tasks and computation time per executed timestep. Unless specified otherwise, each method is evaluated over 500 steps and 10 instances with a per-step planning timeout of 10 s. We use this setting to broaden the evaluation by examining behavior under longer per-step planning times. We compare representative leading approaches for LMAPF as follows:

- **RHCR** (Li et al. 2021b) is a leading LMAPF framework that interleaves execution and planning by solving windowed MAPF subproblems online. In our experiments, we use *Priority-Based Search (PBS)* (Ma et al. 2018) as the underlying MAPF solver, with the execution length set to h=5 and the planning horizon set to w_Π=10.
- **PIBT** (Okumura et al. 2022) (without hindrance) is a fundamental approach in the line of suboptimal methods for MAPF and its lifelong variant, LMAPF.

[1]Or SIPP (Phillips and Likhachev 2011).

- **(Lifelong) LaCAM** denotes our receding horizon extension of LaCAM (Okumura 2023b) without any guidance terms, using full-horizon planning.
- **Guided-PIBT** (Chen et al. 2024) augments PIBT with congestion-avoiding guidance to improve throughput in the lifelong setting.
- **WPPL** (Jiang et al. 2024) is a windowed anytime lifelong solver that combines PIBT with LNS, and it was the champion method in the first LMAPF competition (Chan et al. 2024). We evaluate two variants, using 1 s and 10 s of LNS refinement per timestep as specified in the official implementation, both without either Manual guidance or Guidance Graph Optimization.
- **Our proposed method, LLLG.** Unless noted otherwise, we set w_Φ=20, w_Π=10, and m=2.

For the baselines, the authors' provided codes are used.[2] We exclude learning-based approaches, such as (Jiang et al. 2025), from comparison since they constitute a different methodological category from the heuristic approaches considered in this study. For all LaCAM-based methods, we use the initial solution returned at each timestep (i.e., without anytime refinement) and enable the hindrance tiebreak.

A Qualitative Comparison

We first visualize the effect of guidance in the lifelong setting on two dense benchmark maps: *random-64-64-10* with 1,000 agents and *empty-48-48* with 1,000 agents. Figure 3 compares five LMAPF solvers: RHCR, PIBT, LaCAM, Guided-PIBT, and LLLG. Their qualitative differences are reflected both in the heatmaps and in the stop-count distributions shown at the bottom of the figure.

In *random-64-64-10*, RHCR appears to achieve near-optimal throughput, but at high planning cost. PIBT and LaCAM, lacking explicit congestion-mitigation guidance, concentrate agent interactions near the center of the map and produce pronounced local clusters with frequent stops, although LaCAM appears somewhat less congested, likely due to the hindrance tiebreaking. Guided-PIBT generates coarse detours to mitigate congestion, which disperses stop locations across the map. However, its guidance is time-independent and tends to discard fine-grained spatiotemporal trajectory information, making it overly conservative and leaving avoidable waiting in local bottlenecks. In contrast, LLLG produces informative spatiotemporal local guidance precisely where agents become locally dense, smoothing traffic in bottleneck regions and reducing repeated stopping. Notably, it achieves slightly higher throughput than RHCR while planning about ten times faster.

In *empty-48-48*, RHCR fails frequently in this dense setting. A key reason is that its PBS-based planning starts from an infeasible state, and a feasible solution may not be found within the allotted time limit. In such cases, RHCR falls

[2]RHCR: https://github.com/Jiaoyang-Li/RHCR; PIBT: https://github.com/HirokiNagai-39/pibt-tiebreaking; Guided-PIBT: https://github.com/nobodyczcz/guided-pibt; WPPL: https://github.com/DiligentPanda/MAPF-LRR2023

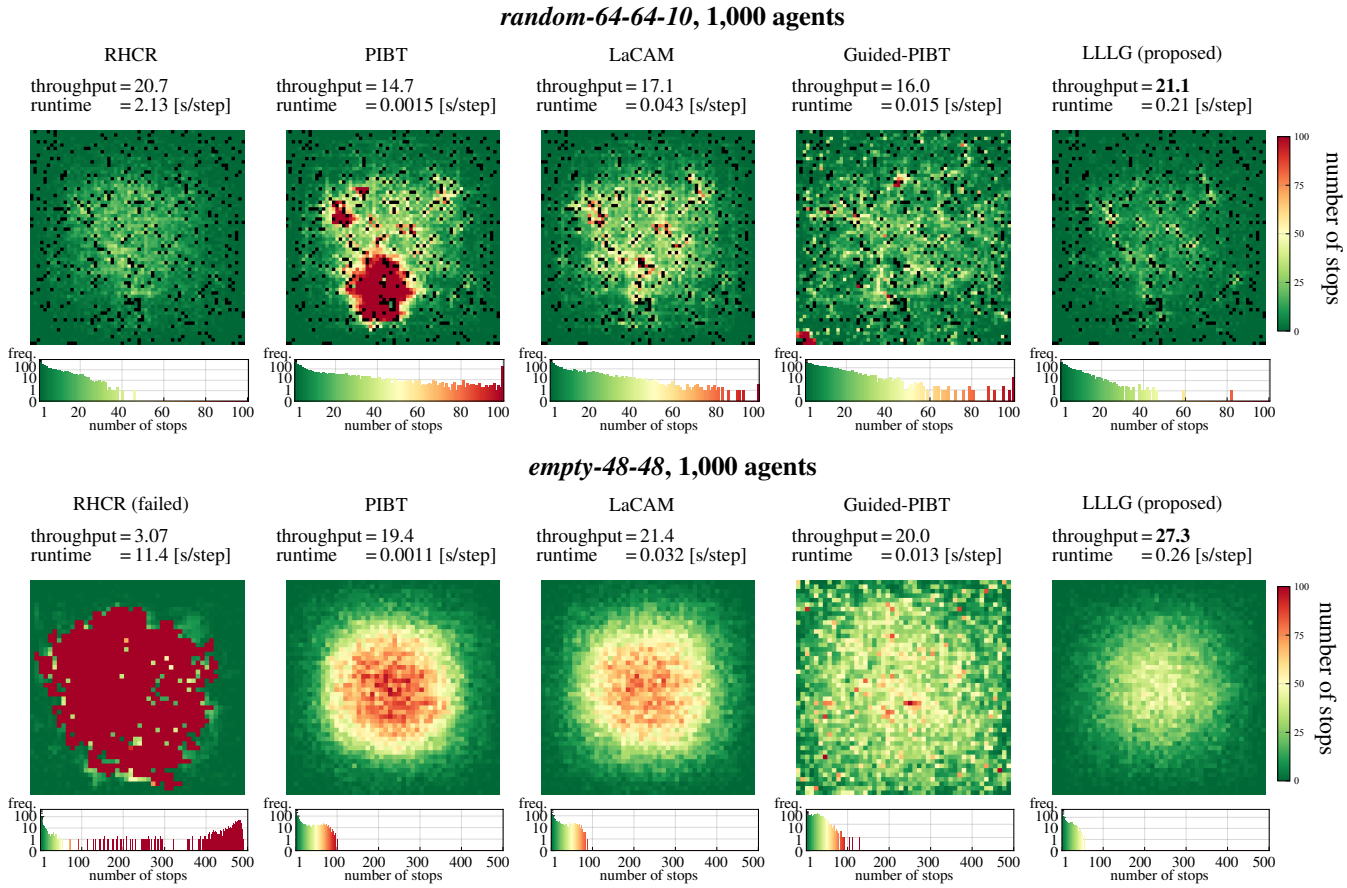

Figure 3: Heatmap of vertex stop counts, where congestion is implied by warm colors. The bottom of each heatmap shows the distribution among vertices of how many times each vertex is used as a stop by any agent.

back to LRA* (Silver 2005), which often results in excessive waiting and a marked drop in throughput. As more agents remain stationary, congestion further worsens, and once the number of non-moving agents exceeds half of the team, the episode is declared a failure and terminated. For the other methods, the qualitative trends become even clearer: PIBT and LaCAM show concentration and stopping, Guided-PIBT disperses traffic more broadly yet still leaves residual waiting, and LLLG most effectively smooths traffic and suppresses repeated stops.

B Quantitative Comparison

Figure 1 presents comparisons of lifelong methods across four representative scenarios. Among them, PIBT-based solvers (PIBT, LaCAM, WPPL, and Guided-PIBT) compute feasible actions instantly. RHCR, in contrast, is built on PBS and is designed to produce promising suboptimal plans in relatively sparse environments, but it incurs substantially higher planning time. Moreover, in *ht_chantry*, the solution quality can lag behind that of PIBT-based solvers. LLLG inherits the computational efficiency of PIBT-based planning while producing consistently better solutions than RHCR, demonstrating that local guidance can bridge the gap between speed and quality.

Figure 4 provides a comprehensive comparison across maps and agent scales. In sparse scenarios (e.g., 400 agents in *random-64-64-20*), LLLG achieves solution quality comparable to RHCR while requiring much less runtime. In dense scenarios (e.g., 1,000 agents in *random-64-64-20*), LLLG substantially outperforms prior methods in throughput, indicating that local guidance is particularly effective when congestion dominates system performance.

An exception is the large *warehouse-20-40-10-2-1*, where Guided-PIBT is particularly effective. We conjecture that this is because the guidance window cannot be chosen to cover the full length of long corridors. As a result, local finite-window guidance can struggle on maps containing one-cell-wide corridors, where agents must account in advance for oncoming traffic that may still lie beyond the current window. This observation is aligned with local guidance for one-shot MAPF (Arita and Okumura 2026).

Scalability. Figure 5 evaluates scalability in LMAPF on the warehouse map with up to 10,000 agents. Given its problem scale, LLLG uses lightweight parameters: w_Φ=8 and w_Π=5. In this setting, RHCR fails to keep up with 2,000 agents, as repeated branch-and-replan becomes a computational bottleneck. In contrast, LLLG remains practical at

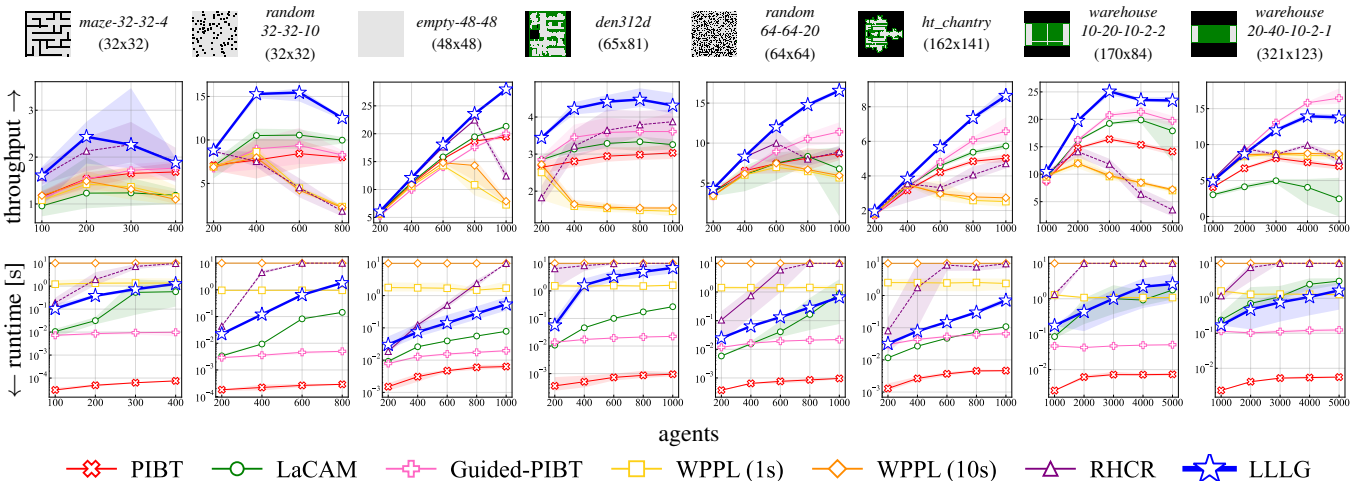

Figure 4: Performance evaluation of lifelong MAPF methods, assessed by throughput and runtime. For each map, the two subfigures report throughput and runtime over 500 steps for each agent scale. Each subfigure reports the average over the solved instances among 10 cases per setting, and the shaded regions indicate the min/max values. Results for failed cases (RHCR) are plotted with dashed lines, where each value is computed from the period before failure.

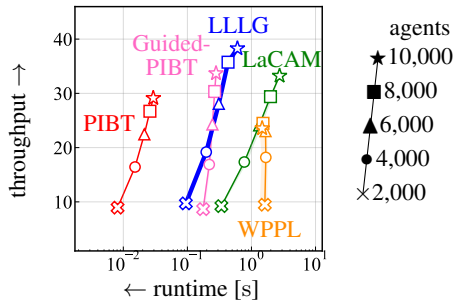

Figure 5: Scalability test on *warehouse-20-40-10-2-2*. LLLG uses a lightweight setting (w_Φ=8, w_Π=5).

an ultra-large scale. Across the tested loads, LLLG consistently produces actions within 1 s per step even for 10,000 agents, and achieves approximately a 30% throughput gain over PIBT. These results indicate that LLLG scales effectively to dense lifelong operation.

C Design Justification

To justify the design of our LLLG framework and better understand its empirical advantage, we isolate its main components and examine their contributions to performance. This section uses the shorter LMAPF simulation steps (100).

Warm Start with Previous-step Solution. Figure 6 evaluates the effect of warm-starting in LLLG. As a baseline, we include LLLG$_\emptyset$, which reconstructs guidance from scratch at every timestep. LLLG$_\Phi$ inherits the guidance paths from the previous guidance, whereas LLLG$_\Pi$ initializes guidance from the remaining suffix of the previous finite-horizon solution (i.e., equivalent to LLLG previously evaluated). As shown in Fig. 6, LLLG$_\Phi$ clearly improves throughput over LLLG$_\emptyset$ at essentially the same runtime, and LLLG$_\Pi$ yields a further improvement without additional computation cost. This suggests that warm-starting local guidance is beneficial in LMAPF and that reusing the previous-step solution is more effective than inheriting guidance alone. We speculate that guidance alone provides only a soft bias with respect to collisions, whereas the previous-step solution gives an explicit collision-free forecast over the near future and therefore a stronger initialization for subsequent search.

Horizon Length and Guidance Window. Figure 7 illustrates the interaction between the planning horizon length w_Π and the guidance window size w_Φ used by LLLG across multiple scenarios. For most agent configurations, increasing the planning horizon improves throughput up to a broad plateau, indicating that horizon planning is more effective when it is long enough to resolve near-term interactions and provide informative guidance. This appears consistent with the LLLG$_\Phi$ versus LLLG$_\Pi$ comparison in Fig. 6, where guidance constructed from a collision-resolving horizon plan tends to coordinate agents more effectively than guidance alone. At the same time, overly long horizons (e.g., w_Π=20) become less useful because future goal assignments are unknown in LMAPF; additional planning effort is spent on predictions that will soon be invalidated, making the added computation cost less cost-effective.

Guidance Refinement. We next investigate guidance refinement during the construction of the predicted horizon plan at each step. In LMAPF, each agent's guidance depends on the others, and traffic pattern changes as tasks are updated online. To cope with this, Alg. 2 refines guidance by repeating the guidance construction for m rounds within each timestep, reusing the previous step's guidance as a cache.

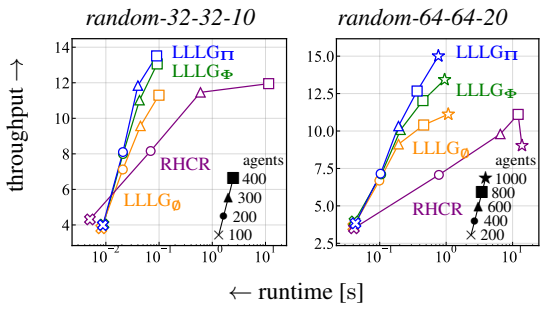

Figure 6: Comparison of warm-start schemes for LLLG.

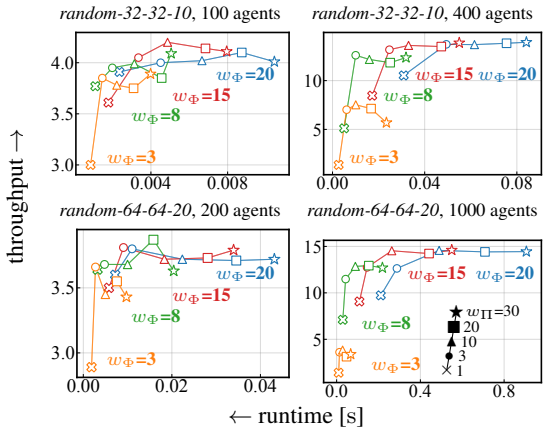

Figure 7: Sensitivity of throughput and runtime to the guidance window size w_Φ and the planning horizon w_Π.

Figure 8 shows that less frequent guidance updates are ineffective. In particular, m=0, which does not refine the inherited guidance and instead regenerates guidance only every window step, substantially degrades throughput. Updating and refining the guidance online at each step yields consistently higher throughput, indicating that "live" guidance based on up-to-date observations is essential for sustaining performance over long horizons. This aligns with observations in the one-shot MAPF case (Arita and Okumura 2026).

Anytime Refinement with Guidance. The preceding results show that LLLG is sufficiently fast for real-time lifelong operation. This leaves room to apply anytime planning within the per-step computation budget. LLLG admits two such refinement options, LaCAM* and LNS; here we focus on LaCAM*, and defer the LNS results to the appendix.

After finding an initial w_Π-step plan, LaCAM* continues the search within the same finite horizon and seeks to improve it. As shown in Fig. 9, the windowed version of lifelong LaCAM exhibits a small improvement with anytime refinement, whereas the same refinement degrades LLLG's performance. This suggests a discrepancy between the high-level cost function used by LaCAM* (i.e., f-value improvement on the w_Π-step window) and the lifelong performance metric of interest (throughput). A similar tendency is observed for LNS: its benefit is map-dependent, improving

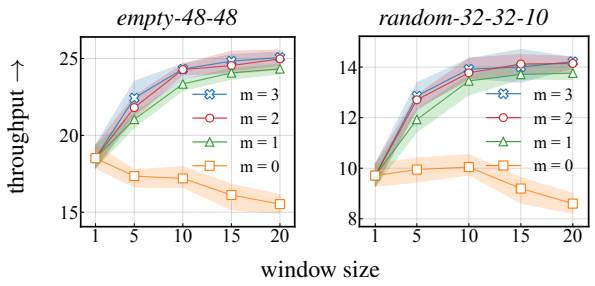

Figure 8: Effect of local-guidance refinement under different window sizes. m=0 does not refine the inherited guidance and updates it only when the guidance runs out. m=1 updates the inherited guidance once per step, and so force.

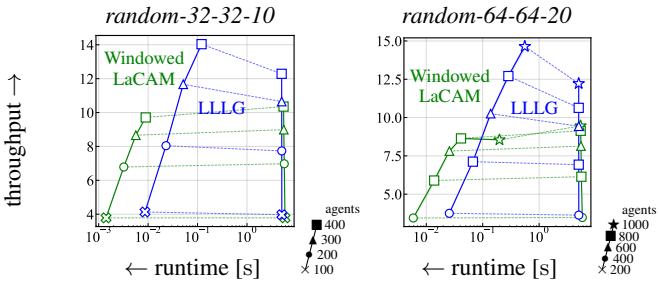

Figure 9: Effect of anytime improvement on the algorithms. The planning runtime budget is 5 s/step in both maps. Dashed lines connect each algorithm to its anytime variant.

performance on some smaller-scale maps but diminishing or even becoming negative in dense settings.

5 Discussion

This study shows that local guidance improves solution quality in lifelong MAPF, boosting throughput under real-time budgets. Integrated into a receding-horizon loop, our LLLG remains fast, scales to large maps, and mitigates dense bottlenecks. A key design choice is how to leverage the previous planning effort. Rather than warm-starting the finite-horizon solution itself, we warm-start the local guidance from the previous-step solution, whose near-term collision-free forecast is more informative than inherited guidance alone.

Empirically, increasing the planning horizon w_Π and the guidance window size w_Φ yields diminishing returns, and throughput improvement appears to saturate even with larger windows. A plausible explanation is that, in LMAPF, future goal assignments after task completion are unknown; thus, longer-horizon planning increasingly optimizes predictions that become irrelevant once agents reach their current goals and are reassigned. Addressing this limitation may require information about future tasks, such as future goal assignments or a predictable goal distribution. Exploring such extensions is beyond the scope of this paper, but it remains a key direction for the future. It is also important to explore cost evaluation for anytime refinement whose finite-horizon improvements better align with lifelong throughput.

Acknowledgments

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

References

Arita, T.; and Okumura, K. 2026. Local Guidance for Configuration-Based Multi-Agent Pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Bnaya, Z.; and Felner, A. 2014. Conflict-Oriented Windowed Hierarchical Cooperative A*. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*.

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)*.

Jain, R.; Okumura, K.; Amir, M.; and Prorok, A. 2026. Graph Attention-Guided Search for Dense Multi-Agent Pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Jiang, H.; Wang, Y.; Veerapaneni, R.; Duhan, T.; Sartoretti, G.; and Li, J. 2025. Deploying Ten Thousand Robots: Scalable Imitation Learning for Lifelong Multi-Agent Path Finding. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*.

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 Annual Symposium on Combinatorial Search (SoCS)*.

Kato, T.; Okumura, K.; Sasaki, Y.; and Yokomachi, N. 2025. Congestion Mitigation Path Planning for Large-Scale Multi-Agent Navigation in Dense Environments. *IEEE Robotics and Automation Letters (RA-L)*.

Koenig, S.; and Likhachev, M. 2002. D*lite. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P.; and Koenig, S. 2021a. Anytime multi-agent path finding via large neighborhood search. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

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. 2021b. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Liang, R.; Veerapaneni, R.; Harabor, D. D.; Li, J.; and Likhachev, M. 2025. Real-Time LaCAM for Real-Time MAPF. *Proceedings of Annual Symposium on Combinatorial Search (SoCS)*.

Ma, H.; Harabor, D. D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2018. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Ma, H.; Li, J.; Kumar, T. K. S.; 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)*.

Okumura, K. 2023a. Improving LaCAM for Scalable Eventually Optimal Multi-Agent Pathfinding. In *Proceedings of 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.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. *Artificial Intelligence (AIJ)*.

Okumura, K.; and Nagai, H. 2025. Lightweight and Effective Preference Construction in PIBT for Large-Scale Multi-Agent Pathfinding. In *Proceedings of Annual Symposium on Combinatorial Search (SoCS)*.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe interval path planning for dynamic environments. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*.

Shankar, A.; Okumura, K.; and Prorok, A. 2025. LF: Online Multi-Robot Path Planning Meets Optimal Trajectory Control. *ArXiv*, abs/2507.11464.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence (AIJ)*.

Silver, D. 2005. Cooperative Pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

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.; Saleem, M. S.; Li, J.; and Likhachev, M. 2025. Windowed MAPF with Completeness Guarantees. 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)*.

A Appendix

LNS is not reliably effective. Improving solution quality in lifelong MAPF via LNS is an important question. However, in our lifelong design, applying LNS at each step can

only refine the predicted finite-horizon plan, while the executed trajectory is committed online and cannot be retroactively revised. Therefore, it is not obvious that per-step LNS refinement should translate directly into better lifelong performance. Figure 10 suggests a nuanced conclusion. On smaller-scale maps, LNS tends to improve performance, indicating that refining the short-horizon plan can indeed reduce inefficiencies that accumulate over time. In contrast, this benefit is not consistent across all maps. In dense settings with many agents, the marginal gain from LNS diminishes and can even become negative, likely because additional search struggles to meaningfully restructure congestion-heavy interactions within a tight step budget.

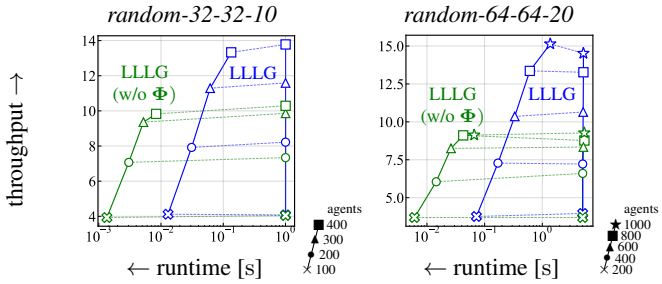

Figure 10: Effect of LNS improvement for different lifelong algorithms. The planning runtime budget including LNS is 1 s/step (left) and 5 s/step (right). Dashed lines connect each algorithm to its LNS-improved variant. LLLG(w/o Φ) corresponds to LaCAM.

Hindrance is a practical option. We also examine the effect of integrating hindrance. Especially in highly dense settings such as shown in Figure 11, combining hindrance with LLLG improves throughput with negligible computation cost. Thus, hindrance is a lightweight enhancement that is generally worth enabling in practice.

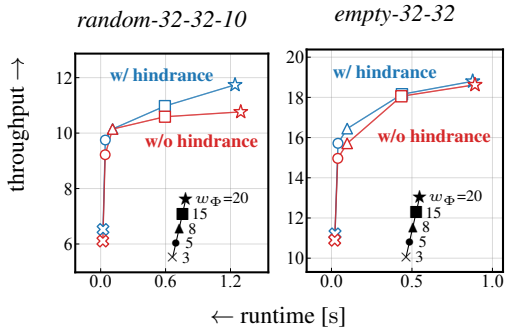

Figure 11: Sensitivity of throughput and runtime to the hindrance and the guidance window size w_Φ. Results is evaluated on *random-32-32-10* and *empty-32-32* with $1,000$ agents.