A Lightweight Traffic Map for Efficient Anytime LaCAM*

Bojie Shen , **Yue Zhang** , **Zhe Chen** and **Daniel Harabor**
Monash University, Faculty of Information Technology
{bojie.shen, yue.zhang, zhe.chen, daniel.harabor}@monash.edu

Abstract

Multi-Agent Path Finding (MAPF) seeks collision-free paths for teams of agents and has a wide range of practical applications. LaCAM*, an anytime configuration-based solver, currently represents the state-of-the-art. Recent work has explored using guidance paths to steer LaCAM* toward configurations that avoid traffic congestion, thereby improving solution quality. However, existing approaches rely on Frank–Wolfe–style optimisation to repeatedly invoke single-agent search before executing LaCAM*, which creates a large computational overhead in large-scale problems. The guide path is also static, which is only helpful for finding the first solution in LaCAM*. To overcome this problem, we propose a new approach that exploits LaCAM*'s ability to construct a dynamic, lightweight traffic map during LaCAM*'s search. Experiments show that our method achieves higher solution quality than state-of-the-art guidance-path approaches in two variants of MAPF problems.

1 Introduction

Multi-Agent Path Finding (MAPF) is a fundamental coordination problem in robotics and artificial intelligence. The problem asks computing collision-free paths for a team of agents navigating a shared environment [Stern *et al.*, 2019]. This problem lies at the heart of numerous large-scale industrial applications, most notably in automated warehouse [Wurman *et al.*, 2008], computer games [Silver, 2005], and railway scheduling [Li *et al.*, 2021a]. In these applications, hundreds or even thousands of moving agents must operate simultaneously to optimise a system objective. In such settings, optimal solvers typically struggle to scale as the number of agents grows [Shen *et al.*, 2023b; Lam *et al.*, 2022; Gange *et al.*, 2019; Shen *et al.*, 2023a; Li *et al.*, 2021b]. Consequently, real-world deployments often rely on suboptimal, unbounded approaches capable of generating feasible plans within tight, real-time computational limits [Okumura, 2024; Okumura *et al.*, 2022; Chen *et al.*, 2024; Jiang *et al.*, 2024].

To address these scalability challenges, rule-based algorithms like Priority Inheritance with Backtracking (PIBT) [Okumura *et al.*, 2022] and search-based frameworks like LaCAM/LaCAM* [Okumura, 2023b; Okumura, 2023a] have emerged. These methods typically rely on "myopic" heuristics to guide the search, such as individual shortest distances, which blindly steer agents into high-traffic areas, resulting in severe congestion and poor solution quality. To mitigate this, recent works introduce guidance mechanisms that steer agents away from potential bottlenecks. For instance, Space Utilization Optimization (SUO) [Han and Yu, 2022] precomputes spatially dispersed paths to diversify agent movements, helping to guide LaCAM/LaCAM* to avoid spatial congestion [Okumura, 2024]. Traffic Flow Optimisation [Chen *et al.*, 2024] approaches adapt ideas from traffic assignment problems, using iterative replanning to optimise congestion penalised guide paths (Frank-Wolfe-style optimisation) towards a User Equilibrium, where no agent can improve its travel time by switching routes. These paths then replace the default distance heuristic to guide PIBT. More recently, learning-based methods have been proposed to further optimise guidance policies or graphs to capture dynamic traffic patterns [Zhang *et al.*, 2024b; Zang *et al.*, 2025].

Despite their success in improving solution quality, these guidance-based approaches suffer from significant limitations regarding computational efficiency and adaptability. Methods like Traffic Flow Optimisation and SUO typically employ repeated computationally expensive pathfinding operations for every agent to reach a traffic equilibrium before the actual search begins. This introduces substantial setup overhead, often requiring dozens of seconds of pre-computation for large-scale instances, which undermines the real-time performance critical for online applications. Similarly, learning-based approaches often incur high training costs or require complex neural network updates that can be computationally intensive to execute online. Furthermore, these learnt policies are often highly specific to the training environment. Applying them to a new map topology or domain typically necessitates an extensive training process involving thousands of simulations.

To address these limitations, we propose a novel online mechanism called the Lightweight Traffic Map (LTM), designed to integrate seamlessly with the LaCAM framework. Instead of relying on an expensive, two-stage offline optimisation phase, our approach exploits LaCAM's inherent ability to rapidly sample configurations during the search itself. By recording the historical solution data from the anytime search process, we construct a dynamic weight map to repre-

sent the congestion in real-time. This traffic map is then used to bias subsequent search iterations and updated in real-time, effectively steering agents away from congested regions without incurring the computational overhead of repeated single-agent path replannings. We evaluate the proposed approach on a wide range of standard benchmark maps. Experimental results demonstrate that our method converges faster and produces higher-quality solutions than leading guidance-based approaches in the classic one-shot MAPF setting. Moreover, in the planning-and-execution MAPF setting, which emphasises anytime behaviour, our approach consistently outperforms the state-of-the-art solver PIE [Zhang *et al.*, 2024a] across all evaluated configurations.

2 Problem Definition

In the (classic) Multi-Agent Path Finding (MAPF) [Stern *et al.*, 2019] problem, we model the environment as a graph $G = (V, E)$, where vertices V represent locations and edges E represent feasible moves between locations. We consider a set of n agents $A = \{a_1, \dots, a_n\}$. Each agent a_i is given a start vertex $s_i \in V$ and a goal vertex $g_i \in V$. Time is discretised into steps $t = 0, 1, 2, \dots$. At each time step, an agent may either *wait* (remain at its current vertex) or *move* to an adjacent vertex. A (discrete-time) path for agent a_i is a sequence of vertices $\pi_i = \langle v_i(0), v_i(1), \dots, v_i(T_i) \rangle$ such that $v_i(0) = s_i$ and for all $t \in \{0, \dots, T_i - 1\}$, either $v_i(t+1) = v_i(t)$ (wait) or $(v_i(t), v_i(t+1)) \in E$ (move). The agent reaches its goal at time T_i with $v_i(T_i) = g_i$, and is assumed to remain at g_i thereafter. A solution is a set of paths $\Pi = \{\pi_1, \dots, \pi_n\}$ that is *collision-free*. In this paper, we consider (i) **Vertex conflicts**: no two agents occupy the same vertex at the same time, and (ii) **Edge conflicts** (swap conflicts): no two agents traverse the same edge in opposite directions at the same time.

Sum-of-loss objective. In this paper, we focus on finding a feasible joint plan Π that minimises *Sum-of-loss* $\mathrm{SoL}(\Pi)$. For a feasible joint plan $\Pi = \{\pi_i\}_{i=1}^{n}$, let $T = \max_i t_i$, SoL can be written as

$$\mathrm{SoL}(\Pi) = \sum_{t=0}^{T-1} |\{i \in \{1, \dots, n\} \mid \pi_i(t) \neq g_i\}| .$$

Intuitively, at each timestep we count how many agents are still not at their goals, and SoL sums this quantity over time. This objective therefore rewards solutions that bring agents to their goals earlier, capturing overall progress rather than only the last arrival time. Compared to makespan, which depends on the last-arriving agent and is sensitive to outliers, SoL measures the cumulative delay across all agents and better reflects overall system throughput, particularly in large-scale settings. In practice, improvements in SoL often also lead to improvements in makespan.

3 Related Works

3.1 PIBT and LaCAM*

Priority Inheritance and Backtracking (PIBT) [Okumura *et al.*, 2022] constructs successor configurations incrementally. Let $Q_t = (v_1^t, \dots, v_n^t)$ denote the current configuration. For each agent a_i, the candidate action set is $\mathcal{A}_i(Q_t) = \{v_i^t\} \cup \{u \mid (v_i^t, u) \in E\}$. Agents are processed according to a priority ordering. For agent a_i, actions $u \in \mathcal{A}_i(Q_t)$ are ranked by an evaluation function $f_i(u)$ and the highest-ranked feasible action is selected. If the selected action is blocked by a lower-priority agent, priority inheritance and backtracking are applied recursively to resolve conflicts. The effectiveness of PIBT strongly depends on the evaluation function f_i, which in original approaches is based solely on individual shortest-path distances i.e., $f_i(u) = \mathrm{dist}(u, g_i)$ in order to guide agents moving toward goal configuration.

Lazy Constraint Addition MAPF (LaCAM*) [Okumura, 2023b] searches over joint configurations and incrementally constructs a solution by generating one feasible successor at a time using a *configuration generator*. Each search node in LaCAM* corresponds to a configuration Q and maintains the accumulated transition cost $g(Q)$ together with a parent pointer, representing a partial joint plan. The search proceeds in a depth-first manner, incrementally constructing a sequence of configurations.

Given a configuration Q, the *configuration generator* using PIBT to produce a feasible successor configuration Q' by selecting one action per agent according to the evaluation function f_i while resolving conflicts. To ensure completeness despite generating only a single successor per expansion, LaCAM* associates each search node with a *constraint tree* that incrementally enumerates alternative successors. Each node in this tree encodes a set of constraints specifying partial assignments of agents to locations in the next configuration (i.e., "who is where"), and the generator must satisfy these constraints when producing successors. When the same successor is generated repeatedly, LaCAM* expands the constraint tree by introducing additional constraints, thereby guiding the generator toward different feasible configurations. By progressively expanding this constraint tree only when necessary, LaCAM* systematically enumerates all feasible successor configurations without explicitly branching over the exponential joint action space.

Let $\mathcal{Q}$ denote the set of all reachable configurations. LaCAM* is complete because the depth-first traversal, together with lazy constraint addition, ensures that every $Q \in \mathcal{Q}$ reachable from the initial configuration is eventually expanded, unless terminated by a time limit. To guarantee eventual convergence to an optimal solution, LaCAM* maintains and updates accumulated costs $g(Q)$ and parent pointers. Whenever a lower-cost path to an already discovered configuration Q is found, $g(Q)$ is updated and the improvement is propagated to descendant configurations via Dijkstra-style relaxation. In addition, LaCAM* incorporates a *swap* operator into the low-level generator to resolve local livelocks, further reducing search effort in dense instances.

3.2 Traffic Optimisation Using Guide Paths

PIBT and LaCAM/LaCAM* rely on evaluation functions based on individual shortest-path distances, which ignore inter-agent interactions and often lead to severe congestion. *Traffic Flow Optimisation* [Chen *et al.*, 2024] addresses this limitation by precomputing time-independent *guide paths* using a traffic-flow model that penalises vertex usage (vertex congestion) and opposing movements (contraflow con-

Algorithm 1: LaCAM* + LTM

Input: MAPF instance $\mathcal{I}$; initial root node N; per-iteration budget B; termination condition $\mathcal{T}$.
Output: Best solution found S_{best}.
Initialization: LTM $\leftarrow$ UniformCostMap; $S_{\text{best}} \leftarrow \emptyset$;

1 **while** $\mathcal{T}$ *is not satisfied* **do** // anytime loop
2 $(S, H) \leftarrow \text{LaCAM*}(\mathcal{I}, N, \text{LTM}, B)$; // H: PIBT history during this run;
3 **if** $S \neq \emptyset$ ***and*** $(S_{best} = \emptyset$ ***or*** $\text{SoL}(S) < \text{SoL}(S_{best}))$ **then**
4 $S_{\text{best}} \leftarrow S$;
5 LTM $\leftarrow$ UpdateLTM(LTM, H); // update traffic map from history;
6 $N \leftarrow$ SelectRestartNode(H); // select restart node;
7 **return** S_{best};

gestion). These guide paths are obtained through a Frank–Wolfe-style iterative optimisation procedure. During execution, agents prioritise moves that align with their assigned guide paths, significantly improving throughput.

Similarly, the enhanced LaCAM* [Okumura, 2024] integrates *Space Utilisation Optimisation* (SUO) [Han and Yu, 2022], which iteratively replans individual paths in a precomputation phase to minimise overlap on a global usage heatmap, producing spatially dispersed routes. These routes serve as a structural bias within the low-level PIBT generator, discouraging agents from entering crowded bottlenecks and reducing search effort in dense environments.

In both approaches, the core mechanism is to replace the original PIBT sorting function $f_i(u) = \text{dist}(u, g_i)$ with a traffic-aware evaluation function derived from precomputed guide paths, thereby biasing action selection toward less congested regions.

4 Our Method

While recent traffic optimisations enhance solution quality, they face two primary limitations. First, the iterative path optimisation introduces non-negligible initialization overhead, which delays the first action and negatively impacts anytime performance. Second, this optimisation is performed only once prior to the search. While the first solution is often significantly improved, subsequent search iterations tend to plateau, yielding only marginal or slow improvements.

To address these limitations, we propose a simple yet efficient approach called *Lightweight Traffic Map* (LTM). The main idea of our approaches leverages the ability of PIBT as a low-level successor generator to rapidly generate feasible configurations. That is, instead of performing iterative path optimisation, we directly collect historical data generated by PIBT to construct a lightweight traffic map. This traffic map is updated dynamically during search and is used to guide PIBT. We also modify the LaCAM* search to iteratively restart from selected configurations.

Algorithm 1 presents the overall framework of our modified LaCAM* with the proposed LTM. The algorithm maintains an anytime search loop that iteratively improves solution quality until a termination condition is met (line 1). At each iteration, a one-shot LaCAM* is called with a bounded node or time budget and guided by the current LTM (line 2). At initialisation, LTM is initialised with uniform traversal costs identical to those of the original map (i.e., unit cost), such that the first bounded LaCAM* execution is equivalent to the original LaCAM*. The best solution found so far is initialised to null, and the search starts from an initial restart node. During the first LaCAM* run, PIBT is used as the low-level configuration generator, and its execution history is recorded.

After LaCAM* returns, first, the resulting solution is then compared against the current best solution, and the best one is retained (line 4). Then, the algorithm directly updates LTM using the collected PIBT history (line 5). The details of maintaining and updating LTM are presented in the next section. Based on the updated LTM and the accumulated search history, a new restart node (or configuration) is selected to initialise the next iteration (line 6). This restart mechanism enables LaCAM* to iteratively explore alternative regions of the search space while continuously refining its traffic guidance. Finally, once the termination condition is satisfied, the algorithm returns the current best solution (line 7).

4.1 Updating the Lightweight Traffic Map

To ensure efficiency, we intentionally keep the design of the LTM simple. The LTM closely follows the structure of the original MAPF graph that defines traversability. However, instead of using uniform costs on undirected edges, the LTM assigns adaptive weights to directed edges to capture traffic information observed during the search.

Definition 1. ***(Lightweight Traffic Map)*** *Given a MAPF instance represented by a graph $G = (V, E)$, the Lightweight Traffic Map is defined as a directed weighted graph $G_{LTM} = (V_L, E_L, w_L)$, where $V_L = V$. For each undirected edge $(v_i, v_j) \in E$, the edge set E_L contains two directed edges (v_i, v_j) and (v_j, v_i). The weight function w_L assigns a non-negative cost to each directed edge, with weights normalized to lie within a bounded interval $w_{LB} \leq w_L(e) \leq w_{UP}$.*

The key idea of LTM is to leverage traffic information (actions and conflict-induced blockage) observed from PIBT executions to update edge weights dynamically. The edge weight reflects its observed traffic condition: edges with higher weights indicate heavier traffic and are therefore less preferred during traversal. By incorporating this information, LTM implicitly penalises congested transitions and encourages agents to explore alternative routes.

To prevent excessive penalization that could completely block certain transitions, all edge weights are normalized within fixed lower and upper bounds $[w_{LB}, w_{UP}]$. This bounded design ensures that no edge becomes prohibitively expensive and preserves the connectivity of the graph. In our experiments, we set $[w_{LB}, w_{UP}] = [0, 10]$, although this range can be easily tuned for different maps or domains. Note that, our LTM shares conceptual similarities with guidance graphs [Zang *et al.*, 2025], but differs in several important aspects. First, we do not explicitly model self-loop edges corresponding to wait actions. Instead, congestion at a vertex is implicitly propagated to its adjacent edges. Second, unlike guidance graphs that rely on expensive offline random instance sampling and training, often requiring hours or days to

construct, our LTM directly encodes traffic information gathered online from previous PIBT executions. As a result, LTM can be built and updated efficiently during the search. Next, we describe how the LTM is updated using historical data collected from PIBT.

Collecting Traffic Information from PIBT

Recall that LaCAM* employs the low-level configuration generator PIBT to construct the next configuration. Given a configuration Q_{from}, PIBT determines the next location for each agent a_i by sorting the candidate vertices $\text{neigh}(Q_{\text{from}}[a_i]) \cup \{Q_{\text{from}}[a_i]\}$ according to an evaluation function f. Initially, this function is defined as $f(v_i) = \text{dist}(v_i, g_i)$ where $\text{dist}(v_i, g_i)$ denotes the shortest-path distance from vertex v_i to the goal g_i of agent a_i. During each iteration of LaCAM*, we collect two types of traffic-related data from PIBT transitions between Q_{from} and Q_{to}.

- **Committed Actions.** For each agent a_i, we record the committed action $(Q_{\text{from}}[a_i], Q_{\text{to}}[a_i])$. This action corresponds to the actual movement chosen by PIBT and is used to increase the traffic cost of the corresponding edge. This is motivated by the observation that PIBT, when guided primarily by shortest-path distances, often directs multiple agents toward the same regions, leading to congestion. Penalizing frequently committed actions helps mitigate such traffic accumulation.
- **Blocked Actions.** For each agent a_i, PIBT may fail to select the highest-ranked action according to f due to conflicts with higher-priority agents, which also indicate potential congestion. We therefore record these as blocked actions of the form $(Q_{\text{from}}[a_i], v_i)$, where $v_i \in \text{neigh}(Q_{\text{from}}[a_i]) \cup \{Q_{\text{from}}[a_i]\}$ and $f(u) < f(Q_{\text{to}}[a_i])$.

For each recorded action (v_i, v_j) during a LaCAM* iteration, we increment the corresponding directed edge cost by 1. Since recorded actions may include wait actions (i.e., $v_i = v_j$), we propagate the increased cost to all outgoing edges from v_i, i.e., $\forall u \in \text{neigh}(v_i), w_L(v_i, u) = w_L(v_i, u) + 1$. This propagation allows vertex congestion to influence adjacent transitions without explicitly modeling self-loop edges. In addition, when an agent has already reached its goal, we ignore any subsequent wait actions and do not increase the corresponding traffic values. This design prevents goal locations from being artificially penalized due to agents waiting for others to finish, which is particularly important in instances with large makespan where prolonged waiting at goals is common.

To ensure correct normalization, we maintain a separate accumulator that records the raw traffic counts. After each update, the accumulated values are normalized into the predefined range $[w_{LB}, w_{UP}]$ to obtain the final LTM weights. During preliminary experiments, we explored more complex handcrafted update functions, but they did not outperform this simple additive scheme. While more sophisticated update rules may exist, designing optimal traffic update functions is beyond the scope of this paper.

4.2 Modifying LaCAM* with LTM

To efficiently use LTM in LaCAM*, we made two modifications to the original algorithm: (i) integrating LTM into the low-level successor generator PIBT used in each one-shot LaCAM; and (ii) modifying the anytime behaviour of LaCAM* to have more frequent restarts.

Integrating LTM into PIBT

Given an updated LTM, integrating it into PIBT is straightforward. Recall that PIBT relies on two key factors that influence the quality of the generated configurations: (i) the evaluation function $f(v_i)$, which determines the action ordering for each agent, and (ii) the priority ordering among agents, which determines conflict resolution when multiple agents compete for the same vertex. We modify PIBT to incorporate LTM in the following ways:

- **Evaluation function.** We replace the original evaluation function $f(v_i) = \text{dist}(v_i, g_i)$, computed on a uniform-cost graph, with the shortest-path distance computed on the weighted LTM. We further optimise the implementation to compute the distance on the dynamically updated weighted graph with a backward continuous A* search using the Manhattan heuristic to avoid precomputation overhead. Since the LTM is updated frequently, the backward A* search is restarted from scratch after each update. Compared to the original implementation, this modification introduces only negligible overhead.
- **Agent priority.** In the original one-shot LaCAM*, agent priorities at the root node are assigned based on the shortest distance $\text{dist}(s_i, g_i)$, and subsequent PIBT executions inherit priorities from parent nodes. We modify this priority assignment to use distances computed on the updated LTM.
- **Other LaCAM* Advances.** We also update distance estimates used in advanced PIBT optimisations, such as the push-and-swap operator used in LaCAM*, to rely on shortest-path distances computed on the updated LTM.

Modifying LaCAM* with frequent restarts

The original LaCAM* operates anytime in a depth-first search manner: it always generates one successor with PIBT and expands on that successor; once the goal configuration is met, it backtracks to the parent of the goal node to generate another successor. This iteration runs until all configurations are seen or pruned. In other words, it can be seen as iteratively running one-shot LaCAM*, growing the search tree and restarting at the last generated node, which is also shown in Algorithm 1. This process could have potential bottlenecks from (i) one LaCAM* iteration struggles to find the goal within a short time; (ii) the backtrack logic could lead to similar or worse solutions, such as agents taking local detours, which slows down the anytime performance. Therefore, we further modify the search to restart more frequently in two ways. First we introduce three early termination conditions to further restrict search effort and improve efficiency.

- The current iteration is terminated as soon as a goal configuration is found. Alternatively, if the search encounters an existed configuration and successfully obtain a better solution, the iteration is also terminated. This design preserves LaCAM's anytime behaviour by immediately returning improved solutions.

- We also terminate the iteration when the evaluation value of a search node, $f(n) = g(n)+h(n)$, exceeds the cost of the current best solution S_{best}. This prevents collecting traffic information from search regions that are unlikely to contribute to solution improvement.
- Each LaCAM* iteration is also constrained by a predefined node budgets. This condition is motivated by the observation that LaCAM* may generate a large number of low-level constraint tree nodes to resolve complex conflicts when PIBT fails to produce a feasible configuration, which can dominate the search time.

Next, we describe how restart nodes are selected and how PIBT is modified to incorporate guidance from the LTM. In principle, any node recorded in the current LaCAM* search tree can be selected as a restart node. However, selecting a restart node uniformly at random is unlikely to yield better solutions. A common and effective strategy is to restart from nodes along search branches that lead to a solution, as these nodes are more likely to guide the search toward improved solutions. The specific restart strategy can be adapted to different application scenarios; in next section, we present concrete strategies for both one-shot MAPF and planning-and-execution settings.

5 Putting it Together

In this section, we discuss how the proposed algorithm, LaCAM*+LTM, can be adapted to two important application settings: (i) one-shot MAPF, where the algorithm is given a fixed time budget to solve a MAPF instance in a single run and returns the best solution found within that budget; and (ii) planning-and-execution MAPF, where the solver is repeatedly given a short time budget and is required to immediately commit a number of actions for execution.

5.1 One-shot MAPF

Adapting LaCAM*+LTM to the one-shot (classic) MAPF setting is straightforward. To preserve LaCAM's superior performance in finding an initial solution, we do not impose any node budget in the first iteration; consequently, the first iteration proceeds identically to the original LaCAM*. From the second iteration onward, we limit each LaCAM* run by a node budget set to ten times the current makespan (i.e., the depth of the search tree). This bound prevents the search from becoming trapped in the exponential growth of the low-level constraint tree when resolving complex conflicts.

Regarding restart node selection, we observe that, given sufficient runtime, restarting from the root node almost always leads to better convergence and higher-quality solutions. However, for time-constrained applications, this strategy can be adapted by selecting restart nodes from the configuration that are closer to a goal configuration.

5.2 Planning and Execution MAPF

Unlike one-shot MAPF, the planning-and-execution setting models a more practical scenario in which robots cannot wait indefinitely to compute a complete plan and must begin executing actions while planning continues. We adopt the standard planning-and-execution model [Zhang *et al.*, 2024a], parameterized by the execution time E and the commitment horizon X. Each action requires E seconds to execute, and the agent must commit to executing X actions at a time. Consequently, execution opens a planning window of duration $E \times X$, during which the planner attempts to improve the solution from the current committed configuration toward the goal. At the initial step, a planning window of length $E \times X$ is available. If no solution is found within this window, the agent commits to X wait actions and continues planning in the next window. This process repeats until a solution is found and execution can proceed.

To adapt LaCAM*+LTM to this setting, we align the termination condition of each LaCAM*+LTM execution with the planning window of length $E \times X$. LaCAM*+LTM is repeatedly executed until all agents reach their goals. Each execution follows the same one-shot MAPF setting, except that the search is not restarted from scratch: the global search tree is preserved and reused across executions. At the beginning of each planning window, we select the restart node N_{cur} corresponding to the agents' current configuration and restrict LaCAM* to expand only the subtree rooted at N_{cur} using depth-first search. This design allows the search to focus on refining future actions while respecting already committed executions. Between successive executions of LaCAM*+LTM, we apply the following modifications:

- At each planning window, only the next X actions are required for execution. Therefore, instead of returning the full plan, we return the prefix of length X from the current configuration N_{cur}.
- After updating N_{cur}, we discard historical PIBT data collected prior to N_{cur}, remove the associated traffic increments, and re-normalize the LTM. This prevents outdated traffic information from biasing future planning.

Finally, if LaCAM*+LTM fails to find a solution within the initial planning window, the search is paused at its termination point. In the subsequent planning window, LaCAM*+LTM resumes from the previously paused search state rather than restarting from the root.

6 Experiments

We implement our proposed LaCAM*+LTM in C++. All experiments were performed on a Linux workstation running Ubuntu 24.04.2 LTS, equipped with an Intel Xeon W-2135 CPU (6 cores, 12 threads, 3.70 GHz) and 125 GB RAM. For reproducibility, the source code are available on repository [1]

6.1 One-shot MAPF

We evaluate one-shot MAPF on eight benchmark maps [Stern *et al.*, 2019]. Standard maps use type-width-height naming (e.g., random-64-64 for a 64×64 grid), whereas warehouse maps use a different scheme (e.g., warehouse-10-20-10-2-1/-2 with sizes 161×63 and 170×84). To assess scalability, we generate 25 random instances per map and vary the number of agents from small cases to 2000 agents. All algorithms have a runtime limit of 30 seconds.

[1] https://github.com/bshen95/lacam-ltm

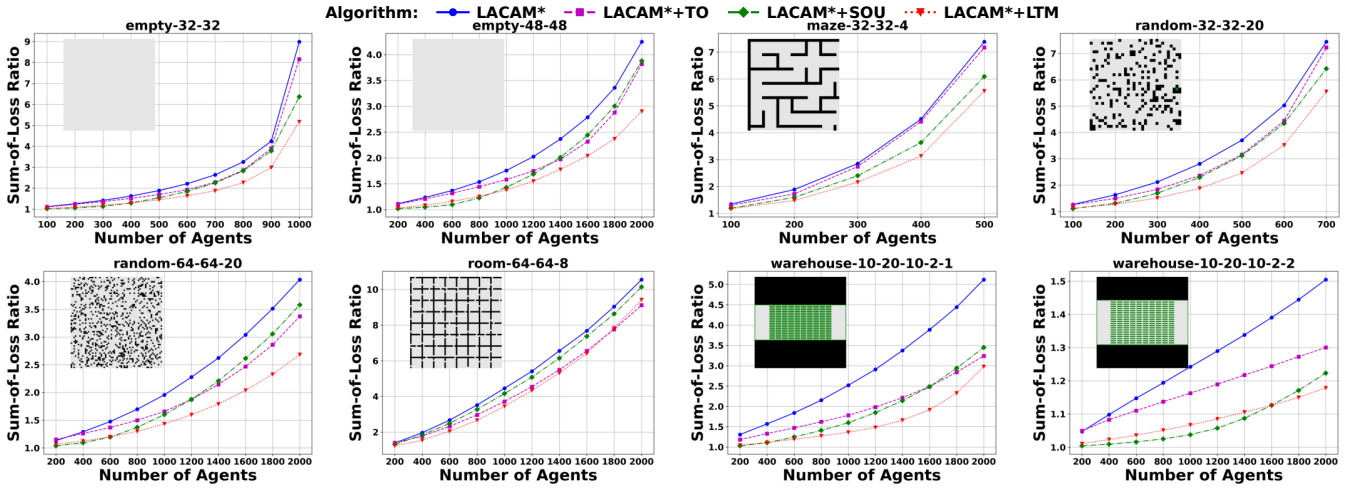

Figure 1: Comparison of solution quality on eight grid-based MAPF benchmarks under the one-shot setting. Each plot shows the average sum-of-loss ratio across different numbers of agents for LaCAM*, LaCAM*+TO, LaCAM*+SUO, and the proposed LaCAM*+LTM.

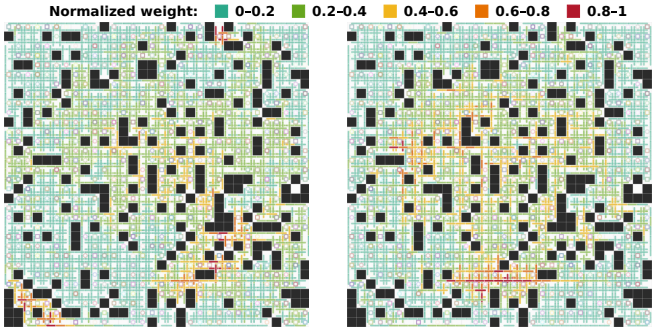

Figure 2: Visualization of the Lightweight Traffic Map (LTM) on the random-32-32-20 map with 400 agents. Circles denote start locations and squares denote goal locations. Edge colors indicate normalized traffic intensity. The LTM is shown after early iterations (left) and after 30 seconds of search (right).

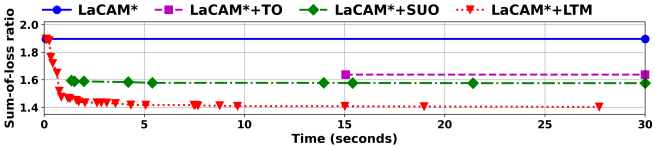

Figure 3: Coverage plot over a 30-second runtime for 1000 agents on the random-64-64-20.map, comparing LaCAM*, LaCAM*+TO, LaCAM*+SUO, and the proposed LaCAM*+LTM.

We compare LaCAM*+LTM against three representative baselines. (1) **LaCAM*** [Okumura, 2023b], the anytime variant of LaCAM. (2) **LaCAM*+TO**, which integrates traffic optimisation [Chen *et al.*, 2024] by repeatedly running Frank–Wolfe optimisation to compute guidance paths that minimise a handcrafted traffic objective. These guidance paths are then used to guide PIBT in LaCAM*. (3) **LaCAM*+SUO**, which applies space utilization optimization [Okumura, 2024] to minimize conflicts among guidance paths. For LaCAM*, TO and SUO, we use the authors' original implementation [Okumura, 2023a; Okumura, 2024; Chen *et al.*, 2024]. For our implementation, **LaCAM*+LTM** is built on top of the original LaCAM* codebase. To ensure fair comparison, traffic optimisation is run for 15 seconds to allow the Frank–Wolfe procedure to converge before executing LaCAM*. SUO is run until the guidance paths stabilise, following the authors' recommendations.

Solution Quality

Figure 1 compares solution quality under a 30-second time limit using the sum-of-loss ratio, computed relative to a trivial lower bound given by the sum of individual shortest-path costs. Overall, the proposed LaCAM*+LTM consistently achieves lower loss ratios than the baselines, particularly in dense scenarios with many agents, indicating better scalability as congestion increases. While LaCAM* performs competitively for small agent counts, its solution quality degrades rapidly as density grows due to its depth-first search strategy, which limits improvement as makespan increases. LaCAM*+TO and LaCAM*+SUO partially mitigate this issue by computing guidance paths at the start of the search; however, since these paths remain fixed, their effectiveness diminishes in large-scale instances. In contrast, LaCAM*+LTM combines frequent restarts with an adaptive Lightweight Traffic Map that continuously captures congestion patterns from previous iterations and updates heuristic guidance. This self-adaptive mechanism enables LaCAM*+LTM to efficiently generate higher-quality solutions over time without handcrafted optimization objectives.

Figure 3 further demonstrates its superior anytime performance, showing faster convergence and a larger number of returned solutions compared to competing methods. We do not directly compare against hybrid approaches such as LaCAM3 [Okumura, 2023b], as their performance largely depends on parallelization and LNS integration. When parallelism is disabled, LaCAM3 becomes equivalent to LaCAM*+SUO. In the next section, we further justify the advantages of our approach through a comparison with an

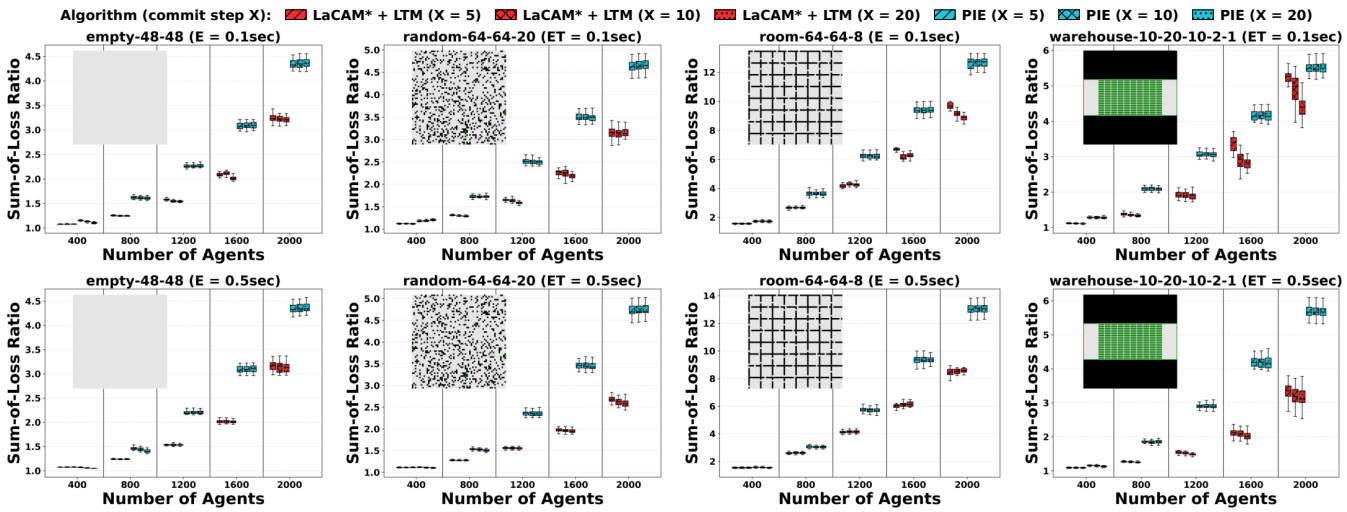

Figure 4: Planning-and-execution MAPF, each figure reports the sum of loss ratio for LaCAM*, LaCAM*+TO, LaCAM*+SUO, and the proposed LaCAM*+LTM. Results are shown for different execution times (E = 0.1 s and 0.5 s) and commitment steps of 5, 10, and 20 steps.

equivalent single-thread solver (PIE).

Inside the Lightweight Traffic Map

Figure 2 illustrates how the Lightweight Traffic Map (LTM) evolves during the search. In the early stages of LaCAM*+LTM, the LTM captures congestion induced by the first few solutions, with traffic initially spread across the map and only a small number of emerging high-traffic corridors. The resulting increases in traversal costs are then used to guide the low-level configuration generator, PIBT, producing improved heuristics in subsequent iterations and enabling LaCAM*+LTM to iteratively enhance solution quality while still returning solutions immediately. As the search progresses, congestion information from successive iterations accumulates: frequently congested area become increasingly penalized, leading to clearer and more concentrated congestion patterns along narrow passages and shared routes. This adaptive redistribution encourages agents to diversify their paths, alleviates congestion in heavily used regions, and ultimately results in improved solution quality over time.

6.2 Planning and Execution MAPF

We conduct experiments on the same benchmark maps used in the one-shot MAPF setting and compare against the state-of-the-art algorithm PIE [Zhang *et al.*, 2024a]. PIE can be viewed as a hybrid approach that combines LaCAM* [Okumura, 2023b] for quickly generating an initial feasible solution and LNS [Li *et al.*, 2022] for improving partial solutions. Specifically, PIE first uses LaCAM* to obtain a feasible plan for fast action commitment. During each subsequent planning window, it then applies LNS to refine the partial plan from the last committed configuration toward the goal, repeating this process until all agents reach their goals. To evaluate performance under different execution constraints, we vary the execution time E among 0.1 s, and 0.5 s, and the commitment horizon X among 5, 10, and 20 steps.

Figure 4 compares solution quality between the proposed LaCAM*+LTM and PIE under different execution constraints. Overall, LaCAM*+LTM consistently achieves lower sum-of-loss ratios as the number of agents increases, demonstrating stronger robustness in dense scenarios. When execution time is short (ET = 0.1 s), both methods are constrained by limited planning time; nevertheless, LaCAM*+LTM maintains better solution quality. As execution time increases to 0.5s, the performance gap widens, indicating that LaCAM*+LTM can more effectively exploit additional planning time. In contrast, PIE degrades more rapidly as agent density increases, mainly due to limitations of LNS. First, LNS improves solutions through destroy-and-repair operations that rely on space-time A*, which becomes increasingly expensive in high-density scenarios, reducing the number of improvement iterations. Moreover, LNS typically replans only a small subset of agents, which limits the impact of each iteration in large-scale instances. Second, in the planning-and-execution setting, LNS initially replans long paths from the start to the goal, making each iteration costly; as execution progresses, although replanned paths become shorter, the remaining room for improvement also diminishes. Consequently, local refinement alone is insufficient to resolve global congestion patterns. By contrast, LaCAM*+LTM continuously updates the traffic map and restarts from the current configuration, enabling adaptive global guidance that mitigates persistent congestion and scales better than PIE.

7 Conclusion

We proposed the Lightweight Traffic Map (LTM), an online and low-overhead mechanism that improves the anytime performance of LaCAM* by dynamically capturing congestion during search. By continuously updating traffic-aware guidance without offline optimization or handcrafted objectives, LaCAM*+LTM achieves faster convergence and higher solution quality than existing guidance-based methods. Experiments in both one-shot and planning-and-execution settings demonstrate its superior scalability and robustness in dense MAPF scenarios.

Acknowledgments

This research was supported by a gift from Amazon.

References

[Chen *et al.*, 2024] Zhe Chen, Daniel Harabor, Jiaoyang Li, and Peter J Stuckey. Traffic flow optimisation for lifelong multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20674–20682, 2024.

[Gange *et al.*, 2019] Graeme Gange, Daniel Harabor, and Peter J. Stuckey. Lazy CBS: implicit conflict-based search using lazy clause generation. In J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*, pages 155–162. AAAI Press, 2019.

[Han and Yu, 2022] Shuai D. Han and Jingjin Yu. Optimizing space utilization for more effective multi-robot path planning. pages 10709–10715, 2022.

[Jiang *et al.*, 2024] He Jiang, Yulun Zhang, Rishi Veerapaneni, and Jiaoyang Li. Scaling lifelong multi-agent path finding to more realistic settings: Research challenges and opportunities. In *Proceedings of the Symposium on Combinatorial Search*, pages 234–242, 2024.

[Lam *et al.*, 2022] Edward Lam, Pierre Le Bodic, Daniel Harabor, and Peter J. Stuckey. Branch-and-cut-and-price for multi-agent path finding. *Comput. Oper. Res.*, 144:105809, 2022.

[Li *et al.*, 2021a] Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. Scalable Rail Planning and Replanning: Winning the 2020 Flatland Challenge. In Susanne Biundo, Minh Do, Robert Goldman, Michael Katz, Qiang Yang, and Hankz Hankui Zhuo, editors, *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling*, pages 477–485. AAAI Press, 2021.

[Li *et al.*, 2021b] Jiaoyang Li, Wheeler Ruml, and Sven Koenig. EECBS: A bounded-suboptimal search for multi-agent path finding. pages 12353–12362, 2021.

[Li *et al.*, 2022] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. MAPF-LNS2: Fast Repairing for Multi-agent Path Finding via Large Neighborhood Search. In *Thirty-Sixth AAAI Conference on Artificial Intelligence*, pages 10256–10265. AAAI Press, 2022.

[Okumura *et al.*, 2022] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. *Artif. Intell.*, 310:103752, 2022.

[Okumura, 2023a] Keisuke Okumura. Improving lacam for scalable eventually optimal multi-agent pathfinding. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 243–251, 2023.

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

[Okumura, 2024] Keisuke Okumura. Engineering lacam*: Towards real-time, large-scale, and near-optimal multi-agent pathfinding. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2024.

[Shen *et al.*, 2023a] Bojie Shen, Zhe Che, Jiaoyang Li, Muhammad Aamir Cheema, Daniel Damir Harabor, and Peter J. Stuckey. Beyond Pairwise Reasoning in Multi-agent Path Finding. In Sven Koenig, Roni Stern, and Mauro Vallati, editors, *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, July 8-13, 2023, Prague, Czech Republic*, pages 384–392. AAAI Press, 2023.

[Shen *et al.*, 2023b] Bojie Shen, Zhe Chen, Muhammad Aamir Cheema, Daniel Damir Harabor, and Peter J. Stuckey. Tracking Progress in Multi-agent Path Finding. *CoRR*, abs/2305.08446, 2023.

[Silver, 2005] David Silver. Cooperative Pathfinding. In R. Michael Young and John E. Laird, editors, *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA*, pages 117–122. AAAI Press, 2005.

[Stern *et al.*, 2019] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, pages 151–158, 2019.

[Wurman *et al.*, 2008] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Mag.*, 29(1):9–20, 2008.

[Zang *et al.*, 2025] Hongzhi Zang, Yulun Zhang, He Jiang, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Jiaoyang Li. Online guidance graph optimization for lifelong multi-agent path finding. In Toby Walsh, Julie Shah, and Zico Kolter, editors, *Thirty-Ninth AAAI Conference on Artificial Intelligence*, pages 14726–14735. AAAI Press, 2025.

[Zhang *et al.*, 2024a] Yue Zhang, Zhe Chen, Daniel Harabor, Pierre Le Bodic, and Peter J Stuckey. Planning and execution in multi-agent path finding: Models and algorithms. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 707–715, 2024.

[Zhang *et al.*, 2024b] Yulun Zhang, He Jiang, Varun Bhatt, Stefanos Nikolaidis, and Jiaoyang Li. Guidance graph optimization for lifelong multi-agent path finding. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 311–320, 2024.