Flow-Based Task Assignment for Large-Scale Online Multi-Agent Pickup and Delivery

Yue Zhang
Monash University
Melbourne, Australia
Yue.Zhang@monash.edu

Zhe Chen
Monash University
Melbourne, Australia
Zhe.Chen@monash.edu

Daniel Harabor
Monash University
Melbourne, Australia
Daniel.Harabor@monash.edu

Pierre Le Bodic
Monash University
Melbourne, Australia
Pierre.LeBodic@monash.edu

Peter J. Stuckey
Monash University
Melbourne, Australia
Peter.Stuckey@monash.edu

ABSTRACT

We study the online Multi-Agent Pickup and Delivery (MAPD) problem, where a team of agents must repeatedly serve dynamically appearing tasks on a shared map. Existing methods either rely on simple heuristics, which result in poor decisions, or employ complex reasoning, which suffers from limited scalability under real-time constraints. In this work, we focus on the task assignment subproblem and formulate it as a minimum-cost flow over the environment graph. This eliminates the need for pairwise distance computations and allows agents to be simultaneously assigned to tasks and routed toward them. The resulting flow network also supports guide path extraction which accelerates planning under real-time constraints. This approach supports real-time execution and scales to 20,000 agents and 30,000 tasks within 1-second planning time, outperforming existing works in terms of computational efficiency, assignment quality and scalability.

KEYWORDS

Multi-Agent Pickup and Delivery; Task Assignment; Multi-Agent Path Finding

ACM Reference Format:
Yue Zhang, Zhe Chen, Daniel Harabor, Pierre Le Bodic, and Peter J. Stuckey. 2026. Flow-Based Task Assignment for Large-Scale Online Multi-Agent Pickup and Delivery. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/MQIK8423

1 INTRODUCTION

Multi-Agent Pickup and Delivery (MAPD) is a fundamental problem in autonomous multi-robot systems that requires a team of agents to continuously execute a large amount of tasks distributed over a shared environment while avoiding collisions with each other [19]. In MAPD, each task consists of transporting an item from a pickup location to a delivery location, and agents operate in an online setting where tasks arrive continuously and must be assigned and executed in real time. This problem has wide applications in warehouse automation, autonomous aircraft-towing vehicles [21], office robots [34] and video games [20]. MAPD involves two subproblems: assigning available tasks to agents and planning collision-free paths for agents to complete tasks. A core challenge lies in balancing task assignment and path planning under strict runtime constraints, and in the presence of thousands or even tens of thousands of agents and tasks. Thus, solvers must react quickly to task arrivals and system state changes, while also avoiding congestion and conflicts.

Some existing approaches solve MAPD offline, where all tasks are known in advance, and try to compute globally optimal solutions for the combined problem within a given runtime [10, 14, 17]. These algorithms can provide high-quality solutions. However, they assume all tasks are known in advance and enough planning time is given upfront, i.e., minutes, hours or more. As a result, they are not designed for online operations, where tasks are not all known *a priori* and agents should not wait while the planner computes. In addition, they also do not scale well beyond dozens of agents due to the high computational complexity of joint reasoning. To address the online setting, Token Passing (TP) and Token Passing with Task Swaps (TPTS) were proposed [19]. TP assigns tasks in a greedy manner by passing a token among agents, allowing them to claim tasks and plan paths one by one. TPTS further allows task swaps between agents. While conceptually simple and reactive, TP and TPTS often produce suboptimal assignments and suffer from computational bottlenecks from time-dependent path planning, especially as the team size increases. To further optimise the solution quality, Regret-based Marginal-cost Based Task Assignment (RMCA) [5] optimises the solution by integrating and solving them as a combined problem. However, these works still suffer from computation bottlenecks when scaling to hundreds of agents.

This work focuses on the assignment side of MAPD, which can be understood as a matching problem which can be solved efficiently using network simplex [2]. However, computing the matching requires an all-pair distance matrix that is, by itself, computationally prohibitive. Existing literature tries to improve the efficiency of matching [1, 28]. We instead propose a new minimum-cost flow formulation, which works directly on the environment graph, eliminating the all-pairs distance matrix and subsequent fully connectted bipartite matching. This is much faster in practice. As a second contribution, we show a way to lightly integrate with the planner and

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/MQIK8423

incorporate traffic estimates from the planner for assignment. The resulting flow also serves as a guide path for the planner.

Flow has been studied in a range of other different problems, including coordinated path finding in a time-expanded network for one-shot Multi Agent Path Finding (MAPF) [30] problem, or computing heuristic costs for planning in MAPF and Anonymous MAPF [23, 28, 32]. Our contribution is a spatial assignment formulation for online MAPD and a closed-loop integration. This use of flow differs from prior work in the area. As a result, we integrate the leading planner from this area, and the flow solution also improves the planner. Our formulation keeps assignment time-independent and lightweight while accounting for cost overheads from spatial congestion induced by agent interactions. In a range of experiments, we show that our approach outperforms existing baselines in terms of solution quality and runtime, and scales to scenarios with over 20,000 agents and 30,000 tasks on large maps.

2 PROBLEM SETUP

A *Multi-Agent Pickup and Delivery* (MAPD) problem consists of n agents $A = \{a_1, \ldots, a_n\}$ on a known 2D grid map $G = (V, E)$, where V is a set of vertices (grid cells) and E is a set of edges that connects adjacent cells. Each agent starts at a unique location and is responsible for repeatedly completing delivery tasks. Each task consists of a pair of locations, a pickup location and a delivery location. To complete a task, an agent must reach the pickup location and then move to the delivery location in order. Tasks are not known in advance. Instead, they appear dynamically over time. A global *task pool* maintains all currently available tasks. Let m be the number of tasks available. Once a task is completed, a new task is released into the pool to maintain a constant number of tasks in the pool (or according to a predefined release policy).

The system runs in discrete time steps. At each step, the solver must assign tasks to available agents and update plans for ongoing assignments. Agents can move to adjacent free cells or wait in place. Collision avoidance must be enforced: agents cannot occupy the same cell at the same time or traverse the same edge in opposite directions simultaneously. The goal is to assign tasks and plan paths to free agents in a way that maximises overall throughput (i.e., the number of completed tasks over time)

We adopt an online execution model similar to that used in [35] and [4], where solvers plan while executing. In this setting, at each step, the solver is given a fixed planning time window, i.e., determined by the execution time for a single action, to compute both task assignments and movement decisions. If the solver does not return within this time limit, agents will pause and wait during that step, leading to delays in task completion.

3 RELATED WORK

3.1 Path Planning Approaches

The path planning problem is a well-studied problem called Multi-agent path finding (MAPF) [30], where a team of agents navigate from the given start to goal positions while avoiding collisions. In this problem, each agent receives only one fixed task that is assumed to be given in advance. Classical approaches include centralised solvers like Conflict-Based Search (CBS) [29] and its many variants, which offer good solution quality, but scale poorly with the number of agents. More recent work explores online MAPF, where planning and execution are concurrent [35]. In this online setting, solvers must return plans or partial plans within a fixed planning time, called *committed paths*, and agents act on committed paths while planners are planning for future paths. The path planning under this setting becomes more challenging, as planners should react quickly. Solvers that are able to produce partial solutions quickly become more desirable. For example, [35] proposes to use a fast method to compute an initial solution, then commit the first several actions and keep improving the uncommitted part of the solution during execution. Similar ideas have also been proposed in [6]. At each step, the solver commits only the next action for each agent, and keeps improving a spatial guide path during execution time.

3.2 Task Assignment Approaches

The task assignment problem consists of finding a minimum-cost set of $\min(n, m)$ disjoint edges going from agents to tasks. This can then be solved optimally using the Hungarian Method [12] or network flow solvers. Related problems have also been widely studied in different areas, such as the multi-robot task allocation (MRTA), where a team of robots need to visit a set of target locations [8, 11, 13], and the vehicle routing problem (VRP) where multiple vehicles need to deliver products to a group of customers [15, 16]. Additional variations of this problem incorporate different constraints, such as adding deadlines or precedence constraints for tasks [3] and introducing time windows to tasks and robots. [26]. Approaches in these topics focus more on optimising under a complex model and assignment constraints from the problem. However, they often rely on simplified cost models to represent the travel time or distance, such as Manhattan distance or single-agent shortest path cost, and robots' coordination is often assumed to be optimistic, for example, ignoring the collision avoidance problem. This assumption does not hold in MAPD problems, where congestion and path conflicts substantially affect task completion cost.

3.3 MAPD Approaches

Several works address MAPD from an offline perspective, which assumes all tasks are known and the computation time is given upfront. In [22], the authors start to generalise the combined problem of path finding and task assignment and solve it with answer set programming. They proposed a three-phase method, which scales to only 20 agents. Meanwhile [17] proposes a two-stage approach that first models the task assignment problem as a Travelling Salesman Problem problem, and then computes the task assignment and plans paths using MAPF techniques. In [10], the authors combine optimal task assignment with CBS-based path planning and solves the joint problem optimally. A similar idea is proposed in [14], which employs branch-and-cut-and-price to solve the combined problem optimally offline. These methods provide strong solution quality but do not support online execution, and they scale poorly beyond hundreds of agents due to their combinatorial complexity.

In contrast, online MAPD methods handle dynamically generated tasks during execution. Token passing (TP) and Token Passing with Task Swaps (TPTS) [19] solve online MAPD in two stages. In TP, agents select the closest task and plan their path to it one by one, and once a task is assigned to an agent, the assignment becomes

fixed and cannot be swapped. While TPTS allows swapping for tasks that have not been picked up yet. [19] also proposed a centralised algorithm, CENTRAL, which uses the Hungarian Method to solve the task assignment problem and computes paths using CBS [29]. Regret-based Marginal-cost Based Task Assignment (RMCA) [5] is the existing state-of-the-art online method that integrates task assignment and path planning and solves them at the same time. RMCA starts with an initial assignment and plan for each agent, and then improves the solution within the runtime available. These methods can handle an online environment, but they often only compute suboptimal solutions for fewer than hundreds of agents due to their computational complexity.

3.4 Anonymous MAPF and Target Assignment

A similar problem is called Anonymous MAPF (AMAPF) [30]. AMAPF studies planning when n agents are interchangeable with respect to n unlabeled targets, which also requires the solver to solve goal assignment and collision-free routing. Research for this topic tries to solve the task assignment by accelerating matching [28], post-refining from greedy assignment [23], or solving the combined problem through a time-expanded network [18]. Compared to MAPD, AMAPF typically handles single-location, anonymous goals ($n = m$), often in one-shot, makespan-oriented settings. In contrast, MAPD involves pickup–delivery pairs with precedence and non-anonymous deliveries (each delivery is tied to a specific destination). The operations are under real-time, tasks are unknown in advance, and solvers are evaluated by throughput.

4 PRELIMINARIES

4.1 Traffic-Guided Planner for Path Planning

In our framework, we decouple task assignment from path planning and solve the task assignment problem. For path planning, we directly use a recent state-of-the-art online MAPF planner, Guided PIBT [6]. This planner operates efficiently in large-scale online MAPF settings by reasoning about future congestion and using *guide paths* as heuristics for determining next actions. We also integrate its traffic-aware cost models and guide paths into our task assignment model (described in later sections).

4.1.1 Traffic-Aware Cost Models. The planner first plans a time-independent guide path for each agent using focal search. During the search, two types of congestion are estimated and incorporated into edge costs. When traversing an edge e from vertex v_1 to v_2, the planner considers: (i) Vertex Congestion (p_v): This estimates the total delay that will occur in the future at vertex v, calculated using $p_v = \left\lceil \frac{n_v - 1}{2} \right\rceil$ where n_v is the total number of agents entering vertex v on its planned time-independent path. (ii)Contraflow Congestion (c_e): This estimates the potentially large delays caused by agents being pushed by other agents to avoid collisions which intend to traverse e in different directions. This is computed as $c_e = f_{v_i,v_j} \cdot f_{v_j,v_i}$ where f_{v_i,v_j} denotes the number of agents currently planned to traverse edge e in the direction from v_i to v_j. The contraflow congestion calculation is found to be effective on warehouse-like maps, where there are many narrow corridors and agents traversing different directions are more likely to incur large wait times [6]. The planner then combines these values into the edge cost $FCost(e) = 1 + p_{v_2} + c_e$

4.1.2 Guide Path Planning and Refinement. Initially, the planner plans time-independent guide paths for each agent one by one that minimise the traffic cost (sum of edge cost along the path) while considering the congestion caused by the planned agents. After all initial paths are computed, the system refines them iteratively by re-planning a subset of agents based on updated congestion estimates and updates the edge weights accordingly within the given runtime. A rule-based solver, Priority Inheritance with Backtracking (PIBT) [24], then uses the resulting guide path as a heuristic to determine the movement for each agent while avoiding collisions.

5 TASK ASSIGNMENT FORMULATION

5.1 Linear Assignment Formulation

A straightforward approach to task assignment is to model it as a minimum-cost bipartite assignment problem between available agents (agents that are not delivering items) and available tasks (tasks that are not picked up yet). An example of this formulation is shown in Figure 1(b). For each agent-task pair, the edge cost is typically the shortest path distance between the agent's start location and the task location. This is typically computed by Dijkstra, in which the search starts from the agent's start location and terminates when all tasks are reached. Then, a complete bipartite graph is constructed from all the agent-task pairs. The goal is to find a minimum-cost one-to-one assignment that minimises the total assignment cost. This problem can be solved optimally as a minimum-cost flow problem on a bipartite graph.

While this formulation is intuitive and provides optimal one-shot assignments, it has several limitations. First, it requires computing all agent-task distances, which becomes costly as the number of agents and tasks increases.[1] Second, the scalability of this formulation is limited in large-scale settings due to the quadratic number of edges. For example, $n = 10000$ agents and $m = 15000$ tasks will result in 150 million edges, and solving the problem with this model at this scale becomes challenging.

5.2 Flow-Based Model

5.2.1 Graph Construction. To overcome these limitations, we propose a spatial flow-based formulation that operates directly on G. As shown in Figure 1(c), instead of computing agent-task costs explicitly, we embed both agents and tasks into a single flow network constructed from the map topology.

- Each cell $v \in V$ of the grid map becomes a node in the directed graph. Then add edges between adjacent free cells, the capacity of each edge is n.
- Add a dummy source node with edges to the current positions of agents that are not delivering, each with max flow one unit. Force a flow from the source of $\min(m, n)$.
- Add a dummy sink node and edges from available pickup locations, each with max flow of one unit.
- Edge costs that connect the map cells can be unitary (1) or an estimated cost, such as traffic estimations.

[1] These can be precomputed if fixed unit costs are used, but not for traffic costs.

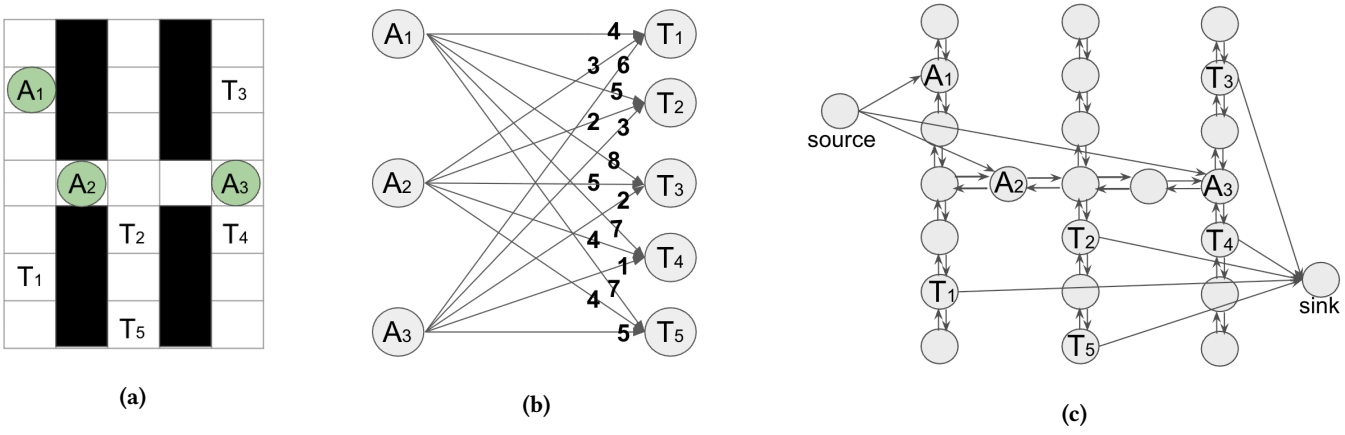

Figure 1: Illustration of task assignment models. (a): the example instance, where A_i represents the agent, and T_i represents the task (pickup location). (b): the bipartite linear assignment model, where each agent connects to every task and the numbers on the edge represent the edge cost, which is the shortest path distance. (c): the flow model, where the map is embedded directly as a flow network and the edge cost is the unit cost.

Algorithm 1 Retrieval from Flow Solution

```
Input: Directed flow network G = (V, E) with flow values f : E → N; set of agents A
Output: Assigned task and guide path for each agent
for all a_i ∈ A do
    v ← node at agent a_i's current location
    P ← empty list
    while v is not a task node do
        Append v to P
        Choose e = (v, v') an edge with positive flow f(e) > 0 from current node v
        f(e) ← f(e) − 1. Remove one unit of flow from e
        v ← v'
    Assign task at v to a_i
    Store P as the guide path for a_i
```

Unlike time-expanded formulations, we do not model time explicitly or enforce capacity constraints to prevent collisions in the flow network. Instead, we allow multiple units of flow to traverse the same edge, and leave the collision avoidance to path planners.

5.2.2 Assignment and Guide Path Retrieval from Flow. This formulation enables the assignment to be solved as a single minimum-cost flow problem. Once solved, we extract task assignments and guide paths for each agent by tracing the unit flow from the agent's current location to a task node in the flow network. The computed flow indicates not only which agent should go to which task, but also a spatial path toward the task, which is aware of traffic congestion and can be used as a guide path for the planners.

As shown in Algorithm 1, for each agent, we start from its corresponding location node and follow the outgoing edges with positive flow. On each node, we select one outgoing edge with positive flow to reach the next node and subtract one unit of flow from the edge. We continue traversing the graph until we reach a task node (i.e., a node with an outgoing edge to the sink), and collecting the visited nodes along the way as the agent's guide path. Once a task node is reached, we assign the corresponding task to the agent and store the constructed path for the downstream planner. Note that Algorithm 1 returns one of possibly many optimal assignments that can be derived from the input flow.

5.2.3 Alternative Edge Costs. The default edge cost for our flow model is the edge cost from the map, i.e., unit cost for grid maps. We also support other cost functions that incorporate different considerations of the problem. For example, using estimated traffic congestion cost helps agents avoid waiting at frequently blocked or delayed areas. Here, we present two alternative dynamic edge costs based on the traffic.

- **Traffic Cost from Planner Estimations:** Here we use the same traffic costs as the planner [6], which uses future traffic estimations to plan paths for agents. We use edge costs $Fcost()$ based on the same traffic estimations in our flow model. That is, at each planning cycle, we use the guide paths of agents that are currently delivering items (computed by the planner) to compute $Fcost(e)$ for each edge in the flow model. Note that agents delivering items will not be reassigned a new task, so this estimate is stable.
- **Avg Waiting Time from Execution:** To support integration with other planners that do not have traffic estimation, we propose an alternative edge cost model that calculates the average waiting time up to this point of the execution (past traffic). We maintain a record of agent waiting times on each edge during execution and incorporate these traffic statistics into the edge cost. For each directed edge $e \in E$, we track two values: the total waiting time W_e to traverse e and the total number of traversals N_e of e. The cost of e is set to its historical average traversal time:

$$PCost(e) = 1 + \begin{cases} \frac{W_e}{N_e} \text{ if } N_e > 0, \\ 0 \text{ otherwise.} \end{cases}$$

which is the unit cost plus the average wait time. Additionally, we apply a decay factor $\gamma \in (0, 1]$ to both W_e and N_e at each planning window. We update W_e and N_e at each step using:

$$W_e \leftarrow \gamma W_e + \begin{cases} \text{t if an agent traverses } e \text{ after waiting } t, \\ 0 \text{ otherwise.} \end{cases}$$

$$N_e \leftarrow \gamma N_e + \begin{cases} 1 \text{ if an agent traverses } e \text{ after waiting } t, \\ 0 \text{ otherwise.} \end{cases}$$

This helps the model emphasize more recent congestion observations while discounting older traffic conditions.

5.3 Complexity Analysis

We analyse and compare the computational complexity of our flow-based task assignment model against the baseline assignment problem. Throughout the analysis, we use $n \leq |V|$ and $m \leq |V|$ and the fact that on a graph of a grid map with $|V|$ nodes and $|E|$ edges, we have $|E| = O(|V|)$. Note that, because the intention is to use edge costs that may not be integer, solving techniques that rely on *cost scaling* [2, Chapter 10] are generally not appropriate for either approach.

5.3.1 Linear Assignment. In the assignment approach, there are two stages of computation:

- Edge weights computation: for each agent, we perform a shortest path search (e.g., Dijkstra) on the map of $|V|$ nodes and $|E|$ edges:
$$O(n(|V| + |E|) \log |V|)) = O(n|V| \log |V|)).$$
- Solving an unbalanced assignment problem [27] takes, in the worst case:
$$O(\min(n^2, m^2)m)$$
time, using the fact that, if $n \leq m$, only one of the closest n tasks to an agent can be assigned to that agent, hence only n^2 edges are necessary, and, similarly, m^2 edges if $m \leq n$.

Thus, the total time complexity of this approach is

5.3.2 Network Flow. Our flow-based model avoids explicit agent-task distance computation and operates on a simple graph representation of the map. The network is constructed directly from the map:

- Nodes: $|V| + 2 = O(|V|)$, including one node for each map cell plus a source node and a sink node.
- edges: $|E| + n + m = O(|V|)$, including one edge for each traversable map edge plus n edges that connects the source node to n agents' current positions and m edges that connects the sink node to m task locations.

Using the algorithm given by [25], given at most $|V|$ edges are capacitated, we find

$$O(|V|^2 \log^2 |V|).$$

The Network Simplex, which we use in our experiments, has the same worst-case time complexity, supposing edge costs do not exceed a constant [33].

Unlike linear assignment, the spatial flow model scales primarily with the map size rather than the number of agent-task pairs. Indeed, if the number n of agents grows linearly with $|V|$ (e.g. 50% agent density), then the worst-case time complexity of solving the linear assignment is

$$O(|V|^3),$$

whereas that of the Network Flow is still

$$O(|V|^2 \log^2 |V|).$$

This makes it more suitable for large-scale MAPD problems, where $n \times m$ can easily exceed $|E|$ and $|V|$ by orders of magnitude.

6 PLANNER INTEGRATION

We integrate our flow model with the planner in two ways.

- **Traffic Cost Estimation from Planner to Flow:** As also illustrated in previous subsections, we model the edge cost in flow using the edge cost estimates based on agents' guide paths and expected future traffic from the planner (*FCost*). This ensures the flow solver uses the same heuristic as the planner, and assigns tasks in a way that avoids regions likely to become congested. As a result, task assignments and subsequent path planning are optimised with respect to the same underlying traffic model.
- **Guide Path Initialisation from Flow to Planner:** Although the planner is fast and scalable, at each time step, initialising guide paths for thousands of agents on large maps can exceed the strict planning time limit (e.g., one second). In real-time settings, the planning time becomes crucial, as agents are waiting for plans. Therefore, we warm-start the planner by using the path extracted from the flow solution. At every timestep, once a minimum-cost flow is computed, we extract guide paths for each agent directly from the solution (as described in Algorithm 1). These paths indicate not only the task assigned to each agent but also an initial route toward the task. We then pass these guide paths to the planner as initial guide path. This accelerates path generation by avoiding redundant search and helps to start the refinement process more quickly.

Algorithm 2 outlines the core planning loop of our framework, when integrating with the path planner. At each timestep, the system updates its state by identifying assignable agents and currently available tasks. If both are present, a minimum-cost flow network is constructed directly over the map topology, where edge costs reflect congestion-aware traversal costs as estimated by the planner. Once the flow is solved, we extract both the task assignments and the guide paths from the resulting unit flows. These guide paths represent spatial trajectories toward assigned tasks, which are then used to warm-start the planner. For each agent, if a new task has been assigned since the last planning step, the agent's guide path is updated accordingly. The planner (e.g., Guided-PIBT or any other compatible online MAPF solver) then uses these guide paths and updated congestion estimates to compute collision-free movement decisions. The system executes the resulting actions, and the environment state is advanced before the next planning step begins. This tightly coupled loop ensures that task assignments and path planning remain synchronised, while maintaining real-time performance even under large-scale settings.

Algorithm 2 Flow Task Assignment and Path Planning

Input: Map $G = (V, E)$, initial agent and task states
while system is running **do**
  Identify assignable agents A_{free} and assignable tasks T_{free}
  if $A_{\text{free}} \neq \emptyset$ **and** $T_{\text{free}} \neq \emptyset$ **then**
    Construct flow network over G
    Set edge costs with *FCost* from the planner
    Solve minimum-cost flow to assign agents to tasks
    Extract path p_i and task assignment t_i for each agent a_i from flow
  for all agent a_i **do**
    if assigned task different from last step **then**
      set guide path of a_i to p_i
  Run planner (e.g., Guided-PIBT) and updating *FCost*
  Execute actions and advance system state

7 EXPERIMENTS

We implement the whole framework in C++[2] on top of the traffic planner [6]. For the linear assignment and network flow solver, we use the open-source library LEMON [7], which, on these instances, was faster than competitors such as Gurobi. The experiments are conducted on a cloud instance with 32GB RAM, 16 AMD EPYC-Rome CPUs. We run one map from the standard grid-based Multi-Agent Path Finding (MAPF) benchmarks [31] and two warehouse-type maps with distance-biased distributions from a recent MAPF/MAPD competition called League of Robot Runners (LoRR) [4]. The maps are: (i) *Random (R)*: a 64×64 map with 3270 traversable cells and 20% random generated obstacles. The agent team size are tested from 400 to 2000, increasing by 400. (ii) *Warehouse Small (WS)*: a 33×57 map with 1277 traversable cells. Among the free cells, there are 40 "E" locations that represent the working stations in the warehouse, and 342 "S" locations that represents the items locations that needs to be picked up. The agent team size are tested from 200 to 600, increasing by 100. (iii) *Sortation Large (SL)*: a 140×500 map with 54320 traversable cells. There are 620 "E" locations and 31540 "S" locations. The agent team size are tested from 4000 to 20000, increased by 4000.

Note the range of agents is deliberately chosen so that we get to the point of having too many agents on the map, and hence throughput reduces. Following the LoRR competition setup, we keep the number of tasks in the task pool to 1.5 times the number of agents. We also use the problem generator from LoRR to generate agents' start and task locations. The start locations of agents are randomly sampled. For the task distributions, the tasks for *Random* are sampled randomly, and the tasks for the other maps are selected only from "E" and "S" locations and are generated based on a distance-based warehouse distribution model used in LoRR, where tasks with fewer distance to working stations will have higher probability to be selected.

7.1 Runtime of different methods

We compare the runtime performance of **Linear Assignment** and **Flow** with unit cost as the edge cost on the map. When using unit cost, the edge cost of Linear Assignment can be precomputed, as for each pair of locations, the cost is static. However, for dynamic costs (e.g., traffic-dependent weights) or very large maps where storing dense distance tables is infeasible, the cost computation should be considered. Therefore, we record both **Linear Assignment** (treating the cost computation time as free) and **Linear Assignment + Dijkstra** (including cost computation using Dijkstra). The time limit for returning assignments and actions is set to 10 minutes, and we simulate for 1000 timesteps. We use *Warehouse Small* and *Sortation Large* to illustrate runtime behaviour across scales.

[2]Code are available at https://github.com/YueZhang-studyuse/LifelongTA

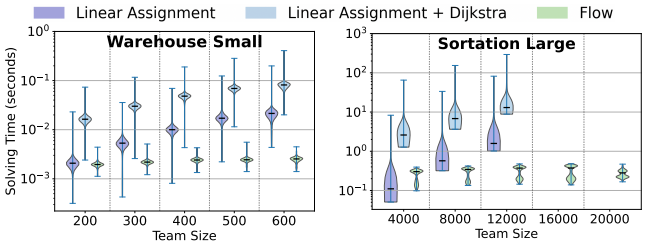

Figure 2: Solving time distributions for different methods and team sizes on two maps. For team sizes 16,000 and 20,000 on Sortation Large, Linear Assignment fails to compute even the first step within 10 minutes.

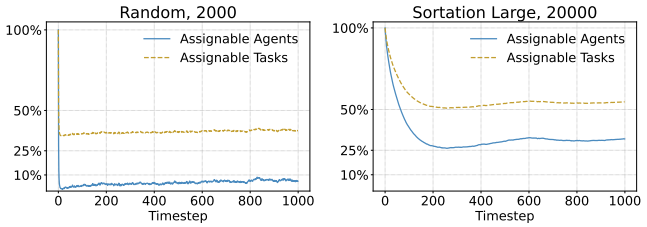

Figure 3: Percentage of assignable agents and tasks over time on Random (2000 agents) and Sortation Large (20,000 agents). "Assignable" means the number of agents/tasks that can be assigned or swapped.

Figure 2 shows the runtime distributions of different methods across different timesteps. Note the time for Linear Assignment varies during execution, because the number of available agents and tasks varies. Overall, Flow achieves consistently low solving times across all team sizes. On *Warehouse Small*, both methods return solutions within 1s, which is acceptable for real-time performance. Flow has a consistent runtime under 0.01s regardless of the team size, while linear assignment and the edge computation both show increasing runtime as team size grows. The performance gap is more dramatic on *Sortation Large*. Linear assignment and its edge cost computation have a quadratic runtime growth. It never completes the task assignment in real-time, and fails to complete within 10 minutes at team sizes 16,000 and 20,000. In contrast, Flow maintains low solving times (<1s) even at this scale.

7.2 Throughput of different methods

We now evaluate the throughput performance of different task assignment methods. Throughput is defined as the number of tasks

Map	n	No Timeout (% vs Greedy)		With 1s Timeout (% vs Greedy)			
		Greedy*	Flow-Unit Cost*	Greedy	Flow-Unit Cost	Flow-Traffic	Flow-Avg Waiting
R	400	6936	**7005 (↑0.99%)**	6925	6972 (↑0.68%)	6964 (↑0.56%)	**6975 (↑0.72%)**
R	800	12688	**12840 (↑1.20%)**	12660	**12826 (↑1.31%)**	12745 (↑0.67%)	12787 (↑1.00%)
R	1200	15839	**16253 (↑2.61%)**	15874	16331 (↑2.88%)	16205 (↑2.09%)	**16402 (↑3.33%)**
R	1600	16172	**17175 (↑6.20%)**	16316	16383 (↑0.41%)	**17097 (↑4.79%)**	17001 (↑4.20%)
R	2000	15411	**15755 (↑2.23%)**	15265	16101 (↑5.48%)	15939 (↑4.42%)	**16111 (↑5.54%)**
WS	200	3508	**3683 (↑4.99%)**	3512	**3642 (↑3.70%)**	3596 (↑2.39%)	3632 (↑3.42%)
WS	300	4773	**4974 (↑4.21%)**	4779	**4919 (↑2.93%)**	4863 (↑1.76%)	4816 (↑0.77%)
WS	400	5570	**5777 (↑3.72%)**	5494	5718 (↑4.08%)	**5761 (↑4.86%)**	5637 (↑2.60%)
WS	500	5791	**5888 (↑1.68%)**	5894	5829 (↓1.10%)	**6174 (↑4.76%)**	5819 (↓1.27%)
WS	600	5585	**5662 (↑1.38%)**	5593	5652 (↑1.05%)	**6185 (↑10.59%)**	5678 (↑1.52%)
SL	4000	13776	**14518 (↑5.39%)**	13642	13173 (↓3.44%)	13449 (↓1.41%)	**14094 (↑3.32%)**
SL	8000	26355	**27634 (↑4.85%)**	25831	18507 (↓28.37%)	25569 (↓1.01%)	**26177 (↑1.34%)**
SL	12000	33952	**36066 (↑6.23%)**	34414	23136 (↓32.81%)	33355 (↓1.72%)	**34568 (↑0.45%)**
SL	16000	31428	**34295 (↑9.12%)**	31993	26490 (↓17.21%)	**34277 (↑7.14%)**	33975 (↑6.19%)
SL	20000	29029	**31379 (↑8.10%)**	27323	24501 (↓10.33%)	**31658 (↑15.91%)**	29477 (↑7.09%)

Table 1: Throughput results across different maps and team sizes. Column 1 is the map name, column 2 is the team size (number of agents) and column 3-8 show the throughput of different methods. Methods marked with "*" are unconstrained (no timeout) for task assignment and initial guide path computation, followed by 1s refinement. The remaining methods operate under a strict 1-second real-time limit per timestep. Percentages in parentheses indicate relative improvement over the respective Greedy baseline. Bold values highlight the best-performing method within each group.

completed over a 1000-timestep simulation. We compare our flow-based models against a baseline called Greedy, under varying team sizes and map structures. Greedy assigns tasks to agents one by one, by minimising the distance between an agent's current location and its proposed task. The distance is assumed to be computed and cached on demand, and shared by the planner. This is also the task assignment strategy in TP [19]. Greedy only assigns new free agents with tasks and does not allow task swapping, which is the best greedy version we found in our experiments (see appendix).[3] Table 1 summarises the results grouped into two categories.

7.2.1 Unconstrained Time Limit. Columns 3-4 (Greedy* and Flow-Unit Cost*) correspond to the setting where task assignment and initial guide path computation are unconstrained in each timestep, and the path refinement has a 1s timeout limit. These methods benefit from sufficient time to compute good plans, which reveals the potential of each strategy when computation is not a bottleneck.

In this setting, we observe that flow-based assignment consistently outperforms Greedy, even with a unit cost (gains from 0.6% to over 9%). The benefits are particularly apparent in large-scale warehouse settings. This illustrates the value of global optimisation for task-to-agent assignment, particularly when the global optimisation does not have too much time overhead, since the flow-based method is fast. However, as team size increases, the challenge shifts from task assignment to path planning. In particular, we observe that the initial guide path computation becomes the primary bottleneck, i.e., taking up to 54s for 20,000 agents on Sortation Large.

Another key finding is that the importance of task assignment varies significantly across map types and task distributions. In random maps with randomly sampled task distributions, many agents are able to pick up nearby tasks just one step away. As a result, the assignment problem in such settings is trivial, and throughput is dominated by local decisions. In contrast, warehouses and sortation centres impose spatial separation between pickup and delivery points, which makes task assignment a more critical component of overall performance. As shown in Figure 3, nearly all agents are currently delivering and only very few are assignable on the random map, while sortation large map maintains a significant portion of assignable agents.

7.2.2 Strict Real-time Constraints. Columns 5–8 in table 1 represent the strict real-time MAPD setting. We follow the setting in [35] where agents are moving while planning. That is, at each timestep, agents need 1 second to execute actions, and the planner has to commit $k-$step of actions and task assignment within a window of $k-$step's execution time. Any time outs of x seconds means agents are waiting in place to receive further instructions from the planner (all agents will wait for x steps). We set $k = 1$. For our flow-based method, we test three edge costs: **Unit Cost**, **Traffic** (Traffic Cost from Planner Estimations) and **Avg Waiting** (Avg Waiting Time from Execution). For Avg Waiting, we set $\gamma = 0.9$. We only warm-start the planner with the flow solution when the cost model is *Traffic* or *Avg Waiting*, because the flow solution with unit cost is simply the shortest path, which does not avoid any congestion.

Overall, through the available time becomes less, the flow-based model outperforms greedy in most cases. Flow-Unit Cost performs well in smaller or less congested environments; the two congestion-aware variants, **Flow-Traffic** and **Flow-Edge Waiting**, consistently outperform both Greedy and unit-cost flow in congested scenarios. In particular, in situations like 20,000 agents in Sortation Large and 500-600 agents in Warehouse Small, Flow-Traffic under a strict timeout limit has better throughput than that with no timeout. In addition, Flow-Avg Waiting also shows similar performances to traffic estimations, which demonstrates that this estimation is

[3]Note Greedy is not compared in Section 7.1 because it is very fast. LA is not compared in Section 7.2, because it optimises the same assignment objective as our flow and return identical assignment costs but substantially slower as shown in Experiment 1.

f	n	Avg Makespan		Avg Timeouts
		RMCA	Flow	RMCA
2	50	324.96 ± 6.99	**299.24** ± 7.80	74.88 ± 6.97
2	80	288.00 ± 3.91	**242.00** ± 8.99	38.00 ± 3.91
2	100	287.20 ± 3.7	**227.20** ± 12.01	37.20 ± 3.70
5	50	319.72 ± 7.58	**271.64** ± 8.28	219.68 ± 7.55
5	80	218.48 ± 5.65	**197.88** ± 5.10	118.48 ± 5.65
5	100	185.52 ± 4.65	**172.92** ± 6.51	85.48 ± 4.60
10	50	315.48 ± 7.85	**271.16** ± 8.48	268.36 ± 9.57
10	80	216.24 ± 4.84	**192.88** ± 5.06	175.52 ± 7.09
10	100	184.12 ± 4.97	**166.36** ± 7.30	147.04 ± 6.50

Table 2: Average makespan (column 3-4) between RMCA and Flow and average number of timeouts for RMCA (column 5) under varying team sizes n and task frequencies f.

suitable when used with planners that do not produce traffic estimations. We also observe a significant decrease for Flow-Unit Cost on Sortation Large. This is mainly because of the overhead for computing or recomputing (due to frequent task swaps) the guide path in the planner. For example, across 1000 timesteps, Flow-Unit Cost fails to find all initial guide paths in the path planner within 1s for 997 times on Sortation Large with 20,000 agents. Since we currently reschedule tasks every timestep, this indicates that less frequent scheduling may be needed in large-scale scenarios. The other two flow-based variants, which can produce the initial guide path to the planners, are less affected by these runtime overheads. Note Greedy does not suffer from this issue, because it does not allow task swapping, hance there is no need to replan guide paths.

7.3 Flow vs existing MAPD approaches

We compare our approach against RMCA [5], the existing state-of-the-art method for online MAPD[4]. We test our method on their task release policy and published problem instances, where tasks are randomly sampled and tasks are released each f step. In this setting, $f \in 2, 5, 10$ number of tasks are released per timestep, and we measure the makespan (timesteps to finish total tasks) for 500 tasks and $n \in 50, 80, 100$ number of agents. For a given f and n, we test 25 instances. Following the approach of RMCA, we do not let agents wait when planning (both path planning and task scheduling) times out, but we count the number of steps that exceed a 1s runtime limit for RMCA as the total number of timeouts. Note flow approach never times out in all the instances in this experiment.

Table 2 reports the average makespan for our method and RMCA and the average number of timeouts for RMCA (with standard deviations). Our approach consistently achieves lower makespan across all instances. The performance gap becomes larger under high task densities (e.g., $f = 10$), where RMCA incurs many timeouts due to its higher computational overhead. In contrast, our method maintains consistent planning times ($< 1s$) across all timesteps.

7.4 Flow on ultra-large maps

We further evaluate the scalability of our flow-based assignment framework on two ultra-large maps: **Orz900d (Orz)**, size 1491 × 656 with 96603 free cells from [31], and **mp_2p_01 Iron harvest (IH)**, 1912 × 1800 with 6545639 free cells from [9]. We simulate with different team sizes $n \in 10000, 20000$ with Greedy and Flow-Unit Cost. In such massive environments, there is no practical way to compute or store accurate heuristics (e.g., all-pairs shortest paths). Thus, we use a lightweight setup: Manhattan distance as a heuristic and PIBT [24], a simple rule-based solver, for path planning. To reduce the overhead of frequent assignment computation at this scale, we schedule task assignment using flow every k timesteps. That is, the flow solver is invoked once every k steps, and the resulting assignments are fixed for the window. If an agent finishes its task during this interval, it remains unassigned until the next round. We set $k = 10$ for Orz and $k = 30$ for IH.

Map	n	Simulate Steps	Throughput		Avg Time(s)
			Greedy	Flow	Flow
Orz	10000	2000	1031	**1042**	0.367(±0.02)
Orz	20000	2000	2031	**2139**	4.138(±7.24)
IH	10000	5000	2624	**2646**	19.819(±1.20)
IH	20000	5000	5698	**5902**	18.201(±1.56)

Table 3: Throughput between Greedy and Flow-based methods and solving time for Flow. We simulate 2000 steps for Orz and 5000 steps for IH.

As shown in Table 3, the flow-based method achieves higher throughput while maintaining tractable runtime. On IH, which has over 6 million free cells, the flow solver remains stable with runtime under 20 seconds, even for 20,000 agents. An additional benefit of the flow model is that, our flow-based model produces globally optimal task assignments with respect to true shortest-path distances without explicitly computing or storing any pairwise paths, because the minimum-cost flow is solved over the grid itself.

8 CONCLUSION

In this work, we study the task assignment problem in online MAPD. While most prior work in this space focuses on improving multi-agent path finding (MAPF), our primary contribution is to the task assignment component. We propose a spatial flow-based task assignment framework that directly operates on the map and avoids costly pairwise distance computations. The model supports real-time execution, planner integration, and high scalability. We further explore two lightweight congestion-aware edge cost models, demonstrating the flexibility of the flow formulation. These models serve as initial examples of how real-time traffic information can be embedded into flow networks to improve coordination and task distribution. Empirically, our method achieves up to 15% throughput improvement over existing baselines. This improvement is not only statistically significant, but also highly impactful in real-world deployments: many warehouses operate every day with over 10,000 robots, and even a single company may manage dozens of such facilities. In these large-scale, continuously running systems, a 15% improvement can lead to substantial operational savings and higher system responsiveness. Future work includes designing more adaptive edge cost models, improving the flow network structure, and extending the framework to support other MAPD variants such as deadline-aware delivery, energy-constrained agents, and dynamic environments.

[4]RMCA cannot scale to the team size in Experiment 2, hence not included

9 ACKNOWLEDGEMENTS

This work is supported by the Australian Research Council under grant DP200100025, and by a gift from Amazon.

REFERENCES

[1] Aakash and Indranil Saha. 2022. It Costs to Get Costs! A Heuristic-Based Scalable Goal Assignment Algorithm for Multi-Robot Systems. *Proceedings of the International Conference on Automated Planning and Scheduling* 32, 1 (2022), 2–10. https://doi.org/10.1609/icaps.v32i1.19779

[2] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice hall.

[3] Xiaoshan Bai, Ming Cao, Weisheng Yan, and Shuzhi Sam Ge. 2019. Efficient routing for precedence-constrained package delivery for heterogeneous vehicles. *IEEE Transactions on Automation Science and Engineering* 17, 1 (2019), 248–260.

[4] Shao-Hung Chan, Zhe Chen, Teng Guo, Han Zhang, Yue Zhang, Daniel Harabor, Sven Koenig, Cathy Wu, and Jingjin Yu. 2024. The league of robot runners competition: Goals, designs, and implementation. In *ICAPS 2024 System's Demonstration track*.

[5] Zhe Chen, Javier Alonso-Mora, Xiaoshan Bai, Daniel D Harabor, and Peter J Stuckey. 2021. Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. *IEEE Robotics and Automation Letters* 6, 3 (2021), 5816–5823.

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

[7] Balázs Dezső, Alpár Jüttner, and Péter Kovács. 2011. LEMON–an open source C++ graph template library. *Electronic notes in theoretical computer science* 264, 5 (2011), 23–45.

[8] Brian P Gerkey and Maja J Matarić. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research* 23, 9 (2004), 939–954.

[9] Daniel Harabor, Ryan Hechenberger, and Thomas Jahn. 2022. Benchmarks for pathfinding search: Iron harvest. In *Proceedings of the international symposium on combinatorial search*, Vol. 15. 218–222.

[10] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph Durham, and Nora Ayanian. 2018. Conflict-based search with optimal task assignment. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*.

[11] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. 2013. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research* 32, 12 (2013), 1495–1512.

[12] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[13] Michail G Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton J Kleywegt, Sven Koenig, Craig A Tovey, Adam Meyerson, and Sonal Jain. 2005. Auction-Based Multi-Robot Routing.. In *Robotics: Science and Systems*, Vol. 5. Rome, Italy, 343–350.

[14] Edward Lam, Peter J Stuckey, and Daniel Harabor. 2025. Optimal Multi-Agent Pickup and Delivery Using Branch-and-Cut-and-Price Algorithms. *Transportation Science* 59, 1 (2025), 104–124.

[15] Gilbert Laporte. 2009. Fifty years of vehicle routing. *Transportation science* 43, 4 (2009), 408–416.

[16] Jan Karel Lenstra and AHG Rinnooy Kan. 1981. Complexity of vehicle routing and scheduling problems. *Networks* 11, 2 (1981), 221–227.

[17] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. 2019. Task and path planning for multi-agent pickup and delivery. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*.

[18] Hang Ma and Sven Koenig. 2016. Optimal Target Assignment and Path Finding for Teams of Agents *(AAMAS '16)*. International Foundation for Autonomous Agents and Multiagent Systems, 1144–1152.

[19] Hang Ma, Jiaoyang Li, TK Satish Kumar, and Sven Koenig. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. 837–845.

[20] Hang Ma, Jingxing Yang, Liron Cohen, TK Kumar, and Sven Koenig. 2017. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 13. 270–272.

[21] Robert Morris, Corina S Pasareanu, Kasper Søe Luckow, Waqar Malik, Hang Ma, TK Satish Kumar, and Sven Koenig. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations.. In *AAAI Workshop: Planning for Hybrid Systems*. 608–614.

[22] Van Nguyen, Philipp Obermeier, Tran Son, Torsten Schaub, and William Yeoh. 2019. Generalized target assignment and path finding using answer set programming. In *Proceedings of the International Symposium on Combinatorial Search*, Vol. 10. 194–195.

[23] Keisuke Okumura and Xavier Défago. 2023. Solving simultaneous target assignment and path planning efficiently with time-independent execution. *Artificial Intelligence* 321 (2023), 103946.

[24] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence* 310 (2022), 103752.

[25] James B. Orlin. 1993. A Faster Strongly Polynomial Minimum Cost Flow Algorithm. *Operations Research* 41, 2 (1993), 338–350. http://www.jstor.org/stable/171782

[26] Jean-Yves Potvin and Jean-Marc Rousseau. 1993. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research* 66, 3 (1993), 331–340.

[27] Lyle Ramshaw and Robert Endre Tarjan. 2012. On Minimum-Cost Assignments in Unbalanced Bipartite Graphs. https://www.labs.hpe.com/techreports/2012/HPL-2012-40.pdf

[28] Indranil Saha et al. 2025. A Scalable Multi-Robot Goal Assignment Algorithm for Minimizing Mission Time followed by Total Movement Cost. *Artificial Intelligence* (2025), 104388.

[29] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

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

[31] Nathan R Sturtevant. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 2 (2012), 144–148.

[32] Jiri Svancara and Pavel Surynek. 2017. New flow-based heuristic for search algorithms solving multi-agent path finding. In *International Conference on Agents and Artificial Intelligence*, Vol. 2. SCITEPRESS, 451–458.

[33] Robert Endre Tarjan. 1997. Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. *Math. Program.* 77 (1997), 169–177. https://doi.org/10.1007/BF02614369

[34] Manuela Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. 2015. CoBots: robust symbiotic autonomous mobile service robots. In *Proceedings of the 24th International Conference on Artificial Intelligence*. 4423–4429.

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