

Alternating Target–Path Planning for Scalable Multi-Agent Coordination

Yu Kumagai^{1,2}, Keisuke Okumura²

¹Hitotsubashi University, Japan

²National Institute of Advanced Industrial Science and Technology (AIST), Japan
{yu.kumagai,okumura.k}@aist.go.jp

Abstract

The concurrent *target assignment and pathfinding (TAPF)* problem extends multi-agent pathfinding (MAPF) by asking planners to allocate distinct targets and collision-free paths to agents. Prior work on TAPF has relied exclusively on Conflict-Based Search (CBS), which tightly couples target assignment and pathfinding, resulting in compute-intensive, non-scalable solutions. In contrast, we propose an iterative refinement framework that decouples target assignment from pathfinding. Our framework builds on modern, fast, suboptimal MAPF solvers, such as LaCAM. Specifically, within a given time budget, it repeatedly solves MAPF for the current target assignment, identifies bottleneck agents via MAPF feedback, and refines the assignment. Empirical results show that feedback-driven reassignment loop is effective, enabling our framework to scale well beyond the reach of the state-of-the-art CBS-based solver while maintaining decent solution quality. This represents a solid step toward practical, large-scale TAPF suitable for real-world setups.

1 Introduction

The concurrent *target assignment and pathfinding (TAPF)* problem (Ma and Koenig 2016; Barták, Ivanová, and Švancara 2021) for multiple agents on a graph is an appealing extension of the classical multi-agent pathfinding (MAPF) formulation (Stern et al. 2019). In TAPF, each agent is given a set of possible target locations, and a planner must not only generate collision-free paths but also assign distinct targets to each agent. This formulation reflects real-world logistics more faithfully than vanilla MAPF, as many such domains are *partially agent-agnostic*: agent-target pairing is to some extent flexible, as long as all targets are completed.

The target assignment, a subproblem of TAPF, is a well-established field with classical solutions such as the Hungarian algorithm (Kuhn 1955), bottleneck matching (Gross and Corporation 1959), and auction-based approaches (Gerkey and Mataric 2004). MAPF, on the other hand, has seen significant developments in the last decade. With the progress in MAPF research, several TAPF algorithms have been proposed (Ma and Koenig 2016; Hönl et al. 2018; Tang et al. 2023; Tang, Koenig, and Li 2024). However, existing TAPF approaches are exclusively built upon Conflict-Based Search (CBS) (Sharon et al. 2015) and its variants, which guarantee optimal or bounded-suboptimal solutions but suffer from

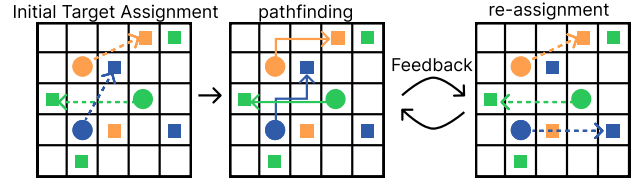


Figure 1: Iterative refinement framework for TAPF.

limited scalability. As a result, current TAPF solutions struggle to handle large instances involving hundreds of agents within practical time constraints. In contrast, recent developments in unbounded suboptimal MAPF algorithms—such as LNS2 (Li et al. 2022), and LaCAM (Okumura 2023)—have demonstrated remarkable scalability. By prioritizing computational efficiency over guaranteed optimality, they solve large instances within a realistic timeframe. This scalability gap motivates a different approach to TAPF.

To this end, we propose an *iterative refinement framework* that decouples target assignment from pathfinding, leveraging modern, fast, suboptimal MAPF solvers as subroutines. As Fig. 1 illustrates, within a given time budget, our framework repeatedly solves MAPF for the current assignment, analyzes the resulting paths to identify bottleneck agents, and uses this feedback to refine the assignment.

While this approach is conceptually straightforward, key design choices remain in (i) *the feedback mechanism*—selecting which assignments to reconsider from the previous path finding—and (ii) *the reassignment strategy*—determining how to reallocate targets based on the feedback. For each perspective, we propose two reasonable methods.

Specifically, for (i) the feedback mechanism, we present *Delay-Based Selection (DBS)*, which identifies bottleneck agents based on their delays from ideal paths, and *Spectral Bottleneck Sampling (SBS)*, which leverages spectral analysis to identify groups of spatially correlated agents experiencing collective delays. For (ii) the reassignment strategy, we present *PIBT*, which adapts PIBT (Okumura et al. 2022) to target reassignment leveraging its efficient collision-free allocation mechanism, and *Local Hungarian*, which applies the Hungarian algorithm to a small subgroup of agents to achieve locally optimal reassignment.

We evaluate these combinations on the commonly-used

MAPF benchmark (Stern et al. 2019) and demonstrate that our framework—especially the combination of DBS and Local Hungarian—scales well beyond the reach of the state-of-the-art CBS-based TAPF solver (Tang, Koenig, and Li 2024) while maintaining competitive solution quality. This moves TAPF meaningfully closer to practical, scalable deployment in real-world applications. The code is available in <https://github.com/Ukuma012/ir-tapf>.

2 Preliminaries

2.1 Problem Definition

We consider a system consisting of a graph $G = (V, E)$ and a set of agents $A = \{1, \dots, n\}$. Let $\mathcal{T}$ be a binary assignment matrix of size $n \times |V|$, where $\mathcal{T}[i][j] = 1$ means that a vertex $v \in V$ of index j can be the target of an agent $i \in A$. For convenience, we will use the notation $\mathcal{T}[i][v]$.

A *feasible assignment* is an injective function, i.e., matching $\mathcal{M} : A \mapsto V$ that satisfies $\mathcal{T}[i][\mathcal{M}(i)] = 1$ for each agent i . A *TAPF instance* is defined by a graph G , agents A , starts $s_i \in V$ for each agent $i \in A$, and a target assignment matrix $\mathcal{T}$ that has at least one feasible assignment. A *solution* to TAPF is a feasible assignment $\mathcal{M}$ and a collection of collision-free paths $\Pi = \{\pi_1, \dots, \pi_n\}$ where π_i is the path for agent i . For each agent i , its path must begin at s_i and end at $\mathcal{M}(i)$. Agent movement models and collisions follow the classical MAPF formulation (Stern et al. 2019): i.e., at each time step, an agent can either move to an adjacent vertex or remain at its current vertex. Two types of conflicts must be avoided: (i) *vertex conflict*, where two agents occupy the same vertex simultaneously, and (ii) *edge conflict*, where two agents traverse the same edge in opposite directions.

In this formulation, the solution cost is assessed by *flow-time* (aka. sum-of-costs), which sums the travel time of each agent until it stops at the target location and no longer moves. Within limited planning time, we are ultimately interested in finding flowtime-minimizing solutions.

We use $\text{dist}(u, v)$ to denote the shortest path distance between vertices u and v on graph G . For a path $\pi = \langle v_0, v_1, \dots, v_T \rangle$, we define $\text{cost}(\pi) = T$ as the path cost (i.e., the travel time).

2.2 Related Work

MAPF. While early work on MAPF focused on developing efficient optimal algorithms such as CBS (Sharon et al. 2015), recent research has increasingly shifted toward suboptimal yet highly scalable, real-time methods. Among these suboptimal approaches, *LaCAM* (*lazy constraints addition search*) (Okumura 2023, 2024) has gained notable attention due to its reliability in delivering solutions quickly even for large numbers of agents operating in densely populated environments. These ultra-fast MAPF methods have opened up novel multi-robot research directions, such as advanced planning-execution frameworks (Zhang et al. 2024a) and MAPF-based multi-robot control in continuous spaces (Shankar, Okumura, and Prorok 2025; Okumura et al. 2026). Our TAPF study also aligns with this direction, leveraging fast MAPF solvers as subroutines to address challenging multi-agent planning problems.

Unlabeled MAPF. While standard MAPF assigns each agent a fixed target, *unlabeled* (or *anonymous*) MAPF treats agents as fully interchangeable: any agent can be assigned to any target. Makespan-optimal unlabeled MAPF is solvable in polynomial time via a flow-network reduction (Yu and LaValle 2013), which jointly optimizes target assignment and pathfinding. However, this approach lacks generality and cannot be applied to TAPF. By contrast, our approach is more closely aligned with suboptimal methods for unlabeled MAPF (Okumura and Défago 2023), which decouple initial target assignment from pathfinding to achieve better real-time planning performance.

TAPF. This setting generalizes both labeled and unlabeled MAPF. It can be viewed as unlabeled MAPF with agent-specific target sets, where the extreme case is labeled MAPF, in which each agent has exactly one designated target. Several TAPF algorithms have been proposed, primarily built upon CBS. For example, CBM (Ma and Koenig 2016), CBS-TA (Hönig et al. 2018), and ITA-CBS (Tang et al. 2023) provide optimal solutions. Their bounded suboptimal versions (Hönig et al. 2018; Tang, Koenig, and Li 2024) have also been proposed, dropping optimality for faster computation. While these CBS-based approaches provide solution quality guarantees, this guarantee comes at the cost of limited scalability. As shown in our experiments (Sec. 6), the state-of-the-art bounded suboptimal algorithm ITA-ECBS (Tang, Koenig, and Li 2024) is unable to solve instances beyond 250 agents within a reasonable time. Beyond CBS, the multi-agent transportation problem casts joint task assignment and pathfinding as a reduction to a sequence of SAT instances (Bachor, Bergdoll, and Nebel 2023), but such reduction-based solving scales only to small instances. This motivates us to develop practical solutions.

3 Iterative Refinement Framework

Our approach to TAPF is straightforward. The framework begins with an (arbitrary) initial target assignment, rapidly solves the corresponding MAPF problem, analyzes the paths to identify problematic assignments as feedback, reassigns targets, and repeats this process until reaching a given time limit. This iterative style has been underexplored, as prior work has focused on tightly coupled target assignment and pathfinding within unified CBS-based framework. Meanwhile, the emergence of ultra-fast MAPF algorithms enables us to quickly estimate the quality of a given target assignment. Using this feedback, we aim to gradually refine the target assignment under a tight time constraint.

This framework, which decouples target assignment and pathfinding, not only enjoys the scalability advantage stemming from modern MAPF solvers, but also offers an engineering benefit of *modularity*. Indeed, the framework accommodates arbitrary target assignment strategies and MAPF solvers, so future advances in either component can be directly integrated without modifying the other.

Algorithm 1 formalizes the framework, detailed below.

Initial Target Assignment. The process at Line 1 could be arbitrary, but it is desirable to be computationally efficient

while assigning each agent to a nearby target, as this provides a strong starting point for subsequent refinement. Although the polynomial-time, optimal Hungarian algorithm exists (Kuhn 1955), its $O(n^3)$ time complexity is impractical for large instances. To this end, we employ a greedy assignment strategy adopted in a scalable unlabeled-MAPF algorithm (Okumura and Défago 2023). This method sorts all feasible agent-target pairs by the shortest-path distance and greedily assigns them in order. It then iteratively improves the assignment through pairwise swaps: if swapping targets between agents i and j reduces the sum of the path lengths and is feasible, the swap is performed. This process repeats until no beneficial swap remains.

Target Refinement Iteration. Each iteration consists of two steps. First, a *feedback mechanism* analyzes the paths from the previous MAPF solution to identify an agent for reassignment (Line 5). Next, based on the identified agent, the framework reassigns targets to a subset of the entire team (Line 6). The quality of the new assignment is then evaluated by solving its corresponding MAPF problem (Line 7). Throughout this process, the framework tracks the best overall solution found so far (Line 9). The framework terminates when interrupted (e.g., by a time limit), returning the best solution found.

Final Path Optimization. After the target refinement terminates, the framework can optionally perform a final path optimization (Line 10). While the target refinement benefits from lightweight MAPF solvers for computational efficiency, the resulting paths are typically suboptimal. Therefore, applying a computationally heavier solver to obtain the final paths can further improve the overall solution quality. In our implementation, vanilla LaCAM is used during the target-refinement stage, while its engineered variant, LaCAM3 (Okumura 2024), which provides better solution quality with a slight increase in computational overhead, is employed in the final step.

Multi-Bottleneck Evaluation. While Alg. 1 presents a naive version that selects a single bottleneck agent per iteration (Lines 5–7; blue-colored), we can enhance the framework by evaluating multiple candidate reassignments for each iteration. Specifically, Alg. 2 provides a drop-in replacement for the corresponding lines in Alg. 1. Suppose that the feedback mechanism returns multiple agents (Line 2 in Alg. 2). The framework then generates an alternative assignment for each. Among these evaluated candidates, we stochastically select one (uniformly at random in the implementation) to continue the refinement. Rather than always choosing the best candidate, this non-determinism avoids search stagnation and encourages broader exploration of the solution space, ultimately yielding better final solutions. In practice, this multi-bottleneck evaluation can be parallelized and therefore does not incur significant time overhead compared to the single-bottleneck version.

4 Target Reassignment

The effectiveness of this framework depends on the method for reassigning targets (Line 6) and the design of the feed-

Algorithm 1: Iterative Refinement Framework for TAPF

input: TAPF instance I
output: task assignment $\mathcal{M}$ and collision-free paths Π

```

1:  $\mathcal{M} \leftarrow \text{InitialAssignment}(I)$ 
2:  $\Pi \leftarrow \text{SolveMAPF}(I, \mathcal{M})$ 
3:  $\mathcal{M}_{\text{best}} \leftarrow \mathcal{M}, c_{\text{best}} \leftarrow \text{GetCost}(\Pi), \Pi_{\text{best}} \leftarrow \Pi$ 
4: while not interrupted do
5:    $a \leftarrow \text{GetBottleneckAgent}(I, \mathcal{M}, \Pi)$ 
6:    $\mathcal{M} \leftarrow \text{ReassignTargets}(a, \mathcal{M})$ 
7:    $\Pi \leftarrow \text{SolveMAPF}(I, \mathcal{M})$ 
8:    $c \leftarrow \text{GetCost}(\Pi)$ 
9:   if  $c < c_{\text{best}}$  then  $\mathcal{M}_{\text{best}} \leftarrow \mathcal{M}, c_{\text{best}} \leftarrow c, \Pi_{\text{best}} \leftarrow \Pi$ 
10:  $\Pi_{\text{best}} \leftarrow \text{SolveNearOptimalMAPF}(I, \mathcal{M}_{\text{best}})$   $\triangleright$  optional
11: return  $\mathcal{M}_{\text{best}}, \Pi_{\text{best}}$ 
```

Algorithm 2: Multi-Bottleneck Evaluation

(replacement for Lines 5–7 of Alg. 1)

```

1: candidates  $\leftarrow \emptyset$ 
2:  $B \leftarrow \text{GetBottleneckAgents}(I, \mathcal{M}, \Pi)$ 
3: for  $a \in B$  do  $\triangleright$  parallelizable
4:    $\mathcal{M}' \leftarrow \text{ReassignTargets}(a, \mathcal{M})$ 
5:    $\Pi' \leftarrow \text{SolveMAPF}(I, \mathcal{M}')$ 
6:   candidates  $\leftarrow$  candidates  $\cup \{(\mathcal{M}', \Pi')\}$ 
7:  $\mathcal{M}, \Pi \leftarrow \text{SelectOneRandomly}(\text{candidates})$ 
```

back mechanism (Line 5). We now describe two approaches for the former: *PIBT* and *Local Hungarian*.

4.1 PIBT

When reassigning a bottleneck agent $a_{\text{bottleneck}}$, a natural difficulty is that its preferred target may already be held by another agent. To handle such conflicts, we adapt the mechanism of Priority Inheritance with Backtracking (PIBT) (Okumura et al. 2022). In its original pathfinding setting, PIBT resolves positional conflicts by having a higher-priority agent recursively displace lower-priority agents occupying desired vertices; displaced agents, in turn, seek alternative vertices through the same recursive process. We apply an analogous mechanism to target reassignment: when the $a_{\text{bottleneck}}$ prefers a target g that is already assigned to another agent, we recursively request it to switch to an alternative target, triggering a chain of reassignments until every displaced agent finds a feasible target or the process fails.

Algorithm 3 embodies this idea. Starting from the bottleneck agent, we enumerate all feasible targets in ascending order of distance from its start position, excluding its current target (Line 3). For each candidate target g , if g is unassigned or the agent currently holding g can be successfully displaced via a recursive call (Line 6), the agent claims g .

Crucially, if the recursive call fails, the algorithm does not abort but instead backtracks and proceeds to the next candidate in the sorted list. This backtracking mechanism explores a richer set of feasible reassignments compared to a greedy single-candidate approach.

Time Complexity. The time complexity closely follows that of the original PIBT. Let K denote the maximum number of feasible targets per agent and F the time to evaluate

a candidate targets. The worst-case complexity is $O(|A| \cdot K \cdot F)$, as the displacement may involve up to $|A|$ agents. In practice, it terminates after far fewer than $|A|$ displacements.

4.2 Local Hungarian Assignment

Where targets are densely clustered, the reassignment of one agent significantly affects the optimal choices for others. To handle such cases, we propose *Local Hungarian* assignment, which jointly optimizes a small subgroup of agents rather than reassigning them individually. Specifically, given a set of agents identified by the feedback mechanism, we treat this entire set as a subgroup. For this subgroup, we collect a pool of candidate targets consisting of their current targets and unassigned targets that are not currently assigned to any agent outside subgroup. We then construct a cost matrix where entry (i, v) represents the shortest path distance from agent i 's start position s_i to the candidate target $v \in V$. The Hungarian algorithm (Kuhn 1955) is applied to this cost matrix to find the minimum linear-sum cost assignment of the subgroup, while ignoring inter-agent collisions.

Instead of re-optimizing the assignment for all agents, this local optimization strategy focuses the optimization effort on the most critical subset of the entire team. Consequently, it resolves the identified bottleneck assignment effectively.

5 Feedback Mechanism

The second key component is the feedback mechanism. A straightforward approach would randomly select an agent for reassignment, but it would ignore the valuable information from the pathfinding results. Instead, we should systematically identify the agent that struggled most, as this agent is likely the cost bottleneck. We propose two feedback mechanisms, called *Delay-Based Selection (DBS)* and *Spectral Bottleneck Sampling (SBS)*.

5.1 Delay-Based Selection (DBS)

DBS, the simplest yet effective idea, identifies an agent to reassign by ranking all agents by their *delay*—how much their actual path cost exceeds the ideal shortest-path cost. Specifically, for each agent i , DBS computes the delay as $d_i = \text{cost}(\pi_i) - \text{dist}(s_i, \mathcal{M}(i))$. The higher d_i means that i 's path has greater room for improvement. Such a delay-based score is commonly used in large neighborhood search refinement for MAPF (Li et al. 2021; Okumura, Tamura, and Défago 2021); here we employ it for target reassignment.

Based on this delay measure, DBS selects the top- m agents ($m \ll n$; set to 10 in our experiments), and then randomly chooses k of them as the current bottlenecks. With the Local Hungarian assignment, we observed that applying these k agents directly within a single reassignment optimization yields better performance; thus, the multi-bottleneck mechanism is disabled with Local Hungarian.

5.2 Spectral Bottleneck Sampling (SBS)

As an advanced method, SBS leverages spectral decomposition to identify groups of spatially correlated agents that exhibit collective delays, through two stages: (i) constructing a *weighted performance discrepancy matrix* D that captures

Algorithm 3: PIBT-style Displacement

input: bottleneck agent $a_{\text{bottleneck}}$, assignment $\mathcal{M}$
output: new assignment $\mathcal{M}$

```

1: return PIBT( $a_{\text{bottleneck}}$ )
2: function PIBT( $a$ )
3:    $C \leftarrow [v \mid \mathcal{T}[a][v] = 1, v \neq \mathcal{M}(a)]$  sorted by  $\text{dist}(s_a, v)$ 
4:   for  $g \in C$  do
5:      $a' \leftarrow b \in A$  s.t.  $\mathcal{M}(b) = g$ 
6:     if  $\nexists a'$  or PIBT( $a'$ )  $\neq$  FAILURE then
7:        $\mathcal{M}(a) \leftarrow g$ ; return  $\mathcal{M}$ 
8:   return FAILURE

```

Algorithm 4: Spectral Bottleneck Sampling (SBS)

input: Instance I , assignment $\mathcal{M}$, solutions Π
output: bottleneck agent $a_{\text{bottleneck}}$

```

1:  $M \leftarrow \text{ComputePotentialConflictMatrix}(I, \mathcal{M})$ 
2:  $D \leftarrow$  zero matrix of size  $n \times n$ 
3: for  $i \leftarrow 1, \dots, n$ ;  $j \leftarrow i + 1, \dots, n$  do
4:    $D[i][j], D[j][i] \leftarrow M[i][j] \times (d_i + d_j)$ 
5:  $\lambda, V \leftarrow \text{SpectralDecomposition}(D)$ 
6:  $a_{\text{bottleneck}} \leftarrow \arg \max_i |V[1][i]|$ 
7: return  $a_{\text{bottleneck}}$ 

```

both spatial correlation and performance degradation among agent pairs, and (ii) applying *spectral decomposition* to reveal groups of agents experiencing correlated bottlenecks. Spectral decomposition is effective because eigenvectors associated with large eigenvalues tend to highlight clusters of agents that mutually interfere in the same congested regions, enabling us to identify more structural bottlenecks. Algorithm 4 shows its pseudocode.

Weighted Performance Discrepancy Matrix, D , aims to capture the group interaction among agents, constructed with two steps (Fig. 2): (i) A *potential conflict matrix* M is computed, where $M[i][j]$ counts the number of vertex and edge conflicts between agents i and j 's ideal shortest paths (Line 1). This quantifies spatial correlation: high $M[i][j]$ indicates spatiotemporal overlaps between agents i and j . Indeed, such a score is often employed in recent MAPF methods to mitigate congestion (Zhang et al. 2024b; Kato et al. 2025; Arita and Okumura 2026); (ii) We next construct the matrix D by combining spatial correlation with performance degradation: $D[i][j] = M[i][j] \times (d_i + d_j)$, where d_i is delay as defined in DBS. From its construction, $D[i][j]$ is high when i and j are both spatially correlated *and* experiencing delays. Note that the matrix symmetry (i.e., $D[i][j] = D[j][i]$) makes it suitable for spectral analysis.

Spectral Decomposition. We next apply spectral decomposition to the obtained matrix D (Line 5), resulting in $D = V\Lambda V^T$ where Λ contains eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$, and $V = [v_1, v_2, \dots, v_n]$ contains corresponding eigenvectors. The eigenvalue λ_k represents the strength of the k -th principal mode of correlated discrepancy, while the eigenvector v_k identifies which agents participate in this mode. Agents with high absolute values in v_k form a group, capturing those that exhibit correlated delays.

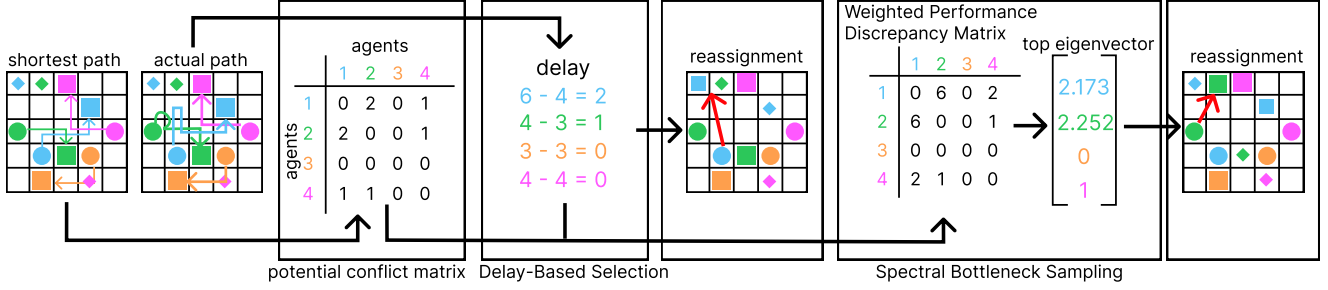


Figure 2: Example of DBS and SBS. Circles represent agents, and large squares represent their currently assigned targets. On the left grids, the top left small diamonds show the other assignable targets for agents 1 and 2.

Bottleneck Selection. For the single-bottleneck selection, we choose the agent with the largest absolute value in the first eigenvector $V[1]$, which corresponds to the largest eigenvalue λ_1 . For the multi-bottleneck case, we consider two styles: (i) selecting k eigenvector groups from the top-ranked groups and picking one agent with the highest absolute eigenvector value from each group; or (ii) using the single highest eigenvector and selecting the top- k agents with the highest absolute eigenvector values within that group. Empirically, the former works well with PIBT, whereas the latter is more effective with Local Hungarian.

Time Complexity. SBS has $O(n^2)$ overhead for constructing the matrix D (Lines 1–4) and $O(n^3)$ for eigenvalue decomposition (Line 5). Meanwhile, D exhibits high sparsity (90-98% in our experiments) due to two factors: (i) spatially distant agents have zero potential conflicts ($M[i][j] = 0$), and (ii) agents that take ideal path contribute zero performance delays ($d_i = 0$). Therefore, we can use iterative sparse eigenvalue solvers, such as the Lanczos method (Lanczos 1950). For finding the largest eigenvalue and its corresponding eigenvector, this reduces complexity from $O(n^3)$ to $O(zm)$ where $z \ll n^2$ is the number of non-zero elements and m is the hyperparameter iteration count. Implementation-wise, we further employ sub-sampling of the entire team, reducing the matrix size from $n \times n$ to $s \times s$ where $s \ll n$ ($s = 100$ in the experiments). The sub-sampling procedure first collects agents whose start positions are close to the bottleneck agents identified in the previous round, based on the Manhattan distance, and then randomly selects the remaining agents. This approach preserves scalability while still providing sufficiently informative feedback for the assignment process.

6 Evaluation

This section evaluates the proposed TAPF framework. The experiments consist of four parts: (i) identifying effective combinations of reassignment and feedback strategies; (ii) contrasting our framework against the state-of-the-art TAPF solver; (iii) assessing its scalability by large instances with up to 10,000 agents; and (iv) validating the effect of the multi-bottleneck method.

Setup. Experiments were conducted on a MacBook Pro with an Apple M1 Chip, equipped with 8-core CPUs and 8 GB RAM. The proposed methods are coded in Rust. We benchmark TAPF methods on four-connected grid maps from the MAPF benchmark (Stern et al. 2019), containing diverse environments with varying obstacle densities and structural characteristics. Using these maps, we generate TAPF instances with an assignment matrix $\mathcal{T}$, constructed under one of the following two scenarios:

- **RANDOM:** Each agent is assigned ten targets, which are randomly selected from V but placed at a moderate distance from its start position, following the random target placement used in prior TAPF studies (Tang, Koenig, and Li 2024; Tang et al. 2023).
- **HOTSPOT:** Targets for all agents are concentrated in a single densely packed region. Within this hotspot, each agent is assigned ten targets and creating about 80% target overlap.

The former scenario represents sparse, easier-to-solve cases, whereas the latter requires careful target assignment.

By default, the framework uses the multi-bottleneck evaluation method (Alg. 2). For each refinement, it identifies three agents as bottlenecks, whose values were adjusted prior to the experiments to strike a balance between exploration and computational efficiency. It then computes reassignments and solves the corresponding MAPF with multi-threading. We employ LaCAM for MAPF solving due to its real-time responsiveness. As a solution metric, we use the “cost” as the flowtime normalized by the sum of the shortest-path lengths between each agent’s start and its nearest target, serving as a sub-optimality upper bound (minimum: 1.0).

6.1 Framework Design

The key components of our framework are the feedback mechanism and the target reassignment strategy. The former can be either DBS or SBS, whereas the latter can use either PIBT or Local Hungarian. We evaluate six combinations of these components, along with a baseline ‘Random’ strategy that selects an agent uniformly at random, without feedback. For example, ‘DBS-Hungarian’ denotes the configuration that uses DBS for bottleneck identification and Local Hungarian for target reassignment. The time limit was

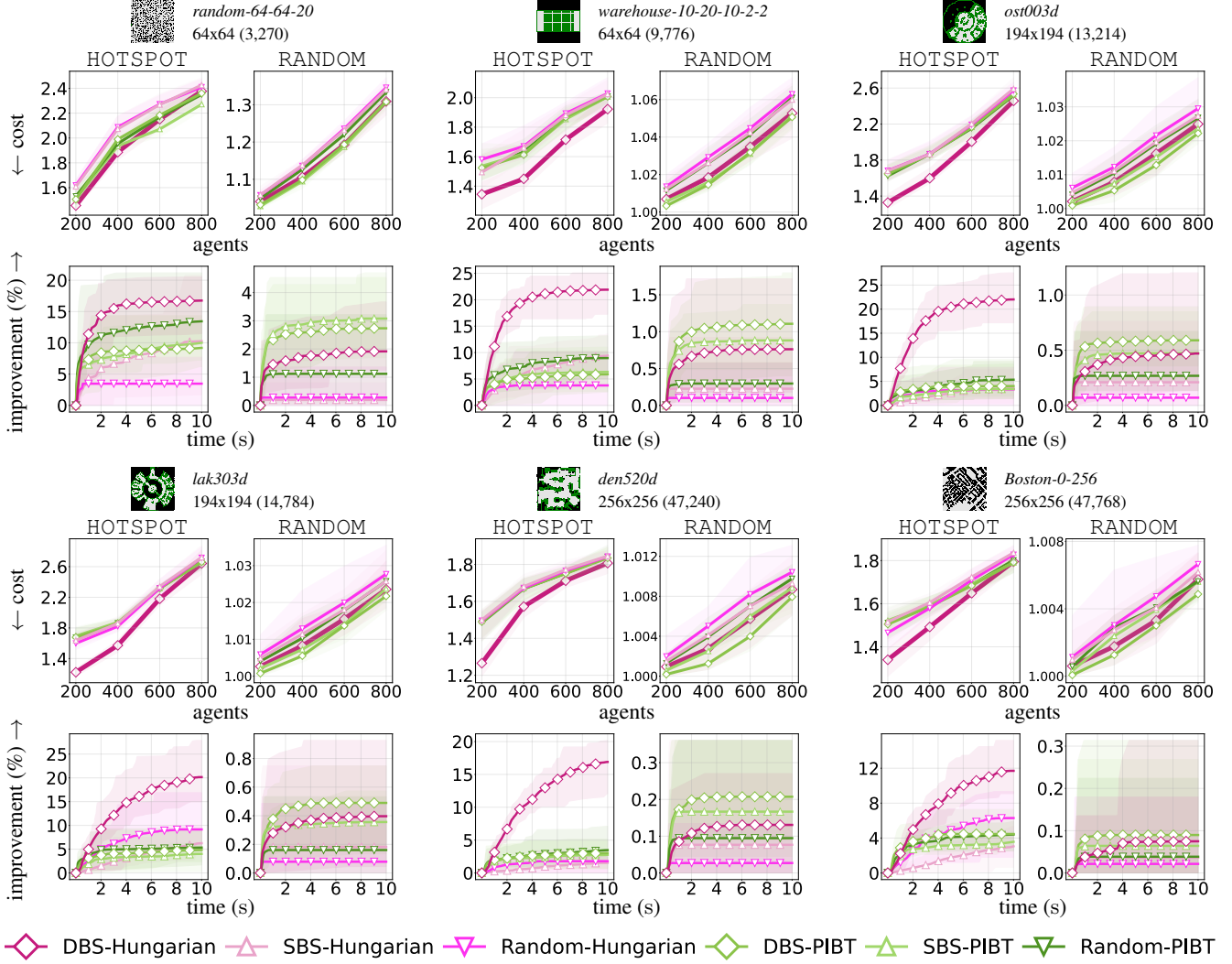


Figure 3: Performance of the proposed TAPF framework using different components. The number of vertices for each grid is shown in parentheses. We show the normalized flowtime (median scores), accompanied by minimum and maximum scores using semi-transparent regions. We further show the improvement rate from the initial solution over time for 200-agent instances.

set to 10 s and 30 instances were prepared for each setting. This experiment omits the final path optimization step to focus on the efficacy of different combinations.

Results. Figure 3 summarizes the results. We first observe that the RANDOM scenario is easier than HOTSPOT, as spatially distributed targets lead to fewer path intersections (see scales of the y-axes). Consequently, RANDOM yields near-optimal initial solutions and leaves only a small margin for improvement. Nevertheless, DBS-PIBT consistently achieves the best costs in this scenario; even when the overall margin is small, its ability to incrementally improve individual agent assignments accumulates into meaningful gains. In contrast, HOTSPOT offers more challenging instances where target reassignment has a significant impact. In either scenario, the initial solutions were obtained rapidly without any failures: e.g., 705 ms for 800-agent cases in

the worst. In HOTSPOT scenario, DBS-Hungarian generally achieves the best scores. For example, in the HOTSPOT results for the 200-agent cases, DBS-Hungarian consistently achieves a 10–25% improvement over the initial solution, demonstrating superior real-time performance compared to the other methods. The time profiling of pathfinding and target reassignment is provided in the appendix; regardless of the combination, the pathfinding component dominates the computational cost.

For further investigation, Tables 1 and 2 provide detailed statistics for the six combinations on *random-64-64-20* with 200 agents in HOTSPOT. Table 1 fixes the time budget, whereas Table 2 fixes the number of reassignment iterations. Focusing on PIBT, Random-PIBT achieves the highest number of iterations (Table 1) and the fastest per-iteration speed (Table 2), while also delivering the best improvement rate

Reassign	Feedback	Iter		Imprv (%)	
		Mean	Min–Max	Mean	Min–Max
Hungarian	DBS	855.3	656–980	17.5	12.3–21.6
	SBS	833.4	741–887	12.2	4.7–21.0
	Random	1901.0	1654–2032	3.6	0.0–7.7
PIBT	DBS	695.1	654–730	9.4	6.2–14.3
	SBS	773.5	707–824	10.9	7.7–13.5
	Random	969.1	833–1060	14.5	10.1–21.2

Table 1: Framework performance with different components on *random-64-64-20* with 200 agents within 10s in HOTSPOT. *Iter*: Number of successful target reassignments. *Imprv*: Improvement rate from the initial solution.

Reassign	Feedback	Time (ms)		Imprv (%)	
		Mean	Min–Max	Mean	Min–Max
Hungarian	DBS	2026	1847–2297	17.4	10.9–22.7
	SBS	2125	1943–2428	12.2	0.0–15.9
	Random	934	883–1044	4.8	0.0–9.8
PIBT	DBS	2839	2705–3203	8.6	4.0–14.3
	SBS	2754	2517–3249	7.5	5.2–12.5
	Random	2016	1803–2466	10.7	6.3–19.1

Table 2: Fixing the number of iterations to 100, using the same setting of Table 1. ‘Time’ taken to reach the iteration limit is shown.

in both settings. This is because PIBT greedily claims the nearest unassigned target from the starting agent, so the resulting chain of reassignments is essentially determined by where that agent is located. In HOTSPOT, DBS/SBS tend to pick agents from the same congested region, so the resulting chains explore similar spatial neighborhoods and narrow the overall exploration breadth; random selection instead scatters the starting agents and covers a wider target-combination space. Turning to Hungarian-based methods, they achieve a better solution quality. Random-Hungarian achieves the highest number of iterations due to its simplicity, while its improvement remains lower than the other two. Meanwhile, both DBS-Hungarian and SBS-Hungarian achieve low iteration counts but deliver substantial improvements, although DBS generally outperforms SBS in both per-iteration speed and improvement efficacy. This is somewhat surprising, given the expectation that SBS would capture more nuanced bottleneck agents than DBS. Based on these observations, we use DBS-Hungarian in the following experiments.

Sensitivity to Initial Target Assignment. Before moving on, we investigate how the initial target assignment (Line 1 in Alg. 1) affects the outcomes. Using the same experimental setting as in Table 1, Table 3 presents the empirical results obtained when a completely random initial target assignment is used. The results demonstrate that the choice of initial assignment has a substantial impact on the final solution quality: all Cost Gap values exceed 1.0, indicating that

Reassign	Feedback	Cost Gap	Iter		Imprv (%)	
			Mean	Mean	Min–Max	
Hungarian	DBS	1.26	447.0	7.4	4.5–14.1	
	SBS	1.03	463.3	19.0	12.7–26.3	
	Random	1.06	1081.3	8.5	1.8–13.2	
PIBT	DBS	1.09	351.6	8.7	4.6–10.8	
	SBS	1.10	375.4	9.4	5.1–12.9	
	Random	1.06	518.9	15.5	11.4–21.9	

Table 3: Framework performance using random initial assignment. The setting follows Table 1. ‘Cost Gap’ shows the final solution costs, divided by those obtained with the greedy initial assignment (Table 1). All values exceed 1.0, meaning that no algorithm achieves the same quality under random initial assignment.

the random initial assignment consistently yields inferior final solutions compared to the greedy initial assignment, despite the subsequent iterative improvement. The relative performance of the framework’s components differs from that in Table 1, where SBS-Hungarian achieves the greatest improvements. We conjecture that the random assignment makes the corresponding MAPF problem harder by introducing many path intersections among the agents’ shortest paths. As a result, SBS is able to identify more effective bottleneck sets than DBS. Conversely, the strong performance of DBS-Hungarian observed so far also depends on having an effective initial assignment.

6.2 Final Path Optimization

The preceding experiments omitted the path optimization step to isolate the efficacy of each component combination. Figure 4 shows that the path optimization consistently improves solution quality across all method combinations, though the magnitude varies sharply: Hungarian-based methods benefit substantially, while PIBT-based methods see only modest gains. *No Target Refinement*, which allocates the full time budget to LaCAM3’s anytime path optimization, also improves the initial solution but plateaus above every combination that includes target refinement. This confirms that solving TAPF efficiently requires the interplay of target refinement and path optimization, not path planning alone.

6.3 Comparison with ITA-ECBS

We compare DBS-Hungarian with ITA-ECBS (Tang, Koenig, and Li 2024), a leading TAPF solver. Both methods are given a time limit of 30 s. For DBS-Hungarian, we allocated 20 s for iterative refinement and 10 s for the final path optimization using LaCAM3’s anytime search (Okumura 2024).

Table 4 highlights the scalability advantage of DBS-Hungarian. DBS-Hungarian achieves a 100% success rate across all settings, whereas ITA-ECBS fails on the majority of instances once the number of agents exceeds 20 in HOTSPOT and roughly 100 in RANDOM, as the CBS conflict tree grows exponentially with agent count. As shown in

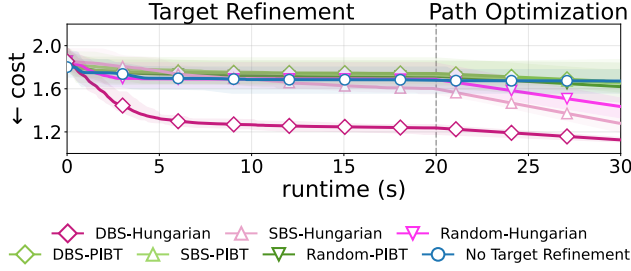


Figure 4: Framework performance on *lak303d* with 200 agents in *HOTSPOT*, using 20s target refinement followed by 10s final path optimization. *No Target Refinement* skips target reassignments and allocates the full time for path optimization.

map	scenario	A	success rate(%)		cost	
			ECBS	D-H	ECBS	D-H
<i>random-32-32-20</i>	<i>HOTSPOT</i>	10	100	100	1.015	1.055
		20	20	100	1.033	1.045
	<i>RANDOM</i>	50	50	100	1.003	1.029
		100	50	100	1.003	1.029
<i>den312d</i>	<i>HOTSPOT</i>	10	70	100	1.004	1.027
		20	30	100	1.011	1.089
	<i>RANDOM</i>	50	93	100	1.001	1.005
		100	40	100	1.001	1.011
		150	13	100	1.001	1.018
<i>warehouse-10-20-10-2-1</i>	<i>HOTSPOT</i>	10	100	100	1.003	1.029
		20	90	100	1.002	1.036
		30	60	100	1.003	1.043
		50	100	100	1.001	1.005
	<i>RANDOM</i>	100	87	100	1.002	1.007
		150	13	100	1.001	1.014
		200	3	100	1.002	1.019

Table 4: Comparison of ITA-ECBS (ECBS) and our proposed DBS-Hungarian (D-H). All scores are averages over solved instances by both. For instance preparation, we gradually increased the number of agents until ITA-ECBS failed to solve all instances among 30 instances generated.

Fig. 3, DBS-Hungarian scales to 800 agents in *HOTSPOT*, far beyond the reach of ITA-ECBS. Moreover, while DBS-Hungarian yields slightly higher costs than ITA-ECBS, the gap is small, indicating competitive solution quality.

6.4 Scalability Test

We evaluate our framework with up to 10,000 agents on *warehouse-20-40-10-2-2*, with a 10 min time limit given the problem scales. Figure 5 shows the time for finding initial solutions and the improvement rate over time. Our method finds initial solutions around 20s even for 10,000 agents. Although the improvement becomes more modest as the number of agents increases, the framework consistently reduces the cost, demonstrating its ability to handle massive-scale TAPF instances while continuing to refine solutions.

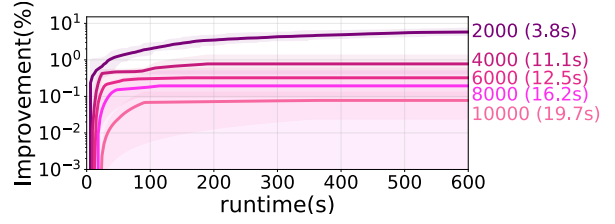


Figure 5: Scalability test. On the right, #agents and the time for finding the initial solutions (in parentheses) are shown.

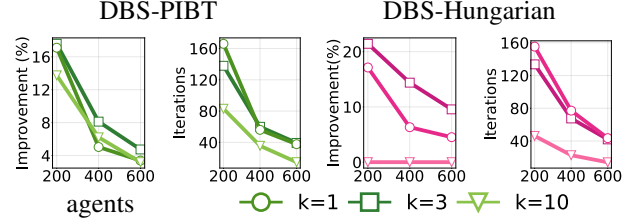


Figure 6: Effect of the number of agents for target reassignment (k) on DBS-{PIBT, Hungarian} using *ost003d*. Improvement rates from the initial solutions and the number of iterations completed within 10s are shown.

6.5 Effect of Multi-Bottleneck Selection

Finally, we investigate the effect of the number of bottleneck agents evaluated per iteration, herein denoted as $k \in \{1, 3, 10\}$. $k = 1$ corresponds to the vanilla framework (Alg. 1), while the other corresponds to Alg. 2. We fix the bottleneck identification method to DBS and assess the effect of k using both the PIBT and Local Hungarian.

PIBT. Figure 6 (left) shows that $k = 3$ achieves the best improvement across all scenarios. $k = 10$ completes significantly fewer iterations due to computational overhead, resulting in smaller improvements. $k = 1$ performs comparably to $k = 3$ up to 200 agents but degrades beyond that due to insufficient exploration diversity.

Local Hungarian. Figure 6 (right) again shows that $k = 3$ as the best choice. $k = 10$ suffers from high per-iteration cost, limiting the total number of iterations, while $k = 1$ enables many iterations but each reassignment is too narrow to yield substantial gains.

7 Conclusion

We introduced an iterative refinement framework for TAPF that decouples target assignment and pathfinding, achieving scalability advantages over the leading TAPF solver while maintaining competitive solution quality. Within this framework, we presented two target reassignment methods (PIBT and Local Hungarian) and two feedback mechanisms (DBS and SBS). Our experiments show that combining DBS and Local Hungarian generally the most effective.

For future work, an interesting direction is to adaptively select the mechanism and reassignment strategy during the refinement loop, akin to the adaptive operator selection in LNS (Li et al. 2021).

Acknowledgments

This research was partially supported by JSPS KAKENHI Grant Number 25K21289, JST ACT-X (JPMJAX22A1), and JST PRESTO (JPMJPR2513).

References

- Arita, T.; and Okumura, K. 2026. Local Guidance for Configuration-Based Multi-Agent Pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.
- Bachor, P.; Bergdoll, R.-D.; and Nebel, B. 2023. The Multi-Agent Transportation Problem. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.
- Barták, R.; Ivanová, M.; and Švancara, J. 2021. From Classical to Colored Multi-Agent Path Finding. In *Proceedings of Annual Symposium on Combinatorial Search (SoCS)*.
- Gerkey, B. P.; and Mataric, M. J. 2004. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *International Journal of Robotics Research (IJRR)*.
- Gross, O.; and Corporation, R. 1959. *The Bottleneck Assignment Problem*. Rand.
- Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2018. Conflict-Based Search with Optimal Task Assignment. In *Proceedings of International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*.
- Kato, T.; Okumura, K.; Sasaki, Y.; and Yokomachi, N. 2025. Congestion Mitigation Path Planning for Large-Scale Multi-Agent Navigation in Dense Environments. *IEEE Robotics and Automation Letters (RA-L)*.
- Kuhn, H. W. 1955. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*.
- Lanczos, C. 1950. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand. B*.
- Li, J.; Chen, Z.; Harabor, D.; Stuckey, P.; and Koenig, S. 2021. Anytime multi-agent path finding via large neighborhood search. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.
- Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.
- Ma, H.; and Koenig, S. 2016. Optimal Target Assignment and Path Finding for Teams of Agents. In *Proceedings of International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*.
- Okumura, K. 2023. LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.
- Okumura, K. 2024. Engineering LaCAM*: Towards Real-Time, Large-Scale, and Near-Optimal Multi-Agent Pathfinding. In *Proceedings of International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*.
- Okumura, K.; and Défago, X. 2023. Solving Simultaneous Target Assignment and Path Planning Efficiently with Time-Independent Execution. *Artificial Intelligence (AIJ)*.
- Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. *Artificial Intelligence (AIJ)*.
- Okumura, K.; Tamura, Y.; and Défago, X. 2021. Iterative Refinement for Real-Time Multi-Robot Path Planning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Okumura, K.; Yang, G.; Gao, Z.; Woo, H.; and Prorok, A. 2026. Concrete multi-agent path planning enabling kinodynamically aggressive maneuvers. *npj Robotics*.
- Shankar, A.; Okumura, K.; and Prorok, A. 2025. LF: Online Multi-Robot Path Planning Meets Optimal Trajectory Control. *arXiv preprint arXiv:2507.11464*.
- Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence (AIJ)*.
- Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of Annual Symposium on Combinatorial Search (SoCS)*.
- Tang, Y.; Koenig, S.; and Li, J. 2024. ITA-ECBS: A Bounded-Suboptimal Algorithm for the Combined Target-Assignment and Path-Finding Problem. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.
- Tang, Y.; Ren, Z.; Li, J.; and Sycara, K. 2023. Solving Multi-Agent Target Assignment and Path Finding with a Single Constraint Tree. In *International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*.
- Yu, J.; and LaValle, S. M. 2013. Multi-agent path planning and network flow. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*.
- Zhang, Y.; Chen, Z.; Harabor, D.; Le Bodic, P.; and Stuckey, P. J. 2024a. Planning and execution in multi-agent path finding: Models and algorithms. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*.
- Zhang, Y.; Jiang, H.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2024b. Guidance Graph Optimization for Lifelong Multi-Agent Path Finding. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.