ITA-LaCAM: A Complete and Scalable TAPF Solver via Assignment-Aware Configuration-Space Search

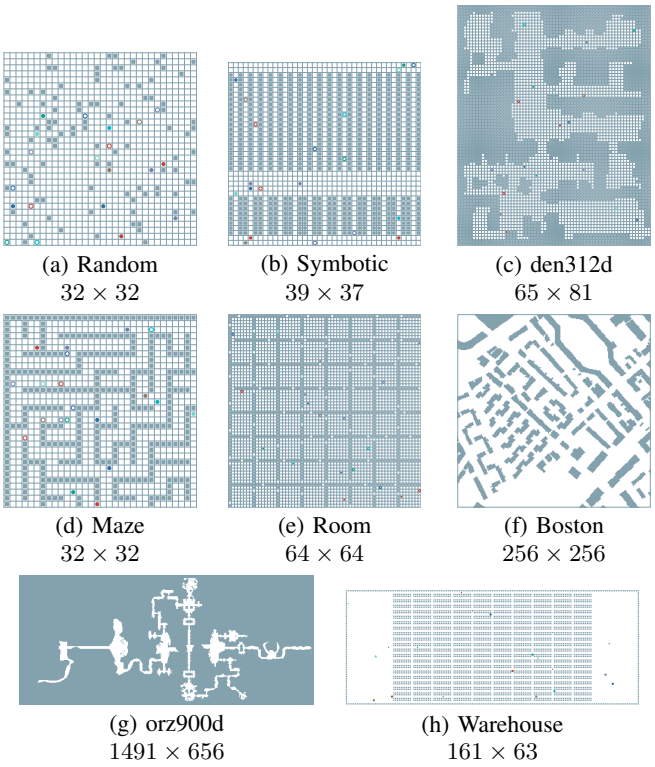
Yimin Tang[1,3], Han Zhang[3], Shao-Hung Chan[3],
Junsoo Kim[1], Erdem Bıyık[1], Sven Koenig[2], and Jingkai Chen[3]

***Abstract*— Combined Target Assignment and Path Finding (TAPF) requires assigning targets for agents while simultaneously planning collision-free paths. We present ITA-LaCAM, a complete and scalable TAPF solver inspired by LaCAM and ITA-CBS. In ITA-LaCAM, each joint-configuration node carries an agent-to-target matching. When a successor is generated, ITA-LaCAM incrementally repairs the matching for the agents that moved and uses the targets to guide PIBT successor generation. This design enables adaptive reassignment without explicitly enumerating the combinatorial assignment space, while preserving LaCAM's completeness and scalability. Across 9,760 benchmark instances on eight maps with 5–200 agents, ITA-LaCAM solved 100% of the instances, compared with 95.6% for IR-TAPF configured with DBS-Hungarian. ITA-LaCAM found an initial solution faster in 84.0% of the comparisons and achieved a lower sum of costs in 65.0% of the instances solved by both methods.**

I. Introduction

Multi-Agent Path Finding (MAPF) plans collision-free paths from agents' starts to distinct preassigned goals and models coordination in domains such as warehouse automation. Because optimal MAPF is NP-hard [1], optimal solvers such as Conflict-Based Search (CBS) [2] and M^* [3] face scalability challenges, while suboptimal methods such as PBS [4], LaCAM [5], and MAPF-LNS2 [6] trade optimality for speed. Many applications instead allow interchangeable targets; warehouse robots, for example, may choose among several storage locations. Combined Target-Assignment and Path-Finding (TAPF) [7] therefore jointly assigns each agent an eligible distinct target and plans collision-free paths, with MAPF as the fixed-assignment special case. Optimal and bounded-suboptimal TAPF solvers, including CBM [7], CBS-TA, ECBS-TA [8], ITA-CBS [9], offer cost guarantees but retain costly CBS-style conflict resolution on large, dense instances.

IR-TAPF [10] improves scalability through iterative refinement, but fixes an assignment while LaCAM solves the induced MAPF problem; feedback arrives only after a complete solution, so an unfavorable assignment may cause substantial delay. To shorten this loop, we propose ITA-LaCAM, inspired by ITA-CBS and built on the complete joint-configuration solver LaCAM* [11], which uses Priority Inheritance with Backtracking (PIBT) [12] for lazy successor generation. Each high-level node stores a minimum-distance agent–target matching that guides PIBT and the heuristic. Every successor repairs only the matching rows of moved agents, updating assignments online without enumerating the assignment space. Our contributions are as follows: (1) We introduce ITA-LaCAM, a fast and complete solver for TAPF that integrates target assignment directly into configuration search. (2) We show empirically that ITA-LaCAM finds first solutions faster than IR-TAPF in 84.0% of cases and lower SOC in 65.0%, while 90% of its paired solutions are within 10.2% of the ITA-CBS optimum.



(a) Random 32 × 32 (b) Symbotic 39 × 37 (c) den312d 65 × 81
(d) Maze 32 × 32 (e) Room 64 × 64 (f) Boston 256 × 256
(g) orz900d 1491 × 656 (h) Warehouse 161 × 63

Fig. 1. The eight benchmark maps used in our experiments.

II. Problem Definition

Combined Target-Assignment and Path-Finding (TAPF) is defined on an undirected graph $G = (V, E)$ with unit-time move and wait actions. Agents $I = \{1, \dots, N\}$ start at vertices s_i, and targets $\mathcal{G} = \{g_1, \dots, g_M\} \subseteq V$ satisfy $M \geq N$. A binary target matrix $A \in \{0,1\}^{N \times M}$ specifies eligibility: $A[i][j] = 1$ iff agent i may use target g_j. A feasible assignment $\tau : I \to \mathcal{G}$ is injective and respects A.

[1]Thomas Lord Department of Computer Science, University of Southern California, Los Angeles, CA, USA. {yimintan,junsooki,biyik}@usc.edu
[2]University of California, Irvine, CA, USA. svenk@uci.edu
[3]Symbotic, Wilmington, MA, USA. {hanzhang,shchan,jichen}@symbotic.com

Given τ, each agent follows a path $\pi_i = [v_0^i, \dots, v_{T^i}^i]$ from s_i to $\tau(i)$ along valid edges or waits and remains at its target after arrival. Paths must avoid vertex collisions ($v_t^i = v_t^j$) and opposing-edge collisions ($v_t^i = v_{t+1}^j$ and $v_{t+1}^i = v_t^j$) for all $i \neq j$ and t. The objective is to find a feasible assignment and collision-free paths minimizing the sum of costs $\mathrm{SOC} = \sum_{i=1}^{N} T^i$.

III. Method

This section presents ITA-LaCAM, which embeds target assignment into LaCAM*'s configuration search for TAPF. We motivate the configuration-space formulation, explain how node-wise assignments guide the search, and describe incremental Hungarian updates within the high-level algorithm.

A. Configuration-Space Motivation

We begin with LaCAM*, summarized in Algorithm 1; without the bold operations, it is the original MAPF algorithm. LaCAM* performs an A*-like search over joint configurations $\mathcal{Q} = (v_1, \dots, v_N)$, where $v_i \in V$ is agent i's location. The initial configuration is $\mathcal{S} = (s_1, \dots, s_N)$, and the fixed MAPF goal is $\mathcal{Q}_{\mathrm{goal}} = (g_1, \dots, g_N)$. A joint configuration $\mathcal{Q}$ is valid if every agent occupies a traversable vertex and no two agents occupy the same vertex, i.e., $v_i \neq v_j$ for all $i \neq j$. Two valid configurations $\mathcal{Q} = (v_1, \dots, v_N)$ and $\mathcal{Q}' = (v'_1, \dots, v'_N)$ are connected by a valid transition if every agent either waits in place or moves to an adjacent vertex, i.e., $v'_i = v_i$ or $(v_i, v'_i) \in E$, and no pair of agents swaps locations, i.e., there do not exist agents $i \neq j$ such that $v_i = v'_j$ and $v_j = v'_i$.

These configurations and transitions induce a valid configuration graph whose edges represent collision-free simultaneous actions. Every collision-free MAPF solution corresponds to a path from $\mathcal{S}$ to $\mathcal{Q}_{\mathrm{goal}}$ in this graph, and every such path is a collision-free solution. This configuration-space perspective supports different planning problems by changing the goal set, transition costs, or task information associated with each state. MAPF has one pre-assigned target per agent, whereas TAPF admits multiple goal configurations because its assignment is not fixed in advance. MAPD and lifelong MAPF can similarly use the same configuration space while successively reaching multiple goals.

We next consider how to explore this graph. CBS first plans individual paths and resolve interactions after detecting collisions. Its intermediate states may therefore combine paths containing vertex or edge collisions and need not correspond to valid paths in the configuration graph. It reasons in a relaxed space containing both the valid graph and additional invalid configurations or transitions. This larger space can be costly in large, congested instances where many conflicts must be detected and resolved. Prioritized Planning and PBS take the opposite approach. They impose priority relations and plan lower-priority agents while treating higher-priority agents as moving obstacles, substantially reducing the search space. However, this restricts search to a subset of the valid graph. A solution may require interactions or precedence relations inconsistent with the selected ordering, so these methods can fail even when the full graph contains a valid path. Applying A* to joint configurations searches exactly the valid graph but suffers from the curse of dimensionality. If each of N agents has five actions, including waiting, one configuration may have up to 5^N successors. Explicitly generating and evaluating all such joint actions at every node

Algorithm 1 ITA-LaCAM. $\mathcal{C}_{\mathrm{init}}$ denotes "no constraint."

Input: **TAPF** instance, transition cost cost_e, and heuristic h
Output: Solution, `NO_SOLUTION`, or `FAILURE`
Notation: $\mathsf{f}(\mathcal{N}) := \mathcal{N}.g + \mathsf{h}(\mathcal{N})$; $\spadesuit := (\mathcal{N}^{\mathrm{goal}} \neq \bot)$

Initialize OPEN and EXPLORED; $\mathcal{N}^{\mathrm{goal}} \leftarrow \bot$
Initialize the root node $\mathcal{N}^{\mathrm{init}}$
  $\mathcal{N}^{\mathrm{init}}.config \leftarrow \mathcal{S}$, $\mathcal{N}^{\mathrm{init}}.tree \leftarrow [\![\mathcal{C}^{\mathrm{init}}]\!]$
  $\boldsymbol{\mathcal{N}^{\mathrm{init}}.\tau \leftarrow \mathsf{task_assignment}(\mathcal{S})}$
  $\mathcal{N}^{\mathrm{init}}.parent \leftarrow \bot$, $\mathcal{N}^{\mathrm{init}}.neigh \leftarrow \emptyset$
  $\mathcal{N}^{\mathrm{init}}.g \leftarrow 0$
OPEN.push($\mathcal{N}^{\mathrm{init}}$); EXPLORED[$\mathcal{S}$] $\leftarrow \mathcal{N}^{\mathrm{init}}$
while OPEN $\neq \emptyset \wedge \neg$interrupt() do
  $\mathcal{N} \leftarrow$ OPEN.top()
  if $\boldsymbol{\mathsf{is_tapf_goal}(\mathcal{N}.config)} \wedge (\neg\spadesuit \vee \mathcal{N}.g < \mathcal{N}^{\mathrm{goal}}.g)$ then
    $\mathcal{N}^{\mathrm{goal}} \leftarrow \mathcal{N}$
  if $\spadesuit \wedge \mathsf{f}(\mathcal{N}^{\mathrm{goal}}) \leq \mathsf{f}(\mathcal{N})$ then
    OPEN.pop()
    continue
  if $\mathcal{N}.tree = \emptyset$ then
    OPEN.pop()
    continue
  $\mathcal{C} \leftarrow \mathcal{N}.tree$.pop()
  low_level_expansion($\mathcal{N}, \mathcal{C}$)
  $\mathcal{Q}^{\mathrm{new}} \leftarrow$ PIBT_configuration_generator($\mathcal{N}, \mathcal{C}, \boldsymbol{\mathcal{N}.\tau}$)
  if $\mathcal{Q}^{\mathrm{new}} = \bot$ then
    continue
  if EXPLORED[$\mathcal{Q}^{\mathrm{new}}$] $\neq \bot$ then
    $\mathcal{N}.neigh$.append(EXPLORED[$\mathcal{Q}^{\mathrm{new}}$])
    DOPEN $\leftarrow [\![\mathcal{N}]\!]$
    while DOPEN $\neq \emptyset$ do
      $\mathcal{N}^{\mathrm{from}} \leftarrow$ DOPEN.pop()
      for $\mathcal{N}^{\mathrm{to}} \in \mathcal{N}^{\mathrm{from}}.neigh$ do
        $g \leftarrow \mathcal{N}^{\mathrm{from}}.g + \mathsf{cost}_e(\mathcal{N}^{\mathrm{from}}, \mathcal{N}^{\mathrm{to}})$
        if $g < \mathcal{N}^{\mathrm{to}}.g$ then
          $\mathcal{N}^{\mathrm{to}}.g \leftarrow g$
          $\mathcal{N}^{\mathrm{to}}.parent \leftarrow \mathcal{N}^{\mathrm{from}}$
          DOPEN.push($\mathcal{N}^{\mathrm{to}}$)
          if $\spadesuit \wedge \mathsf{f}(\mathcal{N}^{\mathrm{to}}) < \mathsf{f}(\mathcal{N}^{\mathrm{goal}})$ then
            OPEN.push($\mathcal{N}^{\mathrm{to}}$)
  else
    Initialize a new node $\mathcal{N}^{\mathrm{new}}$
      $\mathcal{N}^{\mathrm{new}}.config \leftarrow \mathcal{Q}^{\mathrm{new}}$, $\mathcal{N}^{\mathrm{new}}.tree \leftarrow [\![\mathcal{C}^{\mathrm{init}}]\!]$
      $\boldsymbol{\mathcal{N}^{\mathrm{new}}.\tau \leftarrow \mathsf{task_assignment}(\mathcal{Q}^{\mathrm{new}})}$
      $\mathcal{N}^{\mathrm{new}}.parent \leftarrow \mathcal{N}$, $\mathcal{N}^{\mathrm{new}}.neigh \leftarrow \emptyset$
      $\mathcal{N}^{\mathrm{new}}.g \leftarrow \mathcal{N}.g + \mathsf{cost}_e(\mathcal{N}, \mathcal{N}^{\mathrm{new}})$
    OPEN.push($\mathcal{N}^{\mathrm{new}}$)
    EXPLORED[$\mathcal{Q}^{\mathrm{new}}$] $\leftarrow \mathcal{N}^{\mathrm{new}}$
    $\mathcal{N}.neigh$.append($\mathcal{N}^{\mathrm{new}}$)
if $\spadesuit \wedge$ OPEN $= \emptyset$ then
  return backtrack($\mathcal{N}^{\mathrm{goal}}$)
else if $\spadesuit$ then
  return backtrack($\mathcal{N}^{\mathrm{goal}}$)
else if OPEN $= \emptyset$ then
  return `NO_SOLUTION`
else
  return `FAILURE`

is therefore impractical.

LaCAM* addresses this problem through lazy successor generation. Each high-level node maintains a low-level constraint tree that partially specifies agents' next locations. For each constraint, PIBT constructs one promising collision-free successor that satisfies it, while alternative constraints remain available for later visits. Progressive expansion eventually generates every valid successor without enumerating them all at once, so LaCAM* searches the full reachable graph while avoiding the upfront branching of direct A*.

The key observation is that feasible future transitions depend only on the current configuration and problem definition, not on how the configuration was reached. Given an appropriate heuristic toward a goal, LaCAM* can retain the same high-level search framework. Extending it to TAPF therefore mainly requires replacing MAPF's fixed-target guidance with assignment-aware guidance that selects temporary targets at each configuration and directs successor generation toward a feasible TAPF goal. This observation leads to ITA-LaCAM.

B. Integrating Target Assignment into Configuration Search

Extending LaCAM* to TAPF does not require changing its configuration-space search; the main challenge is to guide each configuration toward one of several feasible TAPF goals by determining which target each agent should currently pursue. A straightforward approach first computes an assignment and then solves the induced MAPF instance. CBS-TA follows this structure, while IR-TAPF updates each fixed assignment only after obtaining a complete MAPF solution. Both create a relatively long feedback loop between assignment and path finding: an unfavorable assignment may consume a complete MAPF call before path feedback becomes available.

Inspired by ITA-CBS, ITA-LaCAM computes a feasible minimum-distance matching $\mathcal{N}.\tau$ at every high-level node $\mathcal{N}$. Its temporary targets guide PIBT successor generation and the heuristic. Each new configuration immediately updates the matching from the new agent locations, allowing assignment and path finding to exchange feedback at every search node without changing LaCAM*'s underlying search. The assignment also defines the internal search cost, but does not permanently prune any valid successor before the first solution. Because every valid successor remains reachable through lazy expansion, ITA-LaCAM preserves LaCAM*'s completeness.

C. ITA-LaCAM

Algorithm 1 summarizes ITA-LaCAM; bold operations are the TAPF-specific additions to LaCAM*. Each high-level node stores a configuration, assignment $\mathcal{N}.\tau$, constraint tree, cost $\mathcal{N}.g$, parent, and outgoing neighbors. OPEN contains nodes with ungenerated successors, while EXPLORED maps configurations to nodes. Search uses $\mathsf{f}(\mathcal{N}) = \mathcal{N}.g + \mathsf{h}(\mathcal{N})$; because assignments change, $\mathcal{N}.g$ is an internal search cost rather than final-path SOC.

The constraint tree generates successors lazily. low_level_expansion retains alternative moves as constraints, and PIBT_configuration_generator uses PIBT, guided by $\mathcal{N}.\tau$, to complete each constraint into a collision-free successor. A failed constraint returns $\perp$ without discarding the remaining alternatives.

A new configuration creates a node; a known one adds an edge to its existing node, after which improved costs and parents are propagated. $\mathcal{N}.\tau$ is a minimum-distance eligible matching computed by the Hungarian algorithm. The root solves it fully; subsequent nodes reuse the parent's matching and dual variables and repair only rows of moved agents. A configuration is a TAPF goal when its locations form an eligible one-to-one target assignment.

IV. Experiments

We compare ITA-LaCAM primarily with IR-TAPF using DBS-Hungarian and include optimal ITA-CBS as a baseline. Our implementation builds on LaCAM with LaCAM2 optimizations. Experiments ran on Ubuntu 22.04.1 with an Intel Core i9-12900K and 128 GiB RAM. We use ITA-CBS setting replacing `empty-32-32` with `Symbotic`. Figure 1 shows eight maps.

A. Test Settings

In the *Group Test*, 10–200 agents are partitioned into groups of five sharing five targets. In the *Common Target Test*, each agent has one private and 15 additional targets, with 0%, 30%, 60%, or 100% shared, and we test 15–60 agents (5–60 on `maze`). Twenty seeded instances per map, setting, and agent count yield 3,200 and 6,560 instances, respectively; all admit perfect matchings. Every solver receives the same instances and a 10-second limit. ITA-LaCAM and IR-TAPF run anytime and return their best incumbent, whereas ITA-CBS returns the optimum. Quality comparisons use jointly solved instances and final-path SOC.

B. Success Rates

Figure 2 compares methods with optimal ITA-CBS on the Group Test. ITA-LaCAM solves all 3,200 instances, compared with 94.9% for IR-TAPF and 15.9% for ITA-CBS. ITA-CBS success falls rapidly as agent counts grow, illustrating scalability cost of maintaining optimality. IR-TAPF's failures are concentrated on `maze` and `room`, while it remains near 100% on more open maps. Because Group Test assignments decompose into independent groups of five agents, this pattern suggests that path congestion, rather than global assignment coupling, is the difficulty in this setting.

C. Runtime and Solution Quality

In Figure 3(a), median first-solution times are 0.010 s for ITA-LaCAM and 0.014 s for IR-TAPF. ITA-LaCAM solves every instance, while IR-TAPF times out on 425; its per-configuration assignment updates avoid waiting for a complete solution under a fixed, unfavorable assignment. Figure 3(b) shows SOC in the Group Test (median ratio 0.998), but ITA-LaCAM wins over 90% of paired cases at 60% and 100% target overlap. Against optimal ITA-CBS in Figure 3(c), 90% of ITA-LaCAM's solutions are within 10.2% of optimal on jointly solved instances.

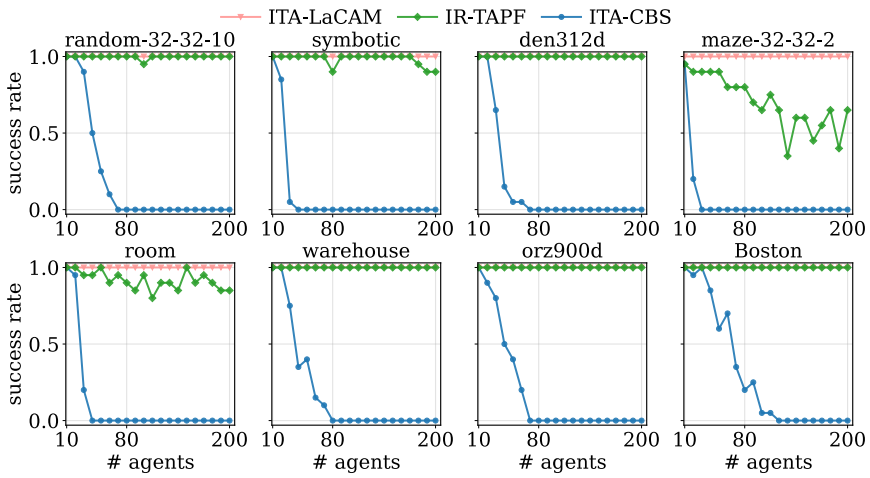


Fig. 2. Group Test success rates over 20 instances per point with a 10-second limit. ITA-LaCAM, IR-TAPF, and ITA-CBS solve 100%, 94.9%, and 15.9% of the 3,200 instances, respectively.

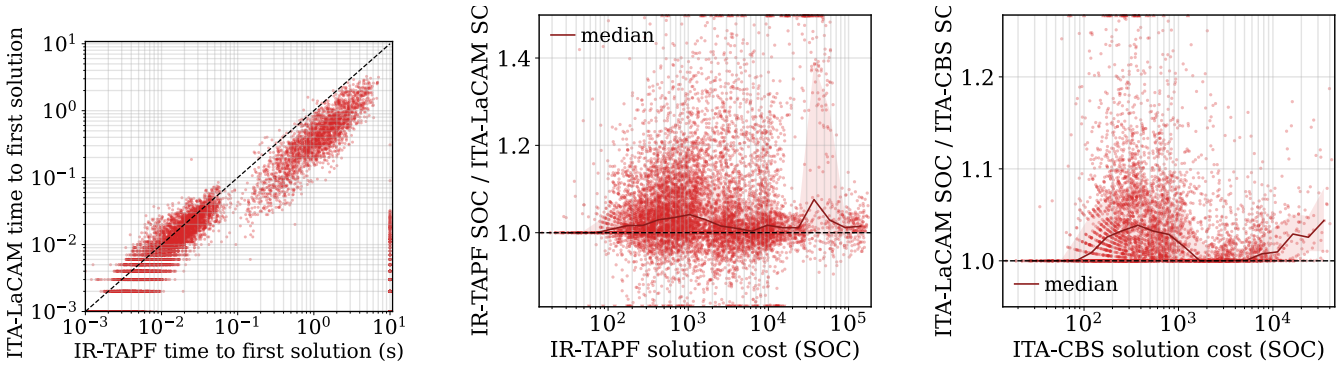


Fig. 3. Performance comparisons. (a) First-solution times on 9,760 instances; points below the diagonal favor ITA-LaCAM, while IR-TAPF failures lie at $x = 10$ s. (b) Paired ITA-LaCAM/IR-TAPF SOC on 9,335 instances (ITA-LaCAM lower: 65.0%; median ratio: 1.020). (c) Paired ITA-LaCAM/ITA-CBS SOC on 4,955 instances (ITA-CBS lower: 67.7%; median ratio: 1.016). Curves in (b) and (c) show binned medians.

V. Conclusion

We presented ITA-LaCAM, which integrates incremental target assignment into LaCAM-style configuration search instead of solving a sequence of fixed-assignment MAPF problems. It solved 9,760 instances within 10 seconds and outperformed IR-TAPF in first-solution time and SOC on 84.0% and 65.0% of comparisons, respectively. Future work includes learned guidance and adaptive assignment-update frequencies.

References

[1] J. Yu and S. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2013.

[2] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, 2015.

[3] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[4] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[5] K. Okumura, "Lacam: Search-based algorithm for quick multi-agent pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

[6] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Mapf-lns2: fast repairing for multi-agent path finding via large neighborhood search," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

[7] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2016, p. 1144–1152.

[8] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[9] Y. Tang, Z. Ren, J. Li, and K. Sycara, "Solving multi-agent target assignment and path finding with a single constraint tree," in *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE, 2023, pp. 8–14.

[10] Y. Kumagai and K. Okumura, "Alternating target-path planning for scalable multi-agent coordination," *arXiv preprint arXiv:2605.07744*, 2026.

[11] K. Okumura, "Improving lacam for scalable eventually optimal multi-agent pathfinding," 2023. [Online]. Available: https://arxiv.org/abs/2305.03632

[12] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, 2022.