

An Integrated Congestion-Aware Framework for Task Allocation and Path Planning in Multi-Agent Pickup and Delivery

Dariush Ebrahimi(✉), Meet Sable, Qing Dai, Yash Ahuja, and Michael Ajibola

Department of Computer Science and Physics, Wilfrid Laurier University, 75 University Ave W, Waterloo, ON N2L 3C5, Canada
debrahimi@wlu.ca, {sabl3770,daix1890,ahuj8260,ajib9610}@mylaurier.ca

Abstract. Large-scale Multi-Agent Pickup and Delivery (MAPD) systems operate in dynamic environments where agents must continuously allocate tasks and plan collision-free paths under strict real-time constraints. As system scale increases, congestion and agent interference become primary bottlenecks, significantly degrading throughput and increasing idle time. This paper presents a scalable lifelong MAPD framework that tightly integrates congestion-aware task allocation with priority-based path planning. Tasks are assigned using heuristic cost estimates that capture both spatial distance and congestion tendencies, reducing the likelihood of bottlenecks before planning occurs. For motion coordination, we extend Priority Inheritance with Backtracking (PIBT) with congestion-aware guidance and deviation refinement, enabling efficient local conflict resolution without global replanning. Extensive experiments in a realistic warehouse benchmark demonstrate that the proposed approach substantially improves task throughput, reduces agent idle time, and remains stable under extreme congestion. The framework scales effectively to scenarios with up to 3,000 agents while consistently meeting real-time execution limits, highlighting its suitability for large-scale robotic logistics systems.

Keywords: Multi-Agent Path Finding (MAPF) · Multi-Agent Pickup and Delivery (MAPD) · Task Allocation · Priority Inheritance with Backtracking (PIBT) · Congestion-Aware Planning · Large-Scale Multi-Robot Systems

1 Introduction

Multi-Agent Path Finding (MAPF) concerns the computation of collision-free paths for multiple agents operating in a shared environment while satisfying spatial and temporal constraints. MAPF serves as a foundational problem for numerous real-world applications, including automated warehouse robotics [1], intelligent transportation systems [2,3], and large-scale simulation and digital

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026, LNCS 16769, pp. 100–115, 2027.
https://doi.org/10.1007/978-3-032-30494-0_7

entertainment. Over the past decade, MAPF has received significant research attention, leading to the development of both optimal and suboptimal algorithms capable of handling increasingly complex scenarios.

The classical MAPF problem [4] assumes a static setting in which all agents are assigned fixed start and goal locations prior to planning. In this formulation, agents move synchronously on a graph and must avoid vertex and edge conflicts. While single-agent path planning can be solved efficiently using algorithms such as A*, extending these approaches to multi-agent systems leads to an exponential growth in the search space. To address this challenge, optimal solvers such as Conflict-Based Search (CBS) were introduced, decomposing the problem into a high-level conflict resolution process and low-level individual path searches [5,6]. CBS and its variants have demonstrated strong performance for moderately sized problems, but often struggle in highly congested or large-scale environments due to the increasing number of agent interactions.

Beyond classical MAPF, many real-world systems operate in dynamic and online settings where tasks appear over time, and agents must continuously replan. This gives rise to the Multi-Agent Pickup and Delivery (MAPD) problem, which extends MAPF by incorporating task allocation alongside path planning. Early online MAPF approaches such as COBRA [7], Token Passing (TP), and Token Passing with Time Swaps (TPTS) [8] emphasize efficiency and real-time execution, though they sacrifice completeness. Subsequent methods, including Multi-Label A* [9] and Task Conflict-Based Search (TCBS) [10], improve solution quality and completeness but remain limited in scalability. More recent hybrid approaches combining MAPF with combinatorial optimization techniques, such as TSP-based task sequencing [11], demonstrate promise but often rely on computationally expensive centralized planning.

In parallel, decoupled and priority-based methods have emerged as practical solutions for large-scale systems. Among them, Priority Inheritance with Backtracking (PIBT) offers an efficient and decentralized mechanism for resolving local conflicts by dynamically propagating priorities among agents [12]. PIBT has shown strong scalability properties and robustness in dense environments, making it particularly suitable for real-time MAPD scenarios. However, when applied without explicit consideration of task allocation and congestion patterns, its performance can degrade as agent density increases.

This paper builds upon congestion-aware PIBT by integrating it with a tailored heuristic-based task allocation strategy. The proposed framework explicitly accounts for congestion tendencies during task assignment, thereby reducing conflict frequency and improving overall system throughput. By jointly addressing task allocation and path planning within a unified framework, the approach achieves improved stability and efficiency in highly congested, large-scale MAPD environments.

The remainder of this paper is organized as follows. Section 2 formally defines the MAPD problem considered in this work. Section 3 presents the proposed algorithm and analyzes its computational properties. Section 4 evaluates the

approach through extensive experimental studies and benchmark comparisons. Finally, Sect. 5 concludes the paper and discusses directions for future research.

2 System Model and Problem Definition

We consider a MAPD problem in a warehouse-like environment modeled as an undirected graph $G = (V, E)$, where V denotes a finite set of discrete locations and E denotes feasible connections between locations. Time is discretized into uniform steps $t \in \mathbb{N}$. The system consists of a set of agents $A = \{a_1, \ldots, a_n\}$, with $|A| \leq |V|$, and a set of pickup-and-delivery tasks $X = \{x_1, \ldots, x_k\}$.

Each agent occupies at most one vertex at any time. A task represents a pickup-and-delivery operation that requires an agent to visit a pickup location and subsequently deliver to a designated goal location. The problem is to dynamically allocate tasks to agents and compute collision-free paths such that the system operates efficiently in a shared and potentially congested environment.

Let $\pi_i[t] \in V$ denote the location of agent a_i at discrete time step t, and let $p_i[t]$ denote the priority of agent a_i at time t. Priorities are used as a mechanism for resolving conflicts during path execution.

Each task $x_i \in X$ is defined by a pickup location $s_i \in V$ (optional for tasks that begin immediately) and a delivery location $g_i \in V$. The objective of the MAPD problem is to maximize the system *throughput*, defined as the number of tasks completed within a designated time horizon, while satisfying a set of feasibility constraints.

At any time, each agent may be assigned at most one task. Agent motion is constrained such that, at each time step, an agent may either move to an adjacent vertex or remain at its current location, i.e.,

$$\pi_i[t+1] \in \text{Neigh}(\pi_i[t]) \cup \{\pi_i[t]\},$$

where $\text{Neigh}(v)$ denotes the set of vertices adjacent to v.

Paths must be collision-free. Specifically, vertex conflicts are not allowed, meaning that no two agents may occupy the same vertex at the same time:

$$\forall i \neq j,\ \pi_i[t] \neq \pi_j[t].$$

In addition, swap conflicts are prohibited, where two agents exchange positions simultaneously:

$$\forall i \neq j,\ (\pi_i[t] = \pi_j[t+1]) \wedge (\pi_i[t+1] = \pi_j[t]).$$

Task execution is subject to precedence constraints: for any assigned task x_i, the corresponding agent must visit the pickup location s_i before reaching the delivery location g_i. Agent priorities $p_i[t]$ may evolve over time to support decentralized and dynamic conflict resolution during execution.

A solution to the MAPD problem consists of two tightly coupled components. First, a task allocation policy dynamically assigns available tasks to agents based

on the current system state and heuristic cost measures. Second, a path planning mechanism computes, for each agent a_i, a collision-free trajectory $\{\pi_i[t]\}_{t=0}^{T_i}$, where T_i denotes the time step at which the agent completes its assigned task by reaching the delivery location g_i.

The proposed framework jointly addresses task allocation and path planning in order to improve throughput and scalability in large-scale, congested MAPD environments.

3 Algorithm Design and Analysis

This section presents the proposed lifelong planning framework for large-scale MAPD, including conflict-aware path planning, the task allocation strategy, and an analysis of computational complexity. Algorithm 1 serves as the main control loop and determines the next action for each agent at every discrete time step.

3.1 Lifelong Planning Framework

The system operates in a lifelong planning setting, where agents continuously update their paths in response to newly assigned tasks, congestion, and conflicts. At each timestep, the planner determines the next vertex $\pi_i[t+1]$ for every agent a_i based on its current position, task status, and local traffic conditions.

Algorithm 1 begins with a preparation phase (lines 1–7). During this phase, heuristic tables storing shortest-path distances to goal vertices are initialized or updated when time permits (lines 2–4). Agents that are newly initialized or assigned a new task are marked for replanning (lines 5–7), ensuring that planning effort is focused only on agents that require updated guidance.

Guided path planning is performed in lines 9–14. For each marked agent, a new path is generated using the congestion-aware planner described in Algorithm 2. After a path is computed, congestion flow values are accumulated along the planned trajectory (line 13), representing anticipated occupancy by higher-priority agents. These flow values bias subsequent planning calls and reduce the likelihood of congestion buildup.

The refinement phase, shown in lines 16–21, addresses agents that deviate from their guided paths. Agents are sorted by descending deviation magnitude (line 16), where deviation is defined as the difference between the agent's current position and its guided path [12]. For each deviating agent, replanning is invoked again using Algorithm 2 (line 19), followed by flow updates (line 20). This refinement mechanism allows agents to gradually return to efficient trajectories without requiring global replanning.

Deviation handling is illustrated in Fig. 1. Each vertex is annotated with a pair $\langle d_{\text{dev}}, d_{\text{goal}} \rangle$, representing the deviation cost from the guided path and the remaining distance to the goal, respectively. During refinement, agents locally minimize this combined cost to rejoin their guided paths. Deviations may occur due to limited planning time, detours introduced by PIBT conflict resolution, or interference from higher-priority agents undergoing replanning.

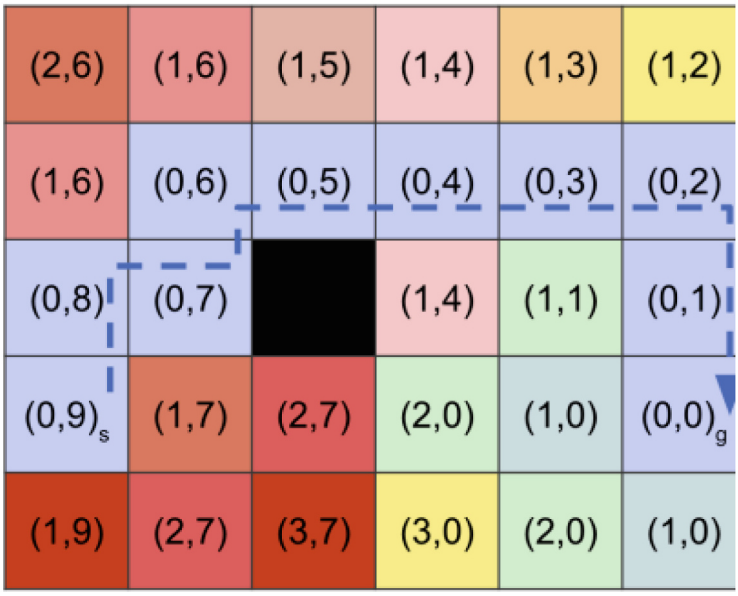


Fig. 1. Deviation handling during refinement. Vertices are labeled with ⟨deviation, distance-to-goal⟩ costs guiding agents back to their planned paths [12].

Conflict resolution is handled in lines 21–25 using Priority Inheritance with Backtracking (PIBT). Agents are processed in descending order of priority (line 23). When a conflict is detected, priorities are dynamically updated to ensure progress for blocked high-priority agents.

Figure 2 illustrates the priority inheritance mechanism. When a high-priority agent is blocked by a lower-priority agent, the lower-priority agent temporarily inherits the higher priority and moves first to vacate the blocking vertex. This priority propagation may extend through a chain of agents and enables decentralized conflict resolution without global replanning, as implemented in lines 21–25 of Algorithm 1.

In highly congested environments, priority inheritance alone may be insufficient. Figure 3 depicts such a scenario, where no admissible forward move exists even after priority propagation. In these cases, backtracking is employed, causing one or more agents to retreat along their recent trajectories. This controlled backward motion releases congestion and restores feasibility. Backtracking is invoked as part of the PIBT process when forward movement is impossible, ensuring deadlock-free execution under extreme congestion [13].

All computationally expensive operations in Algorithm 1 are strictly bounded by a predefined time budget. This guarantees real-time responsiveness and fairness, which is essential for competitive benchmark environments such as the League of Robot Runners [14,15]. The algorithm outputs the next action $\pi_i[t+1]$ for each agent at line 26.

Algorithm 1. Lifelong Path Planning Framework

```
1: for a_i ∈ A do
2:    if t_cur < t_max then
3:       ht ← InitOrUpdate(ht, a_i)
4:    end if
5:    if status(a_i) ∈ {New, Reassigned} then
6:       M ← M ∪ {a_i}
7:    end if
8: end for
9: for a_i ∈ M do
10:   if t_cur < t_max then
11:      π_i ← PLAN(a_i)
12:      C ← UpdateCongestion(C, π_i)
13:   end if
14: end for
15: for a_i ∈ A sorted by δ_i ↓ do
16:   if t_cur < t_max then
17:      π_i ← PLAN(a_i)
18:      C ← UpdateCongestion(C, π_i)
19:   end if
20: end for
21: for a_i ∈ A sorted by p_i ↓ do
22:   if DetectConflict(π_i) then
23:      P ← PIBT(P)
24:   end if
25: end for
26: return {π_i[t + 1] | a_i ∈ A}
```

3.2 Congestion-Aware Path Planning

Algorithm 2 details the congestion-aware path planner used in the guided planning and refinement stages of Algorithm 1. The planner is a modified A* search that incorporates congestion penalties in addition to standard distance heuristics.

Lines 1–2 initialize the search at the agent's current location $v_{s_i} = \pi_i[t]$, setting the path cost g to zero and the heuristic value h using the precomputed table ht. The start node is inserted into the open list (line 3). The main search loop (lines 4–13) expands nodes until the goal vertex v_{g_i} is reached.

For each expanded node v_{curr} (line 5), all neighboring vertices v_{next} are examined (line 6). The evaluation value combines traversal cost, heuristic distance, and congestion penalties derived from traffic flow. If v_{next} is reached for the first time or a lower-cost path is found, node information and parent pointers are updated (lines 7–10), and the node is reinserted into the open list.

After reaching the goal, the planner extracts the least-congested path from the search tree (line 13) and returns it as π_i (line 14).

Algorithm 2. Congestion-Aware Path Planning (PLAN)

```
1: initialize start node at v_{s_i} = π_i[t] with g = 0, h = ht(v_{s_i})
2: push start node into open list
3: while goal v_{g_i} not reached do
4:    v_curr ← pop node with minimum evaluation value
5:    for each v_next ∈ Neigh(v_curr) do
6:       compute evaluation value using distance, heuristic, and flow penalties
7:       if v_next not visited or lower-cost path found then
8:          update node cost and parent
9:          push v_next into open list
10:      end if
11:   end for
12: end while
13: extract least-congested path π_i
14: return π_i
```

Traffic flow is defined as the number of higher-priority agents expected to traverse a vertex in a given direction [12]. Opposing flows are penalized more heavily than same-direction flows, as they are more likely to cause deadlocks.

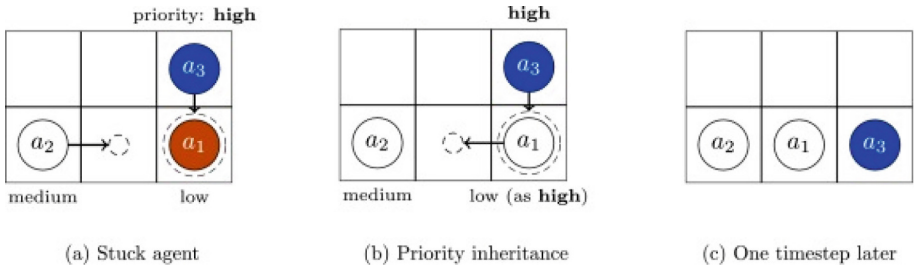
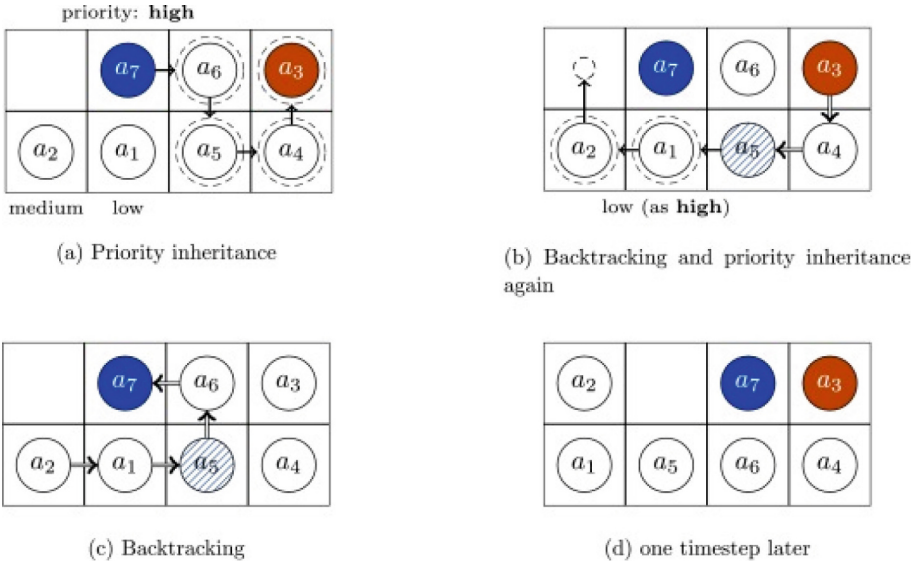


Fig. 2. Priority inheritance mechanism. A lower-priority agent temporarily inherits the priority of a blocked higher-priority agent to resolve conflicts locally [16].

3.3 Task Allocation

Task allocation is performed using the greedy heuristic-based procedure shown in Algorithm 3. At each planning cycle, the algorithm iterates over all vacant agents (line 1) and evaluates all unassigned tasks using a heuristic distance function (line 2). The task with the minimum heuristic value is selected and assigned to the agent (lines 3–5).

The heuristic computation uses BFS on the environment graph and caches results to avoid redundant searches. Once a task is assigned, the corresponding agent is immediately marked for replanning in the next iteration of Algorithm 1.



(a) Priority inheritance

(b) Backtracking and priority inheritance again

(c) Backtracking

(d) one timestep later

Fig. 3. Backtracking mechanism used when priority inheritance alone cannot resolve congestion. Agents retreat along recent trajectories to release space [16].

Algorithm 3. Heuristic-Based Task Allocation

```
for a_i ∈ A_vacant do
    x_j^* ← arg min_{x_j ∈ X_avail} ComputeHeuristic(a_i, x_j)
    if x_j^* ≠ NULL then
        task(a_i) ← x_j^*
        X_avail ← X_avail \ {x_j^*}
    end if
end for
```

3.4 Computational Complexity

Optimal MAPF is NP-complete, illustrated in Appendix A; Our proposed method prioritizes scalability and real-time feasibility via PIBT-based priority allocation. Each invocation of Algorithm repeated at a designated interval (e.g., 1 s). Algorithm 2 performs a modified A* search with worst-case complexity $O(|V| + |E| \log |V|)$. Since replanning is applied only to a subset of agents constrained by a time budget, the per-timestep complexity of the lifelong planner is approximately

$$O(|A| \cdot (|V| + |E| \log |V|)).$$

Task allocation (Algorithm 3) evaluates all unassigned tasks for each vacant agent. Each heuristic computation requires a BFS, yielding a worst-case complexity of

$$O(|A| \cdot |T| \cdot (|V| + |E|)).$$

All components run in polynomial time and are explicitly time-bounded, enabling the system to scale to thousands of agents while maintaining responsiveness in dynamic and congested MAPD environments.

4 Performance Evaluation

The MAPD problem has been widely studied, resulting in the development of numerous tools and benchmarks for systematic evaluation. To assess the performance of the proposed framework, we use the official simulation environment provided by the League of Robot Runners [15], a competitive benchmark designed to evaluate lifelong MAPD algorithms under realistic constraints. The simulator emphasizes robot dynamics, continuous task arrivals, online task allocation, congestion-aware planning, and strict real-time execution limits. This setting enables a robust evaluation of scalability and responsiveness in dynamic and highly congested warehouse-like environments.

4.1 Environment Setup

All experiments were conducted on a machine equipped with an Intel i7 9th-generation six-core CPU, 16 GB of RAM, and running Ubuntu Linux. The MAPF Start-Kit framework was used as the development and evaluation platform. This framework provides standardized interfaces for agent control, task allocation, and path planning, ensuring fair comparison across different MAPD approaches. Benchmark instances were generated using the MAPF Benchmark Archive, which supports flexible configuration of maps, agent counts, and task distributions.

Experiments were conducted on a large-scale sortation map of size 140×500, modeling a warehouse environment with narrow corridors and multiple pickup and drop-off locations. The map consists of blocked cells representing walls and obstacles, open cells that agents may traverse, designated pickup points, and delivery endpoints.

Tasks were specified using standard task files, where each task consists of a sequence of pickup and delivery locations that must be visited in order. Comment lines are ignored during parsing. For each experiment, three input files were generated: a map file defining the environment layout, a task file specifying pickup and delivery requests, and an agent file defining the initial positions of all agents.

To evaluate scalability, instances were generated with the number of agents ranging from 250 to 3,500 in increments of 250. Each simulation ran for 2,500 timesteps, with each timestep corresponding to one second of simulated time. At every timestep, the MAPD algorithm was allowed a maximum of one second of computation time to determine the next action for all agents.

Figure 4 shows a snapshot of the running simulation. Agents are represented as blue circles annotated with unique identifiers, and their facing directions are indicated by small dots. Open cells are shown in white, task destinations in

orange, and obstacles in black. New tasks arrive dynamically throughout the simulation, and vacant agents are assigned tasks online using the heuristic-based task allocation algorithm described in Sect. 3.

At each timestep, agents compute their next actions by integrating task assignment, congestion-aware path planning, deviation refinement, and priority-based conflict resolution using PIBT. This snapshot illustrates the dense interactions among agents navigating narrow corridors while executing tasks in real time.

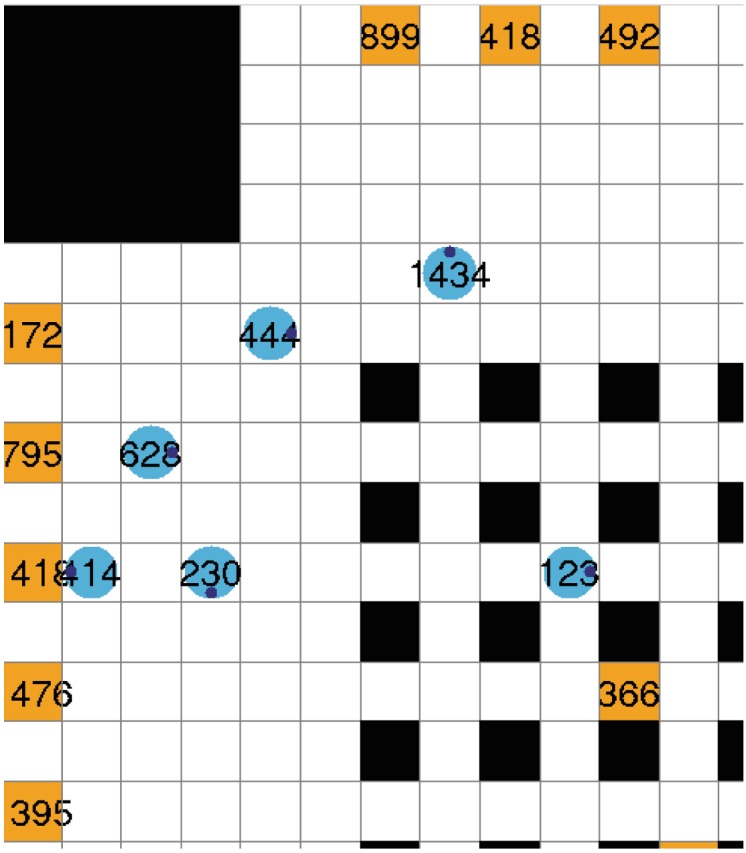


Fig. 4. Snapshot of the MAPD simulation on the sortation map. Agents navigate dynamically while executing pickup and delivery tasks under congestion and conflict constraints. Agents are represented as blue circles annotated with unique identifiers, and their facing directions are indicated by small dots. Open cells are shown in white, task destinations in orange, and obstacles in black. (Color figure online)

At the end of each simulation run, detailed output logs were generated. These logs include per-agent statistics such as average task completion time, number

of completed tasks, and idle time, as well as global metrics including the total number of tasks completed, average task waiting time, and computational performance. In particular, we measured the execution time of the MAPD algorithm at each timestep and the average computation time spent on task allocation and path planning.

The primary objectives of the evaluation were to assess scalability as the number of agents increases and to verify that the algorithm consistently operates within the one-second time constraint while maintaining high throughput.

4.2 Numerical Results

Figure 5 illustrates the relationship between the number of completed tasks and the number of agents over 2,500 timesteps. As expected, task throughput initially increases with the number of agents due to improved parallelism. However, the reference algorithm exhibits a noticeable performance drop at around 3,000 agents, while the proposed approach maintains a steady upward trend. This indicates improved robustness under high agent density.

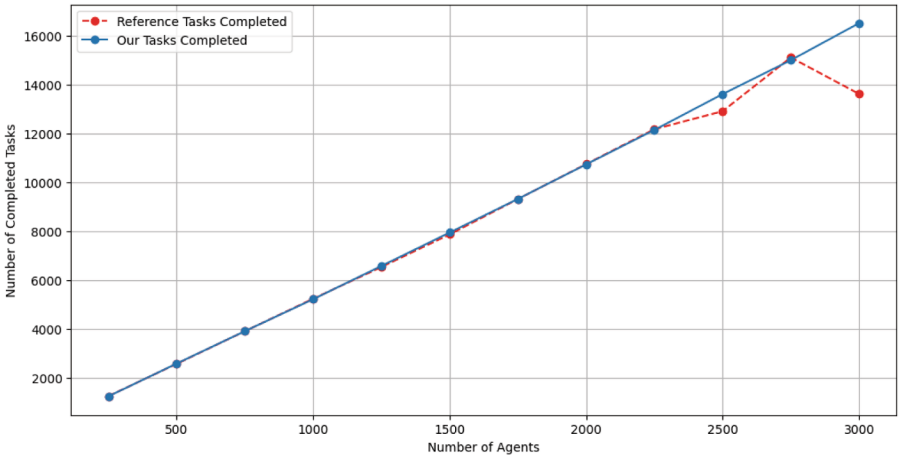


Fig. 5. Number of completed tasks versus number of agents over 2,500 timesteps (semilogarithmic scale).

Figure 6 plots the number of idle agents as the total agent count increases. The results show an exponential growth in idle agents beyond a certain scale, reflecting diminishing returns due to limited task availability and increased congestion. Nevertheless, the proposed algorithm mitigates this effect by more effectively distributing tasks and reducing congestion-induced blocking.

Figure 7 reports the average task completion time. While increasing the number of agents initially reduces completion time, a sharp increase is observed in

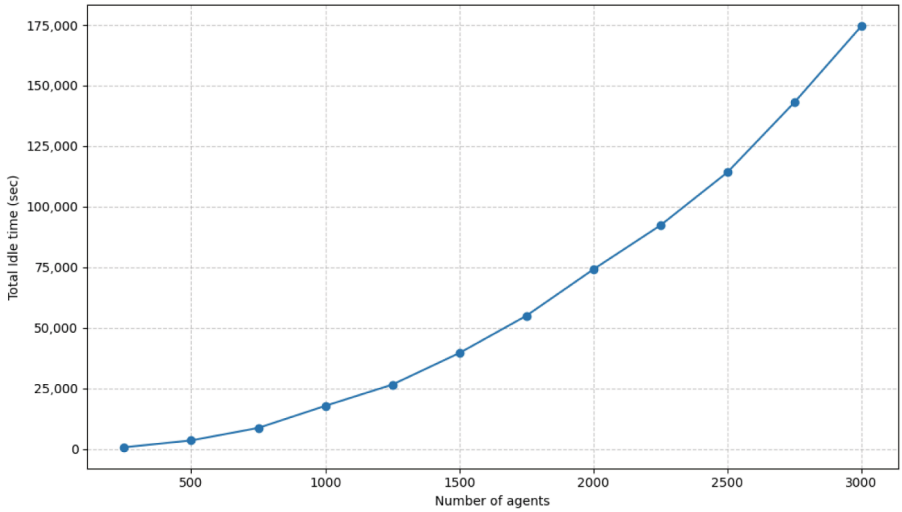


Fig. 6. Number of idle agents as a function of total agent count.

the reference algorithm at 3,000 agents, indicating severe congestion and inefficiencies. In contrast, our approach exhibits more stable behavior, suggesting improved scalability and congestion handling.

Due to memory limitations of the testing platform, simulations could not be executed beyond 3,000 agents. Nevertheless, the observed trends strongly suggest that the proposed method scales more gracefully than the baseline under extreme density.

5 Conclusion

This paper presents a scalable lifelong MAPD framework that integrates congestion-aware path planning, deviation handling, and heuristic task allocation with priority-based conflict resolution. Building upon the robustness of PIBT and its congestion-aware extensions, we introduce an enhanced task allocation strategy that significantly improves throughput under high agent density.

Experimental results demonstrate that the proposed approach consistently outperforms a strong reference implementation, successfully managing up to 3,000 agents on a large-scale sortation map of size 140×500. The improvements are particularly evident in highly congested scenarios, where task completion rates remain stable and average completion times degrade more gracefully.

Future work includes validating these results across additional benchmark maps and task distributions [14,17], as well as further investigating the causes of exponential growth in idle agents at large scales and how adaptive task generation or predictive allocation strategies may alleviate this limitation.

A Appendix: NP-Completeness of MAPF and MAPD

This appendix provides a concise proof sketch establishing the NP-completeness of the Multi-Agent Path Finding (MAPF) problem and its extension, the Multi-Agent Pickup and Delivery (MAPD) problem. The proofs follow standard complexity-theoretic arguments commonly adopted in the MAPF literature.

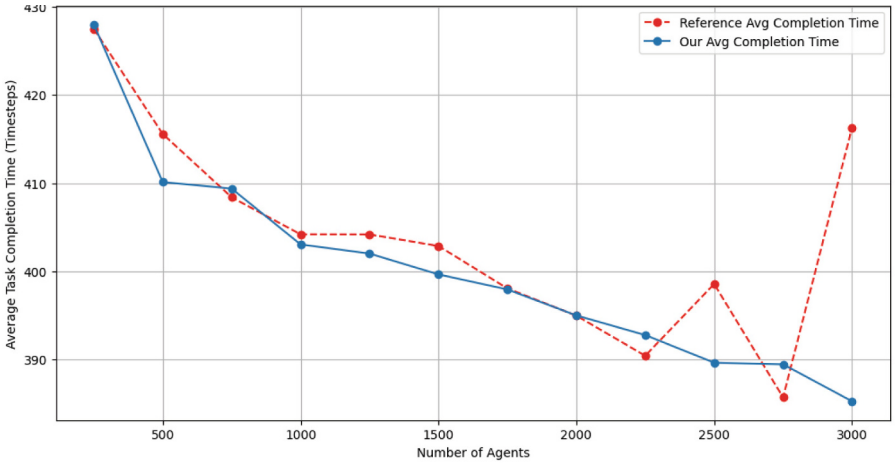


Fig. 7. Average task completion time as the number of agents increases.

A.1 NP-Completeness of MAPF

MAPF Is in NP. The MAPF problem belongs to the complexity class NP because any candidate solution can be verified in polynomial time. Given a set of paths, one for each agent, verification proceeds by checking two conditions. First, each path must connect the agent's start vertex to its goal vertex while respecting graph adjacency. Second, the paths must be collision-free, meaning that no two agents occupy the same vertex at the same timestep (vertex conflict) and no two agents traverse the same edge in opposite directions simultaneously (edge conflict).

Both checks can be performed efficiently. Path validity can be verified by scanning each agent's path, and collision detection can be performed by comparing agent positions at each timestep. Since the number of agents and timesteps is finite and bounded by the input size, the overall verification process runs in polynomial time.

MAPF is NP-Hard. MAPF is NP-hard via a polynomial-time reduction from the Sliding Tile Puzzle (also known as the 15-puzzle, Fig. 8), a well-known NP-complete problem.

The Sliding Tile Puzzle consists of an $n \times n$ grid populated by $n^2 - 1$ numbered tiles and a single empty cell. A move consists of sliding a tile into the empty space, and the objective is to transform an initial configuration into a specified goal configuration.

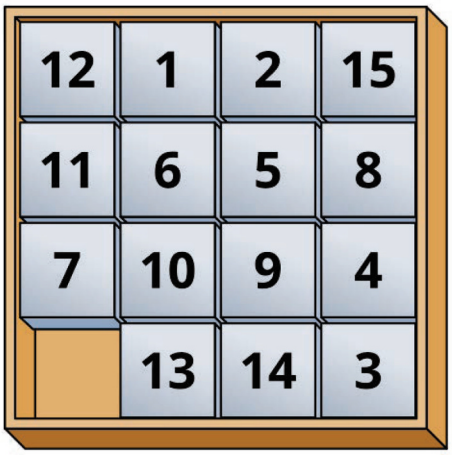


Fig. 8. Example of the 15-puzzle. The objective is to rearrange tiles into numerical order by sliding them into the empty space.

To reduce the Sliding Tile Puzzle to MAPF, we construct a MAPF instance as follows. Each grid cell corresponds to a vertex in the MAPF graph, and edges connect adjacent cells. Each numbered tile is treated as an agent whose start position corresponds to the tile's initial location and whose goal position corresponds to the tile's desired final location. The empty cell serves as the only free vertex, enforcing that agents can move only by swapping positions with this empty location, exactly mirroring the puzzle's movement constraints.

A collision-free solution to the MAPF instance corresponds directly to a valid sequence of tile movements solving the puzzle. Conversely, any solution to the puzzle induces collision-free paths for all agents in the MAPF instance. The construction involves a linear mapping of tiles and grid cells and can therefore be completed in polynomial time.

Conclusion for MAPF. Since the Sliding Tile Puzzle is NP-complete, the reduction to MAPF is polynomial, and MAPF is in NP, it follows that MAPF is NP-complete.

A.2 NP-Completeness of MAPD

Problem Definitions. The MAPF problem takes as input a graph and a set of agents, each with a fixed start and goal location, and asks for collision-free

paths for all agents. The MAPD problem generalizes MAPF by introducing tasks, where each task consists of a pickup location and a delivery location, and agents must execute these tasks without collisions, possibly under time or cost constraints.

Reduction from MAPF to MAPD. To prove the NP-hardness of MAPD, we reduce MAPF to MAPD in polynomial time. Given an arbitrary MAPF instance, we construct a corresponding MAPD instance by assigning each agent exactly one task. For each agent a_i in MAPF, we define a task with pickup location equal to the agent's start vertex and delivery location equal to its goal vertex.

This construction forces each MAPD agent to execute precisely the same movement required in the MAPF instance. Any collision-free solution to the MAPD instance, therefore, induces a valid solution to the original MAPF instance. Conversely, if the MAPF instance has no solution, then the corresponding MAPD instance is also infeasible.

The reduction is straightforward and linear in the number of agents, since it only introduces one task per agent and does not alter the underlying graph.

Conclusion for MAPD. MAPD is in NP because a candidate solution consisting of agent paths and task execution sequences can be verified in polynomial time. Since MAPF is NP-complete and reduces to MAPD in polynomial time, MAPD is NP-hard. Together, these results imply that **MAPD is NP-complete**.

References

1. Wurman, P.R., D'Andrea, R., Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI Mag. **29**(1), 9–19 (2008)
2. Li, J., et al.: Departure scheduling and taxiway path planning under uncertainty. In: AIAA Aviation 2019 Forum (2019)
3. Li, J., et al.: Scalable rail planning and replanning: winning the 2020 flatland challenge. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 31, pp. 477–485 (2021)
4. Stern, R.: Multi-agent path finding - an overview. In: ARTIFICIAL INTELLIGENCE: 5th RAAI Summer School. Dolgoprudny, Russia (2019)
5. Wikipedia Contributors: Multi-agent pathfinding, Wikipedia, The Free Encyclopedia (2024). https://en.wikipedia.org/w/index.php?title=Multi-agent_pathfinding&oldid=1230323665
6. Kaduri, O., Boyarski, E., Stern, R.: Experimental evaluation of classical multi-agent path finding algorithms. In: AAAI Symposium on Combinatorial Search. https://ojs.aaai.org/index.php/SOCS/article/download/18560/18349/22079
7. Čáp, M., Vokřínek, J., Kleiner, A.: Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In: Proceedings of the International Conference on Automated Planning and Scheduling, pp. 324–332 (2015)

8. Ma, H., Li, J., Kumar, T., Koenig, S.: Lifelong multi-agent pathfinding for online pickup and delivery tasks. In: Proceedings of the 16th International Conference on Autonomous Agents and MultiAgent Systems, pp. 837–845 (2017)
9. Grenouilleau, F., van Hoeve, W.-J., Hooker, J.N.: A multi-label a* algorithm for multi-agent pathfinding. In: Proceedings of the International Conference on Automated Planning and Scheduling, pp. 181–185 (2019)
10. Henkel, C., Abbenseth, J., Toussaint, M.: An optimal algorithm to solve the combined task allocation and path finding problem. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4140–4146 (2019)
11. Kudo, F., Cai, K.: A TSP-based online algorithm for multi-task multi-agent pickup and delivery. IEEE Robot. Autom. Lett. **8**(9), 5910–5917 (2023)
12. Chen, Z., Harabor, D., Li, J., Stuckey, P.J.: Traffic flow optimisation for lifelong multi-agent path finding. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 18 (2024)
13. Andreychuk, A., Yakovlev, K., Surynek, P., Atzmon, D., Stern, R.: Multi-agent pathfinding with continuous time. Artif. Intell. **305**, 103662 (2022)
14. Shen, B., Chen, Z., Cheema, M.A., Harabor, D.D., Stuckey, P.J.: Tracking progress in multi-agent path finding. arXiv preprint (2023)
15. Chan, S.-H., et al.: The league of robot runners: competition goals, designs, and implementation (2023). https://www.leagueofrobotrunners.org/
16. Okumura, K., Machida, M., Défago, X., Tamura, Y.: Priority inheritance with backtracking for iterative multi-agent path finding. Artif. Intell. **310**, 103752 (2022)
17. Stern, R., et al.: Multi-agent pathfinding: definitions, variants, and benchmarks. In: Proceedings of the International Symposium on Combinatorial Search, vol. 10, no. 1, pp. 151–158 (2021)