From Gridworlds to Warehouses: Adapting Lightweight One-shot Multi-Agent Pathfinding for AGVs

Hiroki Nagai[1,2], **Keisuke Okumura**[1]

[1]National Institute of Advanced Industrial Science and Technology (AIST), Japan
[2]Keio University, Japan
{nagai.hiroki39,okumura.k}@aist.go.jp

Abstract

Multi-agent pathfinding (MAPF) under one-shot planning is a core component of warehouse automation, yet classical formulations typically assume four-connected 2D grids with unit-time moves in four directions. To fill reality gaps while still being trackable with discrete combinatorial search, this work proposes a more practical counterpart tailored to differential-drive AGVs. We term this *multi-agent warehouse pathfinding (MAWPF)*, featured with four constraints: *(i)* agent actions are restricted to straight motion and in-place rotation; *(ii)* rotations require multi-step costs; *(iii)* acceleration and deceleration are considered, and; *(iv)* follower collisions are prohibited to prevent rear-end crashes. To solve MAWPF efficiently, we adapt representative suboptimal MAPF algorithms—PP, LNS2, PIBT, and LaCAM—and conduct comprehensive benchmarking. Our experiments reveal that PP and LNS2 struggle to solve instances with many agents, while PIBT-based approaches achieve preferable scalability with increased solution cost. We believe that these constitute an important step toward adapting classical gridworld MAPF to operational warehouse setups.

1 Introduction

Multi-agent pathfinding (MAPF) is a versatile abstraction for a variety of multi-agent planning problems, such as traffic intersection management [Dresner and Stone, 2008], railway scheduling [Li *et al.*, 2021a], autonomous parking [Okoso *et al.*, 2019], and conveyor routing [Kato and Okumura, 2026]. To enhance its engineering transferability beyond specific applications or infrastructures, most academic studies on MAPF adopt a gridworld scenario [Stern *et al.*, 2019], in which all agents use a planar graph as a workspace representation—typically a four-connected 2D grid—and take unit-length actions in unit time, in a fully synchronous manner. In this gridworld context, at each timestep, each agent occupies exactly one cell, which simplifies collision management. Such an assumption indeed reflects several characteristics of real-world *warehouse* setups [Wurman *et al.*, 2008], a representative MAPF application, where autonomous guided vehicles (AGVs) perform tasks while following predefined grid structures.

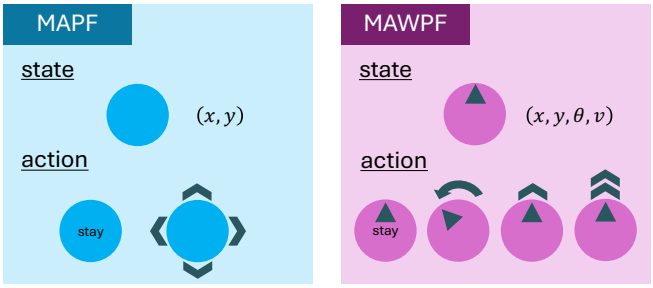

Figure 1: Classical MAPF and MAWPF.

This gridworld abstraction has enabled the research community to develop a wide variety of capable and foundational MAPF algorithms. In fact, recent developments of real-time, scalable MAPF algorithms—such as PIBT [Okumura *et al.*, 2022] and LaCAM [Okumura, 2023b]—are remarkable, capable of handling hundreds of agents in a second. Meanwhile, to deploy these algorithms in actual warehouse setups and maximize overall system performance, it is necessary to incorporate robot kinodynamic constraints tailored to AGVs. Such constraints include ***(i)* rotation constraints**, where an AGV cannot move in an arbitrary direction, ***(ii)* turning action costs**, where an AGV cannot complete a 90-degree rotation within a single time unit, ***(iii)* follower conflicts**, where an AGV must avoid entering a cell that was just vacated, and ***(iv)* acceleration and deceleration considerations**, where an AGV can traverse multiple cells within a single time unit.

Although several established post-processing methods exist to translate gridworld MAPF plans into AGV-compatible ones [Hönig *et al.*, 2016], these approaches inherently rely on suboptimal plans due to their use of over-simplified robot dynamics, making it difficult to achieve high system performance [Yan *et al.*, 2025]. Several studies have also explored extensions of gridworld MAPF—for example, incorporating rotation constraints [Zhang *et al.*, 2023; Chan *et al.*, 2024]—yet a unified formulation that integrates multiple practical AGV-specific considerations within a single framework is missing.

The aforementioned four AGV-related considerations are common in multi-AGV systems across different infrastruc-

tures. Given their economic impact and universality, we consider it valuable to define the corresponding problem that bridges academic research and industrial practice in MAPF.

To this end, we introduce the *multi-agent warehouse pathfinding (MAWPF)* problem as a gridworld counterpart of MAPF tailored to warehouse environments, and investigate how computationally lightweight MAPF algorithms can be adapted to this setting. MAWPF is carefully designed to capture the aforementioned four constraints stemming from AGV dynamics, while discrete, combinatorial search-based MAPF algorithms remain applicable with modest adaptation. In particular, we describe how representative MAPF solvers—Prioritized Planning (PP) [Erdmann and Lozano-Perez, 1987], LNS2 [Li *et al.*, 2022], PIBT, and LaCAM—are applied to MAWPF. We then systematically evaluate these solvers to delineate their current capabilities and limitations. We believe that this effort represents an important step toward upgrading conventional MAPF benchmarking with real-world grounding and operational relevance.

We begin the rest with the classical MAPF formulation and then define MAWPF, followed by reviews of MAPF algorithms and kinematics-aware variants. After that, we describe how MAPF algorithms are applied to MAWPF and report their empirical performance. The code and appendix are available from https://github.com/hirokiNagai-39/mawpf.

2 Classical MAPF

To facilitate understanding of MAWPF, we here describe the classical yet most commonly-used MAPF formulation [Stern *et al.*, 2019] to date. The system consists of a set of agents $A = \{1, 2, \ldots, n\}$ operating on a graph $G = (V, E)$ under a discrete, globally synchronized time model. At each timestep, an agent may either wait at its current vertex or move to an adjacent vertex. Feasibility is defined with respect to two collision constraints: *(i) vertex conflicts*, where two agents occupy the same vertex at the same time, and *(ii) edge conflicts*, where two agents traverse the same edge in opposite directions within a single timestep. Given distinct start and goal vertices $(s_i, g_i) \in V \times V$ for each agent $i \in A$, the one-shot MAPF problem asks for a finite sequence of actions for every agent that brings all agents from their starts to their respective goals without conflicts. Solution quality is evaluated using the *sum-of-costs (SoC)* (aka. flowtime), defined as the sum, over all agents, of the time until each agent first reaches its goal and remains there thereafter.

3 MAWPF

Classical MAPF provides a simple and intuitive abstraction for coordinating multiple agents, which has made it a widely used benchmark problem. However, it often departs from the operational realities of warehouse transportation, where differential-drive AGVs are prevalent and their motion is constrained by heading changes and speed control. To investigate how standard MAPF algorithms behave under such more realistic conditions, we first introduce a problem formulation that reflects these constraints. Specifically, we propose *multi-agent warehouse pathfinding (MAWPF)* as an MAPF tailored to the realities of automated warehouse distribution. The differences from classical MAPF are described below, and Fig. 1 provides an overview.

3.1 Problem Definition

Workspace. We consider a four-connected 2D grid map with *static* obstacles. The workspace is modeled as a graph $G = (V, E)$, where V is the set of obstacle-free cells and E connects pairs of four-neighbor cells; agents move along edges in E.

Configuration. In MAWPF, a *configuration* $\mathcal{Q}$ refers to the array of states of all agents at a given timestep. Unlike classical MAPF, each agent state includes not only its grid location but also its *direction* and *speed*. Formally,

$$\mathcal{Q}[i] = (x_i, y_i, \theta_i, v_i) \quad v_i \in \{0, 1, \ldots, V_{max}\}. \tag{1}$$

Here, speed v is the number of grid cells advanced in the next time step, and θ is the angle measured counterclockwise from the positive x-axis. In addition, the model is parameterized by $V_{\max}$, the maximum number of grid cells an agent can traverse in one timestep, and T_{rot}, the number of timesteps required to complete a $90\,\mathrm{deg}$ rotation.

Action. At each timestep, agent i updates its state by applying *(i) movement* and then *(ii) speed change*. In the movement phase, the agent selects exactly one of the following: *stay*, *forward*, or *rotation*. The *stay* action keeps the agent at the current cell and is available only when $v_i = 0$. The *forward* action moves the agent straight ahead by v_i cells in the direction θ_i; this action is available only when $\theta_i \in \{0, 90, 180, 270\}\ \mathrm{deg}$. The *rotation* action rotates the agent on the spot by $90/T_{\mathrm{rot}}\ \mathrm{deg}$ clockwise or counterclockwise, and it is available only when $v_i = 0$.

After the movement phase, the agent performs a speed change by selecting exactly one of *keep*, *acceleration*, or *deceleration*. The *keep* action leaves the speed unchanged. The *acceleration* action increases the speed by one, i.e., $v_i \leftarrow v_i + 1$, and is available only when $v_i < V_{\max}$ and $\theta_i \in \{0, 90, 180, 270\}\ \mathrm{deg}$. The *deceleration* action decreases the speed by one, i.e., $v_i \leftarrow v_i - 1$, and is available only when $v_i > 0$ and $\theta_i \in \{0, 90, 180, 270\}\ \mathrm{deg}$. Consequently, an agent cannot accelerate or stop abruptly.

Vertex Occupation. At timestep t, during the movement $\mathcal{Q}_t[i] \to \mathcal{Q}_{t+1}[i]$, agent i occupies the entire line segment range from geometric vertex (x_i^t, y_i^t) to (x_i^{t+1}, y_i^{t+1}): e.g., when an agent with $v_i = 2$ moves from $(0, 0)$ to $(2, 0)$, it occupies the three geometric vertices $(0, 0)$, $(1, 0)$, and $(2, 0)$.

Collision. At a given time step, a collision is considered to occur when a geometric vertex (x, y) is occupied by a pair of agents $i, j \in A$, $i \neq j$. This collision definition encompasses not only *vertex collisions* and *edge collisions*, but also *follower collisions*. A follower collision is a constraint preventing agents from entering the grid cells occupied by other agents in the previous timestep, in case those agents malfunctioned and did not move during that step.

MAWPF problem. Let $\mathcal{S}$ and $\mathcal{G}$ denote the start and goal configurations, respectively. The solution to MAWPF is a sequence of collision-free configurations $\Pi = (\mathcal{Q}_0, \ldots, \mathcal{Q}_k)$ that satisfy $\mathcal{Q}_0 = \mathcal{S}$ and $\mathcal{Q}_k = \mathcal{G}$.

Problem Difficulty. While it is possible to comprehend MAPF and MAWPF as pathfinding on a graph whose vertices are the configurations, a key difference between the MAWPF graph and the one used in classical MAPF is that the MAWPF graph is *directed*. This directionality arises from the assumption that agents cannot accelerate or decelerate abruptly: once an agent commits to a motion state, the set of feasible next states is constrained, and an agent is not guaranteed to be able to return to its previous vertex. As a result, the planner can easily enter a stuck (dead-end) situation where no feasible continuation exists without violating kinematic feasibility or collision constraints. With multiple agents, such irreversibility increases the likelihood of system-wide deadlocks, making planning failures more frequent. In these senses, solving MAWPF is inherently more difficult than solving classical MAPF on undirected graphs where agents can more readily backtrack and resolve local conflicts.

4 Related Work

Having defined MAWPF as a warehouse-grounded extension of classical MAPF, we next review prior work that informs our study from two perspectives: scalable MAPF solvers developed under the classical formulation, and MAPF variants that explicitly incorporate kinodynamic constraints.

MAPF Solvers. A naive approach to solve MAPF optimally is to employ joint-state search over configurations via A* [Hart *et al.*, 1968; Standley, 2010]. However, it is impractical because the branching factor grows exponentially with the number of agents, quickly making the search intractable. This motivates researchers to explore a different style of planning representation; for example, CBS [Sharon *et al.*, 2015], a celebrated MAPF algorithm, searches over constraint sets and replans agent-wise paths to resolve collisions.

CBS and its extension (e.g., [Li *et al.*, 2021b]) is more scalable than joint A*, yet it can still miss tight time budgets in scenarios with hundreds of agents or more. Therefore, a substantial line of research has focused on *unbounded suboptimal but scalable* MAPF algorithms that sacrifice some degree of optimality to achieve real-time performance on large instances. Representative lightweight methods include *Prioritized Planning (PP)* [Erdmann and Lozano-Perez, 1987], *LNS2* [Li *et al.*, 2022], *PIBT* [Okumura *et al.*, 2022], and *LaCAM* [Okumura, 2023b]; these will be reviewed in Sec. 5. In particular, the combination of LaCAM with PIBT has been reported as highly effective, enabling genuinely large-scale planning (on the order of thousands to even ten-thousands of agents) while maintaining practical runtime.

In this paper, we build on these scalable, lightweight MAPF techniques rather than relying on optimal algorithms since our target application is large-scale warehouse transportation. Moreover, the increased problem complexity arising from advanced kinodynamic constraints makes optimal planning less tractable, thereby reinforcing the practical relevance of suboptimal approaches.

Kinodynamic-considered MAPF. Classical MAPF typically abstracts away robot kinodynamics by assuming holonomic, discrete-time motion with instantaneous turns and instantaneous changes of velocity; however, for real AGVs in warehouses, such simplifications can misrepresent the true feasibility and execution cost.

Two approaches are commonly adopted to address this mismatch. The first category post-processes gridworld MAPF solutions to make them compatible with AGV dynamics. For example, [Hönig *et al.*, 2016] employs a simple temporal network to translate kinematics-agnostic plans into ones that respect non-holonomic motion with bounded translational and rotational velocities, while [Yan and Li, 2025] uses a continuous-time single-agent search to obtain such plans. This direction benefits from the scalability of grid-based solvers; however, the resulting solutions are often suboptimal, as high-level grid-based plans inherently introduce discrepancies between estimated and actual execution costs [Yan *et al.*, 2025].

Another line of work extends the classical MAPF formulation to account for robot dynamics explicitly and solves the resulting problem directly. This approach narrows the reality gap and enables the computation of plans with improved execution fidelity. For example, MAPF with Turn Actions (MAPF$_T$) [Zhang *et al.*, 2023] augments the action space by modeling turning as an explicit action, while MAPF with Kinematic Constraints (MAPFKC) [Ali and Yakovlev, 2021] incorporates speed and acceleration, and CL-MAPF [Wen *et al.*, 2022] focuses on Ackermann-steering, car-like robots, to name just a few. Our MAWPF falls within this category, but two features distinguish it in terms of the practicality and applicability of existing MAPF methods: *(i)* MAWPF adopts a fully discrete state space, which allows existing discrete search-based MAPF algorithms to be adapted directly (unlike MAPFKC); *(ii)* MAWPF jointly considers AGV-specific constraints, including motion, rotation, and collision characteristics (unlike MAPF$_T$ and CL-MAPF).

5 Algorithms

This section describes how we adapt existing lightweight MAPF algorithms to solve MAWPF under our differential-drive action model and warehouse-specific safety constraints.

5.1 Prioritized Planning (PP)

PP [Erdmann and Lozano-Perez, 1987] is a sub-optimal and incomplete solver that assigns priorities to agents and then plans collision-free paths in priority order. Similar to MAPF implementation, PP for MAWPF plans agents sequentially with a fixed priority ordering. For each agent, we run space–time A* [Silver, 2005] on the *MAWPF configuration graph*, whose vertices encode kinodynamic configurations and whose edges correspond to admissible MAWPF actions (e.g., forward motion and in-place rotation). Previously planned trajectories are treated as dynamic obstacles via a time-indexed reservation table: a transition at time t is rejected if it occupies any reserved cell at layer t. Since one MAWPF action may traverse multiple cells, we reserve and check all cells along the swept segment. The resulting path is then committed to the reservation table, including goal-hold after arrival. Overall, this algorithm constitutes the simplest baseline approach for solving MAWPF.

5.2 MAPF Large Neighborhood Search (LNS2)

LNS2 [Li *et al.*, 2022] is a sub-optimal and incomplete solver that first computes each agent's path without considering collisions, and then resolves collisions via large neighborhood search. LNS2 for MAWPF is a destroy-and-repair local search that reduces collisions by replanning only a subset of agents. It starts with *soft* prioritized planning: *as in PP for MAWPF*, each agent is planned on the MAWPF state graph with a time-indexed reservation structure, but collisions are allowed with an extra penalty proportional to the number of reserved cells intersected by the swept motion segment. It then repeatedly selects a subset of agents and destroys and repairs their paths until collisions are eliminated or the time budget is exhausted.

5.3 PIBT

PIBT [Okumura *et al.*, 2022] maps a collision-free configuration $\mathcal{Q}^{\text{from}} \in V^{|A|}$ to another $\mathcal{Q}^{\text{to}}$; iterating this mapping yields an MAPF solver that is incomplete and sub-optimal. Agents are assigned *priorities* and reserve vertices for the next timestep in priority order according to their *preferences*. If a low-priority agent conflicts with a high-priority agent, it inherits the higher-priority agent's position, recursively calls PIBT to vacate the cell, and then reserves another vertex.

Multi-step PIBT. Under a differential-drive model with only *move forward* and *rotate in place*, an agent receiving priority inheritance cannot always vacate immediately: because it cannot move laterally, side interactions may require multiple primitive actions before the contested cell is cleared. As shown in Table 1, the original PIBT can hardly solve MAWPF. We therefore extend PIBT to a *multi-step* variant, allowing an inherited agent to execute a short action sequence (e.g., rotate then move forward) to vacate in a kinematically feasible way, enabling PIBT-style coordination for differential-drive MAPF. Along this line, Enhancing PIBT [Yukhnevich and Andreychuk, 2026] solves lifelong MAPF for differential two-wheeled AGVs by reserving short-horizon segments rather than a single vertex; we similarly develop a multi-step PIBT-based algorithm for one-shot MAWPF, detailed at the end of this section.

Definition of Stop Path. In original PIBT, if agent i's plan fails, it chooses *stay*; in MAWPF, emergency stops are impossible, so *stay* is not always selectable. We thus define the *stop path* (the counterpart of *stay*) as the shortest path from the current state to a stop. For $L = 3$, examples include: $(0,0,0,3) \to (3,0,0,2) \to (5,0,0,1) \to (6,0,0,0)$. When agent i's plan fails, it is assigned the *stop path*.

Priority Inheritance for MAWPF. While several inheritance rules are possible, using the *stop path* as the analogue of *stop* makes the following rule natural: as shown in Fig. 2, among agents not yet assigned a path, priority is inherited by the agent whose stop path conflicts with agent i.

MAWPF Adaptation. The blue parts in Alg. 1 indicate the modifications from the original PIBT. Starting from the current configuration $\mathcal{Q}^{\text{from}}$, the procedure constructs an L-length configuration sequence $\{\mathcal{Q}^{\text{to}}_0, \mathcal{Q}^{\text{to}}_1, \dots, \mathcal{Q}^{\text{to}}_{L-1}\}$ by assigning each agent $i \in A$ a feasible horizon path; agents with $\mathcal{Q}^{\text{to}}_{L-1}[i]{=}\perp$ are planned via the recursive procedure $\mathsf{PIBT}(i)$ (Lines 1–3). We enumerate all admissible L-step action sequences via Breadth-First Search on the MAWPF configuration graph. Inside $\mathsf{PIBT}(i)$, candidate horizon paths $Paths \leftarrow \mathsf{path}(\mathcal{Q}^{\text{from}}[i])$ are sorted by the distance from $Path[L-1]$ to g_i (Lines 8–9), and tested in that order: the algorithm checks whether reserving the entire segment causes any collision (Lines 10–12) and commits a feasible candidate by setting $\mathcal{Q}^{\text{to}}[i] \leftarrow Path$ (Line 13). It then applies segment-level *priority inheritance*: for each unplanned agent j, it checks whether its default $StopPath[j]$ would collide with current reservations (Procedure inherit, Lines 4–6; condition in Lines 15); if so, it calls $\mathsf{PIBT}(j)$, and if this fails it backtracks and rejects the current candidate (Lines 15–16). If no candidate yields consistent reservations, it assigns $\mathcal{Q}^{\text{to}}[i] \leftarrow StopPath[i]$ and returns INVALID (Lines 20–21).

	$\lvert A\rvert{=}5$	10	15	20
PIBT	0.76	0.20	0.04	0.00
Multi-step PIBT ($L{=}6$)	1.00	1.00	1.00	1.00

Table 1: Success rate comparison between original PIBT and multi-step PIBT for MAWPF on *random-64-64-20* with $V_{\max}{=}2$, $T_{\text{rot}}{=}2$, using the rolling horizon method.

	$\lvert A\rvert{=}50$	100	150	200
Macro-successor	0.00	0.00	0.00	0.00
Rolling horizon	1.00	1.00	1.00	1.00

Table 2: Effect of rolling horizon (*random-64-64-20*, $L{=}6$, $V_{\max}{=}2$, $T_{\text{rot}}{=}2$).

Algorithm Improvements. PIBT for MAWPF incorporates implementation-level refinements to improve scalability under kinodynamic constraints.

- **Division sort** (Line 9): Instead of fully sorting $Paths$, we repeatedly sort only the top-K candidates and, if none is adopted, proceed to the next top-K until a path is selected. In practice, substantially fewer than $|Paths|$ elements are often sorted, yielding a noticeable reduction in computation.
- **Pruning** (Line 8): Among length-L candidate paths with identical first and last configurations, we keep only the one with the smallest number of moves, removing redundant candidates that return to the same endpoint configuration and discouraging oscillatory behavior. By reducing $|Paths|$, pruning directly accelerates a single configuration-generation call.

5.4 LaCAM

LaCAM [Okumura, 2023b] is a search-based meta-algorithm for MAPF that rapidly finds feasible solutions on large, dense instances. It reduces branching by avoiding explicit enumeration of the joint action space and instead generating successors *lazily* using constraints. LaCAM is *complete*: it returns a solution if one exists and otherwise reports failure. To do so, LaCAM employs two level search. The *high level* searches

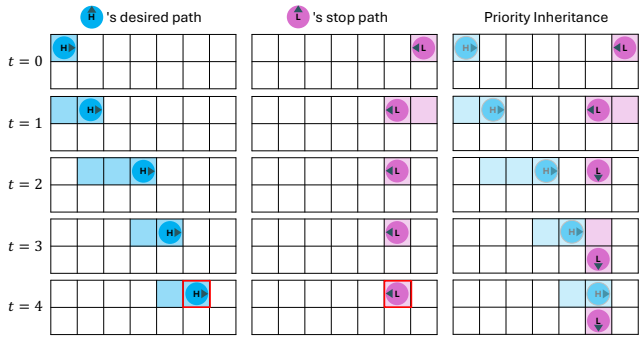

Figure 2: Priority Inheritance for MAWPF. Illustrates case $L = 4$. Colored vertices represent those occupied by agents at each time step. The path of the higher-priority agent (left) and the stopping path of the lower-priority agent (center) collide at $t = 4$, triggering priority inheritance. The lower-priority agent is then planned (right).

over *configurations*, constructing a sequence from $\mathcal{S}$ to $\mathcal{G}$. The *low level* grows a *constraint tree* whose nodes impose requirements such as "agent i must be at vertex v in the next configuration." For a selected node, LaCAM invokes a configuration generator such as PIBT to produce a valid next configuration; if generation fails, it switches to another node, by lazy successor enumeration.

MAWPF Adaptation. We retain the high-level search and replace the configuration generator with *PIBT for MAWPF* (Alg. 1). As this generator returns an L-step configuration array, two integration options arise. One treats the L-step array as a *macro-successor*, advancing the high level by L timesteps per expansion. The other uses only the first configuration and advances in a *rolling-horizon* manner, committing one timestep and re-invoking the generator at the next node.

Our evaluation shows that the macro-successor scheme fails to solve MAWPF instances (Table 2); thus, we adopt the rolling-horizon integration. This is likely due to the directed MAWPF configuration graph, where committing to multi-step expansions can quickly lead to dead-end configurations. Further details are provided in the appendix. Following this observation, we also adopt the rolling-horizon scheme when using *PIBT for MAWPF* alone.

6 Evaluation

This section evaluates the performance of the MAWPF algorithms: **PP**, **LNS2**, **PIBT**, and **LaCAM**.

6.1 Experimental Setup

The evaluation metrics are planning *success rate*, *runtime*, and *sum-of-cost*. Experiments were conducted on a laptop equipped with an M3 Pro Apple Silicon chip and 18 GB of RAM. All code was written in C++. Unless specified, $V_{\max}$ and T_{rot} were set to 2. For PIBT, we varied the lookahead horizon L from 5 to 7. The evaluation used 12 maps from the MAPF benchmark [Stern *et al.*, 2019]; see Fig. 3. The values reported were generated from 25 test cases, prepared for each map and each number of agents, with randomized start and goal states. In addition to positions, each start/goal includes a heading sampled uniformly from $\{0, 90, 180, 270\}$ deg, and the agent speed at both the start and the goal is set to 0.

Algorithm 1 PIBT for MAWPF

input: configuration $\mathcal{Q}^{\mathrm{from}}$, agents A, goals $(g_i)^{i \in A}$
output: configurations $\{ \mathcal{Q}^{\mathrm{to}}_0, \mathcal{Q}^{\mathrm{to}}_1, ..., \mathcal{Q}^{\mathrm{to}}_{L-1} \}$

```
for i ∈ A do
    if Q^to_{L-1}[i] = ⊥ then PIBT(i)
return { Q^to_0, Q^to_1, ..., Q^to_{L-1} }
procedure inherit(j)
    if StopPath[j] would cause a collision then return TRUE
    return FALSE
procedure PIBT(i)
    Paths ← path(Q^from[i])
    sort Paths in ascending order of
    dist(Path[L − 1], g_i) for Path ∈ Paths
    for Path ∈ Paths do
        collision ← FALSE
        if Path would cause a collision then continue
        Q^to[i] ← Path
        for j ∈ A do
            if j ≠ i ∧ Q^to_{L-1}[j] = ⊥ ∧ inherit(j) then
                if PIBT(j) = INVALID then
                    collision ← TRUE ; break
        if collision then continue
        return VALID
    Q^to[i] ← StopPath[i]
    return INVALID
```

6.2 One-shot MAWPF Results

Overview. The results, shown in Fig. 3, follow a trend similar to classical MAPF. Across most maps, LaCAM achieved the highest success rate and consistently solved instances with several hundred agents within a few seconds. This suggests that the LaCAM+PIBT pipeline preserves a certain degree of scalability and real-time performance in MAWPF despite richer kinodynamic constraints. In contrast, PIBT alone yielded lower success rates, indicating that LaCAM is critical for handling difficult configurations. PP and LNS2 showed lower success rates and generally longer runtimes than LaCAM. Although performance depended on each map, PP and LNS2 typically succeeded on instances with about 50–100 agents, while LaCAM solved a substantially larger fraction within the same time budget. Figure 4 summarizes the relationship between runtime and success rate across all scenarios and further highlights LaCAM's high success rate and responsiveness.

Anomaly on *warehouse-20-40-10-2-1*. A notable exception is *warehouse-20-40-10-2-1*, where LaCAM attains substantially lower success rates than PP and LNS2. This is consistent with PIBT's sensitivity to narrow passages: width-one corridors often induce severe contention and blocking, making greedy, inheritance-based decisions prone to stalling [Okumura, 2023a]. This interpretation is supported by *warehouse-20-40-10-2-2*: LaCAM solved all instances there but failed frequently on *warehouse-20-40-10-2-1*, with corridor width (two vs. one) as the key difference. Thus, the drop is a plausible consequence of applying PIBT to maps dominated by long, width-one corridors.

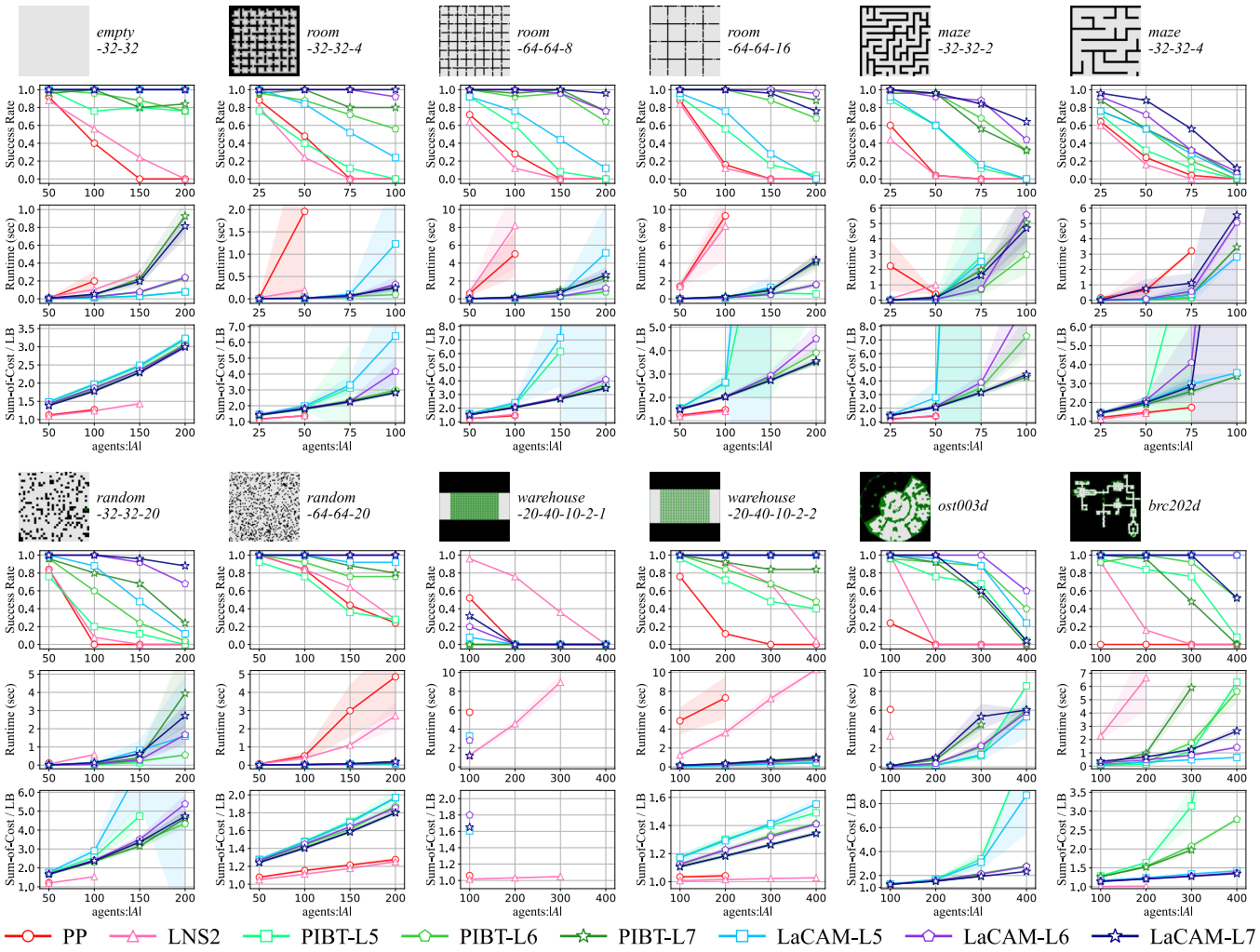

Figure 3: Results for one-shot MAPF. The success rate of planning within 10 s (top), average runtime (middle) and SoC normalized by lower bound ($LB = \sum_{i \in A} c_i^\star$, where $c_i^\star$ is the shortest-path cost of agent i on the MAWPF configuration graph, from $(x_i^s, y_i^s, \theta_i^s, v{=}0)$ to $(x_i^g, y_i^g, \theta_i^g, v{=}0)$, ignoring all other agents; 1.0 is minimum; bottom) for successful cases are shown. $V_{\max}{=}2$ and $T_{\mathrm{rot}}{=}2$ were used.

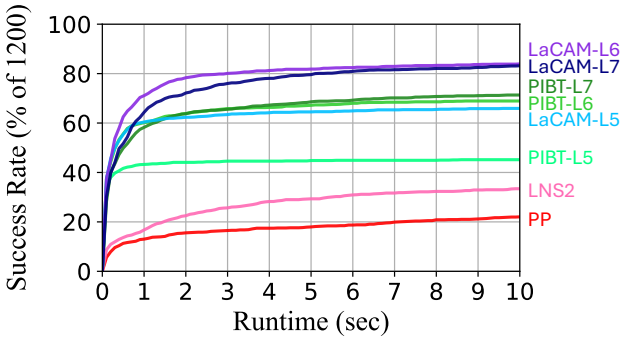

Figure 4: The number of solved instances among 1,200 instances on 12 four-connected grid maps shown in Fig. 3.

Solution Quality was broadly consistent with classical MAPF [Okumura, 2023b]. PP and LNS typically produced smaller sum-of-cost than LaCAM+PIBT. This is expected because PP/LNS repeatedly compute (near-)shortest single-agent paths via A*-like searches, whereas PIBT is a feasibility-driven heuristic based on greedy ordering, and does not explicitly optimize objectives such as sum-of-cost.

Effects of Path Length in PIBT/LaCAM. Larger L generally increased runtime because our PIBT adaptation employs a rolling-horizon interface: PIBT computes an L-step configuration array at each expansion, while LaCAM commits only the first configuration. On many maps, $L{=}\{6, 7\}$ achieved higher success rates than $L = 5$, suggesting that longer lookahead reduces dead-ends and failed successor generation. It can also improve solution quality by mitigating locally feasible but globally inefficient choices that later require detours or waiting, thereby reducing sum-of-cost.

6.3 Ablation Study

To assess *division sort* and *pruning* for improving PIBT, we conducted an ablation study (Fig. 6). Division sort reduced

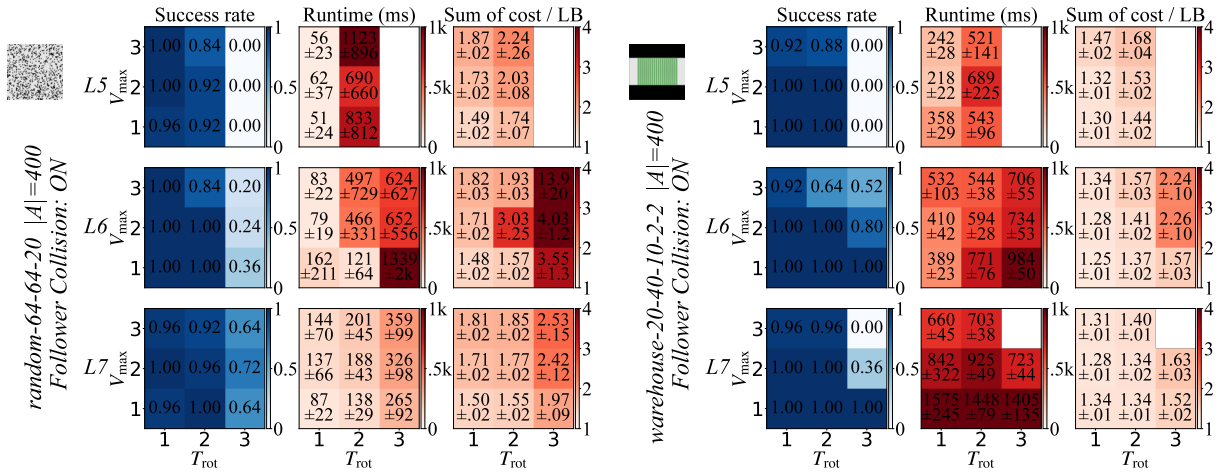

Figure 5: Results of experiments using LaCAM for MAWPF, varying the maximum speed and the number of steps required for a 90 deg turn. The success rate of planning within 10 s (left), average runtime (middle), and normalized SoC for successful cases are shown.

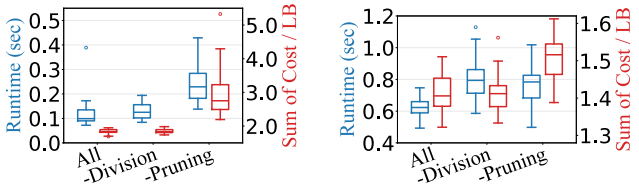

Figure 6: Results of the ablation study (LaCAM, $L = 6$, $V_{\max} = T_{\rm rot} = 2$) on *random-64-64-20* (left) and *warehouse-20-40-10-2-2* (right), both with $|A| = 400$. "All" denotes the full method, while "-Division" and "-Pruning" indicate ablated variants where each respective component is excluded. Planning was successful in all scenarios.

runtime, indicating that ranking candidate paths is a non-trivial bottleneck, since PIBT for MAWPF generates far more candidates than the original PIBT under the enriched transition model. Pruning further decreased runtime by removing candidates with identical endpoint configurations, and it improved sum-of-cost by filtering redundant detours such as oscillatory motions.

6.4 Experiments with Other Problem Settings

We further examined how kinodynamic parameters influence performance, focusing on LaCAM, which achieved the highest success rate among the compared methods. We fixed $L = 6$ and varied $V_{\max}$ and $T_{\rm rot}$, summarizing success rate, runtime, and sum-of-cost in Fig. 5.

Overall, larger $T_{\rm rot}$ degrades success rate and solution quality. Longer rotations consume more timesteps within the short horizon, leaving fewer steps for translation; thus, lookahead becomes less effective, and the planner behaves more like a smaller-L setting, making feasibility harder to sustain and typically increasing sum-of-cost.

The effect of $V_{\max}$ on runtime depends on map structure. On *random-64-64-20*, runtime increases with $V_{\max}$, consistent with a larger search space. On *warehouse-20-40-10-2-2*, higher $V_{\max}$ can reduce runtime, likely because long straight corridors better exploit acceleration and yield simpler, less congested progress, thereby reducing planning effort. We also conducted experiments allowing follower collisions; details are provided in the appendix.

7 Conclusion and Discussion

We define multi-agent warehouse pathfinding (MAWPF), extending classical MAPF with differential-drive AGV constraints—multi-step rotations, acceleration/deceleration, and conservative collision definition to enhance safety. Our empirical observation is that, among popular lightweight MAPF solvers, a rolling-horizon LaCAM+PIBT adaptation scales to hundreds of agents on benchmarks, achieving high success rates within a few seconds.

Why can't we scale to thousands of agents? In classical MAPF, LaCAM can solve benchmark instances with on the order of thousands of agents. In MAWPF, while LaCAM was the most scalable among the compared methods, its practical limit in our experiments was only a few hundred agents. We attribute this gap to two factors.

First, configuration generation is expensive. Each LaCAM expansion must produce a feasible next configuration, implemented in our MAWPF adaptation via a PIBT-style short-horizon procedure. Because MAWPF imposes richer kinodynamic constraints, even after pruning each agent can still have dozens to $\sim$100 candidate short paths, increasing the cost per configuration.

Second, as discussed in Sec. 3, the MAWPF configuration graph is directed since heading/speed constraints make transitions hard to reverse. This encourages dead-end states and can slow LaCAM's high-level search. These observations are for one-shot MAWPF; lifelong MAWPF may handle more agents via frequent replanning and incremental progress.

Outlook. Future work includes improving solution quality and robustness on challenging layouts (e.g., narrow corridors) via more cost-aware successor generation and additional refinement mechanisms.

Acknowledgments

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

References

[Ali and Yakovlev, 2021] Zain Alabedeen Ali and Konstantin Yakovlev. Prioritized sipp for multi-agent path finding with kinematic constraints. In *Interactive Collaborative Robotics (ICR 2021)*, Lecture Notes in Computer Science (LNAI), 2021.

[Chan *et al.*, 2024] Shao-Hung Chan, Zhe Chen, Teng Guo, Han Zhang, Yue Zhang, Daniel Harabor, Sven Koenig, Cathy Wu, and Jingjin Yu. The league of robot runners competition: Goals, designs, and implementation. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, 2024.

[Dresner and Stone, 2008] Kurt Dresner and Peter Stone. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research (JAIR)*, 2008.

[Erdmann and Lozano-Perez, 1987] Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. *Algorithmica*, 1987.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[Hönig *et al.*, 2016] Wolfgang Hönig, T. K. Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, 2016.

[Kato and Okumura, 2026] Takuro Kato and Keisuke Okumura. Conveyor parcel routing with order-contiguous arrivals. *arXiv preprint arXiv:2605.13035*, 2026.

[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 *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, 2021.

[Li *et al.*, 2021b] Jiaoyang Li, Wheeler Ruml, and Sven Koenig. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 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 *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

[Okoso *et al.*, 2019] Ayano Okoso, Keisuke Otaki, and Tomoki Nishi. Multi-agent path finding with priority for cooperative automated valet parking. In *ITSC*, 2019.

[Okumura *et al.*, 2022] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence (AIJ)*, 2022.

[Okumura, 2023a] Keisuke Okumura. Improving lacam for scalable eventually optimal multi-agent pathfinding. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*. International Joint Conferences on Artificial Intelligence Organization, 2023.

[Okumura, 2023b] Keisuke Okumura. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

[Sharon *et al.*, 2015] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence (AIJ)*, 2015.

[Silver, 2005] David Silver. Cooperative pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2005.

[Standley, 2010] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[Stern *et al.*, 2019] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of Annual Symposium on Combinatorial Search (SoCS)*, 2019.

[Wen *et al.*, 2022] Licheng Wen, Yong Liu, and Hongliang Li. Cl-mapf: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints. *Robotics and Autonomous Systems*, 2022.

[Wurman *et al.*, 2008] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 2008.

[Yan and Li, 2025] Jingtian Yan and Jiaoyang Li. Multi-agent motion planning for differential drive robots through stationary state search. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2025.

[Yan *et al.*, 2025] Jingtian Yan, Zhifei Li, William Kang, Stephen F Smith, and Jiaoyang Li. Analyzing planner design trade-offs for mapf under realistic simulation. *arXiv preprint arXiv:2512.09736*, 2025.

[Yukhnevich and Andreychuk, 2026] Egor Yukhnevich and Anton Andreychuk. Enhancing pibt via multi-action operations. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2026.

[Zhang *et al.*, 2023] Yue Zhang, Daniel Harabor, Pierre Le Bodic, and Peter J. Stuckey. Efficient multi agent path finding with turn actions. In *Proceedings of Annual Symposium on Combinatorial Search (SoCS)*, 2023.