Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

LaCAM* Variants for Minimizing Makespan in Multi-Agent Path Finding

Omer Idgar[1], **Dor Atzmon**[2], **Ariel Felner**[1]
[1]Ben-Gurion University
[2]Bar-Ilan University
idgaro@post.bgu.ac.il, dor.atzmon@biu.ac.il, felner@bgu.ac.il

Abstract

Multi-Agent Path Finding (MAPF) requires conflict-free paths. Optimal MAPF solutions often minimize the sum of the costs of the paths (SOC), or their maximum (*makespan*, MKS). *LaCAM** is a recent anytime MAPF solver, eventually converging to the optimal solution. However, LaCAM* was reported to have a considerably slow convergence speed to the optimum. Although this is true for SOC, in this paper, we show that LaCAM* can quickly find optimal MKS solutions. Additionally, currently, due to its anytime nature, LaCAM* uses a branch-and-bound search mechanism. We introduce a version of LaCAM* that uses IDA* and show that it has superb performance for finding optimal MKS solutions, even with thousands of agents. We explain all these phenomena.

1 Introduction and Background

Multi-Agent Path Finding (MAPF) (Stern et al. 2019) is the problem of navigating multiple agents, such as drones, robots, vehicles, etc., without conflicts. The input to MAPF is $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ – an undirected graph, and $S = (s_1, \ldots, s_k)$ and $G = (g_1, \ldots, g_k)$ – lists of start and goal vertices for k agents, respectively. *Plan* $\Pi = (\pi_1, \ldots, \pi_k)$ is a list of paths, where each *path* $\pi_i \in \Pi$ starts with s_i and ends with g_i. The path is composed of *move* actions (go to an adjacent vertex), and *wait* actions (stay idle in current vertex). Two paths π_i and π_j *conflict* either when the agents are simultaneously at some vertex or when the agents traverse an edge in opposite directions. A *solution* to MAPF is a conflict-free plan Π; that is, any two paths do not conflict. The cost $c(\pi_i) = |\pi_i| - 1$ of path π_i is the number of actions performed in it. Two common cost functions for solutions Π are: (1) *sum-of-costs* (SOC), which is $C(\Pi) = \sum_i c(\pi_i)$, and (2) *makespan* (MKS), which is $C(\Pi) = \max_i c(\pi_i)$. Optimally solving MAPF for SOC or MKS is NP-hard (Yu and LaValle 2013; Surynek 2010). However, advanced algorithms optimally solve MAPF for many agents (Sharon et al. 2015; Li et al. 2021; Gange, Harabor, and Stuckey 2019; Lam et al. 2022; Maliah, Atzmon, and Felner 2025).

MAPF search algorithms are executed in either a *vertex configuration space* (VCS) or a *path configuration space* (PCS). In VCS, each node n represents the vertices occupied by the agents. For n, we estimate the cost of reaching a goal for each agent separately, ignoring all other agents. Then, for SOC, $h(n)$ is the sum of all these estimates and, for MKS, $h(n)$ is their maximum. In PCS, each node represents a set of (possibly conflicting) paths for all agents.

LaCAM (Okumura 2023b) is a MAPF algorithm, capable of quickly finding suboptimal solutions. LaCAM has two search levels. On the high level, the algorithm is executed in VCS, starting from a $Root$ node containing the start vertices of the agents (start configuration). The high level performs a *depth-first search* (using a LIFO stack), and iteratively takes the node from the head of the stack and partially expands it – only a single successor is generated, and the node remains in the stack. A node is removed from the LIFO stack when all its successors have been generated. The low level chooses a (conflict-free) configuration for the next successor, often using *PIBT* (Okumura et al. 2022), a fast MAPF algorithm that efficiently moves the agents towards their goals. PIBT gives higher priority to agents with longer paths. LaCAM halts when the first goal configuration is found.

*LaCAM** (Okumura 2023a) is a recent anytime solver, which extends LaCAM and converges to the optimal MAPF solution (for SOC or MKS). On the high level, LaCAM* searches the VCS in a *depth-first branch and bound* manner (DFBnB) and prunes nodes n with $f(n) = g(n) + h(n) \geq U$, where U is the cost of the incumbent (best solution found thus far). LaCAM* halts only after it finishes scanning the entire search tree thus guaranteeing the solution optimality.

When LaCAM*'s low-level chooses a configuration, e.g., by PIBT, the agents may eventually become congested in narrow spaces. Therefore, recently, an improved version of LaCAM* was introduced (Okumura 2024) (denoted LaCAM*2), which has a few modifications. The main modification, called *Space utilization optimization* (SUO/Scatter), prevents multiple agents from reaching crowded areas by preferring configurations that better scatter the agents. While LaCAM*2 finds a (suboptimal) solution quickly, it converges to the optimal solution very slowly (for SOC).

Conflict-Based Search (CBS) (Sharon et al. 2015), a common optimal MAPF algorithm, searches in PCS (unlike LaCAM). CBS plans paths for the agents and, iteratively, resolves conflicts until a solution is found. Each of these algorithms LaCAM* and CBS has its own forte; while LaCAM* can quickly find (unbounded) near-optimal solutions (for SOC or MKS), CBS is considered better for finding

#Agents	Suboptimal		Optimal	
	SOC	MKS	SOC	MKS
100	100%	100%	0%	98%
200	100%	100%	0%	92%
300	100%	100%	0%	92%
400	100%	100%	0%	92%
500	100%	100%	0%	90%
600	100%	100%	0%	88%
700	100%	100%	0%	82%
800	100%	100%	0%	80%
900	100%	100%	0%	74%
1000	100%	100%	0%	64%

Table 1: LaCAM*2-BnB's success rate for SOC and MKS.

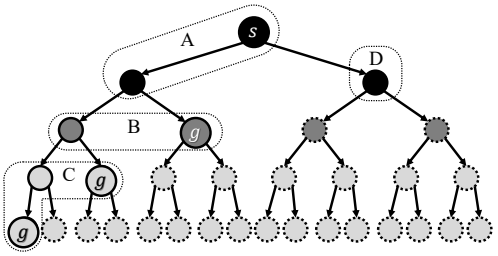

Figure 1: Node classes. Black: MEN; Dark gray: MBEN; Light gray: NEN. Solid circles: Expanded by DFBnB.

optimal or bounded-suboptimal solutions (again, for SOC or MKS) (Sharon et al. 2015; Li et al. 2019, 2021; Maliah, Atzmon, and Felner 2025; Barer et al. 2014; Li, Ruml, and Koenig 2021; Švancara et al. 2024; Boyarski et al. 2022).

In this paper, we optimally solve MAPF for MKS with LaCAM*, focusing on two main weaknesses of LaCAM*. A first weakness is that LaCAM* explores the VCS. In VCS, the number of successors of each node as well as the size of the state space is exponential in the number of agents. To prove optimality, as in standard heuristic search, every node n that may lead to a better solution (i.e., with $f(n) < C^*$, where C^* is the optimal solution cost) *must* be expanded (Dechter and Pearl 1985). In this paper, we show that, despite this weakness, LaCAM* can quickly optimally solve MAPF for MKS. As we explain, unlike SOC, the heuristic estimate for MKS is very accurate, especially on large and sparse maps. Therefore, in many cases, the number of nodes that must be expanded to prove optimality is very small or even zero. We experimentally demonstrate that, in such sparse maps, LaCAM* significantly outperforms CBSм, a CBS-based algorithm specifically designed for MKS, making LaCAM* the state-of-the-art optimal MAPF solver for MKS on these maps. A second weakness of LaCAM* is its DFBnB search strategy (denoted LaCAM*-BnB). With DFBnB, nodes n that cannot lead to a better solution (with $f(n) > C^*$) are expanded. Nevertheless, optimality can be proven while *never* expanding any of these nodes. Thus, we consider LaCAM* that searches in a IDA* manner (denoted LaCAM*-IDA*), which avoids expanding such nodes. Experimentally, LaCAM* with IDA* often significantly outperforms LaCAM* with DFBnB and CBSм in finding optimal MKS solutions, presenting state-of-the-art performance on various benchmark maps.

2 Finding Optimal Solutions with LaCAM*

LaCAM searches the VCS in a DFS manner. The main reason for this is that LaCAM was developed to find a (suboptimal) solution as fast as possible (the original paper was titled *"LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding"*) (Okumura 2023b). Following the same direction, LaCAM* (titled *"Improving LaCAM for Scalable Eventually Optimal Multi-Agent Pathfinding"*) (Okumura 2023a) searches in a DFBnB manner. It initially finds a solution and then repeatedly improves the quality of the solution until the optimal solution is found and proven. LaCAM* aimed to be scalable (the author mentions that *"it [LaCAM*] suboptimally solved 99% of the instances retrieved from the MAPF benchmark"*) but it did not excel in converging to the optimal solution (*"the convergence speed was slow in large instances with many agents"*). All this was reported and tested for SOC. However, *what about MKS?*

We experimentally examined the difference in finding suboptimal and optimal solutions for minimizing SOC and MKS with LaCAM*2-BnB on 50 problem instances of the *warehouse-20-40-10-2-1* map (Stern et al. 2019), with 100, 200, . . . , 1,000 agents. Table 1 presents the success rate (percentage of solved instances within a time limit of 60 sec).

As was already known, LaCAM*2-BnB excels in finding a first suboptimal solution and was able to suboptimally solve *all* problem instances for both SOC and MKS. Nevertheless, as expected, LaCAM*2-BnB could not optimally solve any problem instance for SOC. Importantly, LaCAM*2-BnB is executed in VCS. Given an underlying graph $\mathcal{G}$ with $|\mathcal{V}| = n$ vertices and k agents, there are $\approx n^k$ possible nodes for LaCAM*2-BnB. Specifically, our warehouse map contains 22,599 vertices and the size of the state space, for 1,000 agents, is $\approx 22,599^{1,000}$, which is a huge number. Despite all this, it turns out that LaCAM*2-BnB was able to optimally solve many problem instances for MKS (even 32 instances with 1,000 agents). These findings inspire the use of LaCAM* as an optimal algorithm for MKS. However, *how is it possible that such large problem instances were optimally solved?* We deepen into this next.

Why LaCAM* Works Well for MKS

It is well known that nodes expanded by heuristic search algorithms can be divided into three classes based on their f-value. (Dechter and Pearl 1985) **(1)** *Must-expand nodes* (MENs): nodes n with $f(n) < C^*$; they must be expanded to guarantee optimality. **(2)** *Maybe-expand nodes* (MBENs): nodes n with $f(n) = C^*$; they may or may not be expanded. **(3)** *Never-expand nodes* (NENs): nodes n with $f(n) > C^*$; they can be prevented from being expanded as they cannot possibly lead to an optimal solution or help in proving it. Figure 1 illustrates a search tree with three goal nodes at depths 2, 3, and 4. Assume a unit-edge cost, a zero heuristic value for all nodes, and a DFS that prefers the left successor first. Here, $C^* = 2$. The black nodes (at depths 0 and 1) are MENs, the dark-gray nodes (all at depth 2) are MBENs, and the light-gray nodes (depths 3 and 4) are NENs.

Let us classify the traversal of DFBnB on the example tree into three phases. **Phase 1** (finding): DFBnB first dives

into the left branch and finds the solution at depth 4. **Phase 2** (improving): it then gradually improves its solution until it reaches the optimal solution at level 2. **Phase 3** (proving): it continues to expand nodes to prove the optimality of the solution. DFBnB expands all nodes represented by a solid-line circle. We divide the nodes expanded by DFBnB into four groups (labeled A, B, C, and D). A, B, and C are MENs, MBENs and NENs that are expanded by DFBnB until finding the optimal solution. They are visited during phases 1 and 2. D are MENs that are expanded after the optimal solution has been found, during the proving phase (phase 3).

There are two sources for heavy computation in DFBnB. (1) DFBnB continues its DFS to prove the optimality of the solution. These are the D nodes in phase 3; one such node in our example. (2) The work done in phases 1 and 2, where DFBnB expands NENs in C; three such nodes in our example. But, on larger trees there might be many (sometimes an exponential number of) such C nodes (NENs) and D nodes (MENs). We remedy these two inefficiencies in turn.

D Nodes with the MKS Heuristic. As stated above, in SOC, $h(n)$ is the sum of the estimates of all agents to reach their goals from n. In MKS, $h(n)$ is the maximum of these estimates. A deeper look revealed that the initial heuristic estimate at the root node $Root$ ($h(Root)$) was perfect for *all* of the optimally solved problem instances for MKS by LaCAM*2-BnB. That is, $h(Root)$ for these instances precisely equals the makespan of the optimal solution. Figure 2(a) presents $\Delta = C^* - h(Root)$ for SOC and MKS on the same map (from Table 1) with 50, 100, ..., 250 agents. As can be seen, as the number of agents increases, for SOC, Δ increases ($h(Root)$ degrades), while, for MKS, Δ remains 0 ($h(Root)$ remains perfect). The reason that $h(Root) = 0$ for MKS is that almost all conflicts can be resolved (in the sense that configurations with conflicts are not created) by extending the paths of agents with non-maximal cost without increasing the cost of the solution (which is only affected by the maximal path). Only in rare cases (e.g., a cardinal conflict between two agents with maximal-cost paths) should a maximal-cost path be increased.

This explains why LaCAM* was very strong for MKS. In the case of a perfect heuristic, there are no MENs (with $f(n) < C^*$) in the search tree. Therefore, when the optimal solution is first found (end of phase 2), there are no more nodes to be expanded (D is empty) and the rest of the nodes that were not expanded at all have $f(n) \geq C^*$. Thus, for MKS, phase 3 does not exist, and the search can be immediately halted. In fact, the only modification we added to LaCAM*2-BnB (and LaCAM*-BnB) is that, when the cost of the best solution found (U) satisfies $U = h(Root)$, the algorithm immediately halts (this was not part of the original algorithm) without even the need to backtrack to the root.

C Nodes with the MKS Heuristic. We evaluate the impact of C nodes (NENs with $f(n) > C^*$) expanded by LaCAM*2-BnB for minimizing MKS on empty grids of sizes 32×32, 64×64, and 128×128. The results are presented in Figure 2(b), where every curve corresponds to a different grid size. Each empty grid was tested with agents that occupy 5%, 10%, ..., 30% of the grid (x axis).

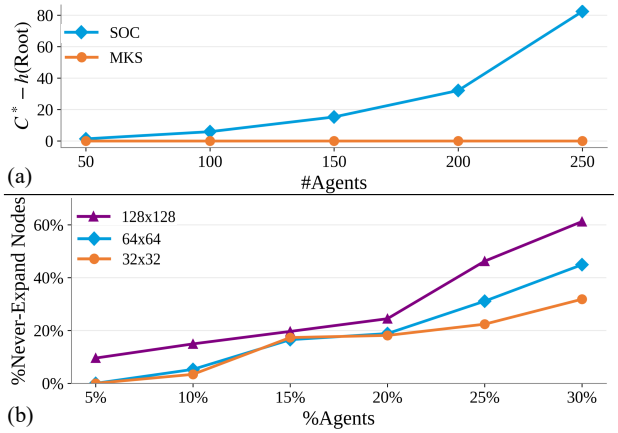

Figure 2: (a) Heuristic accuracy ($\Delta = C^* - h(Root)$) for SOC and MKS. (b) Percentage of NENs on empty grids.

We measured the average percentage of NENs expanded by LaCAM*2-BnB out of the total expanded nodes.

In general, increasing the number of agents also increases the percentage of NENs, as it is more likely for LaCAM*2-BnB to reach a non-optimal solution first. Notably, the percentage of NENs can be very high; in the 128×128 grid with 30% agents on average, more than 60% of the expansions were of NENs. We also observed individual cases where more than 99.7% were such NEN expansions. These results motivate the use of different search strategies, as we propose next, which do not expand these NENs.

Continuing LaCAM, standard LaCAM* searches in a DFBnB manner on its high level. Indeed, DFBnB is an anytime algorithm that converges to the optimal solution. However, here, we solely aim at finding optimal solutions. Hence, different known search strategies can also be employed while maintaining the optimality of the returned solution without expanding any of the NENs. In particular, we consider *Iterative Deepening* A* (IDA*) (Korf 1985), denoted LaCAM*-IDA*. IDA* performs DFS iterations according a threshold T. In each iteration, T is incremented to be the lowest f-value observed during the previous iteration. This ensures that the first solution found by IDA* is optimal. Therefore, running LaCAM* with IDA* (or A*) will not expand any NENs, and C nodes will never be expanded, while preserving optimality. This would be a great saving over DFBnB.

The main drawback of IDA* is that there is a unique iteration for each distinct f-value and such MENs are re-expanded in more than one iteration. Recall that, A* (Hart, Nilsson, and Raphael 1968) performs a systematic *best-first search* (BFS) and expands the node with the lowest f-value. Given that both A* and IDA* break ties according to the lowest h-value then in the last iteration of IDA* (with $T = C^*$) both algorithms expand exactly the same set of nodes. As Figure 2(a) shows, for MKS, $h(Root)$ is often perfect. Therefore, in such cases, only a single iteration of IDA* will be performed and each expanded node will only be expanded once (similarly to A*). By contrast, for SOC, $h(Root)$ is imperfect and becomes more inaccurate as the number of agents increases, many iterations will be required.

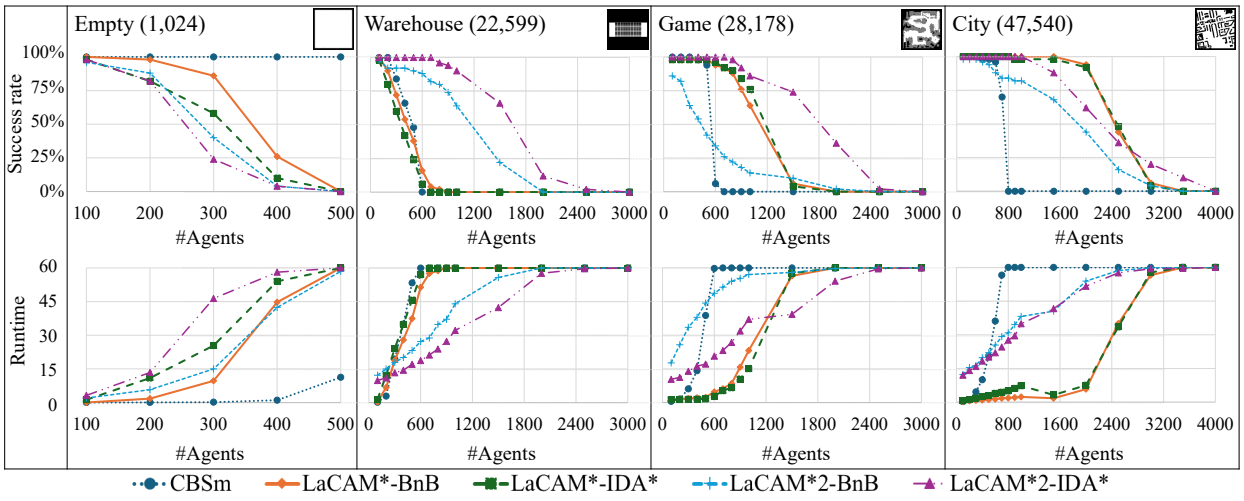

Figure 3: Success rate (top) and average runtime in seconds (bottom) on four benchmark maps. Time limit 60 seconds

3 Experimental Study

We conducted experiments (on an Intel® Core Ultra 9-185H CPU (8 cores) with 16GB RAM) with CBSm, LaCAM*-BnB, LaCAM*-IDA*, LaCAM*2-BnB, and LaCAM*2-IDA* on four representative benchmark maps from the *MovingAI* repository (Stern et al. 2019): *empty-32-32* (denoted Empty), *warehouse-20-40-10-2-1* (Warehouse), *den520d* (Game), and *Berlin_1_256* (City). As the number of agents for each map in the benchmark is limited (usually up to 1,000), we generated problem instances with the *MAPF-LNS2* (Li et al. 2022) instance generator using the publicly available implementation. For each map, 50 problem instances were created with 100, 200, ..., 1,000, and then 1,500, 2,000, ..., 4,000 agents. Our implementation is available at https://github.com/OmerIdgar/LaCAM-Makespan-Versions.

Figure 3 shows the success rate (top) and average runtime (bottom); the time limit per instance and the runtime of an unsolved instance is 60 seconds. The number next to the map's name is the number of empty cells in the map.

CBSm vs. LaCAM*. CBSm outperformed the LaCAM* solvers in the small map (Empty). A similar trend was observed by Maliah, Atzmon, and Felner (2025). In small maps, due to high density, often, $h(Root) < C^*$ and, as explained above, MENs exist for LaCAM*. Thus, searching with CBS on the PCS is a better choice (similarly to SOC). Moreover, the IDA* variants may need to perform multiple iterations. Thus, their DFBnB counterparts might perform better. By contrast, in the three larger maps, the LaCAM* versions significantly outperformed CBSm, highlighting that, unlike previously thought, LaCAM* is a very strong algorithm for optimally solving MAPF for MKS. Importantly, an IDA* variant of LaCAM* was almost always the best algorithm on the larger maps. In fact, LaCAM*2-IDA* optimally solved instances that contain thousands of agents for the first time (e.g., in City with 3,500 agents). In general, in all the instances that were optimally solved by either of the LaCAM* versions, $h(Root) = C^*$ was observed.

DFBnB vs. IDA*. In the three larger maps, LaCAM*2-IDA* significantly outperformed LaCAM*2-BnB. Unlike LaCAM*, The low-level of LaCAM*2 chooses configurations that better scatter the agents to sparse areas. For LaCAM*2-BnB, when scattering the agents, the f-value of successors often increases. This results in a longer runtime for DFBnB before finding the optimal solution due to NENs of type C. However, LaCAM*2-IDA* only considers nodes with minimal f-values. Therefore, when a successor of a higher f-value is created, it enforces the nodes to generate more successors until one with the minimal f-value is generated. As a result, LaCAM*2-IDA* scatters the agents but restrains this behavior by choosing a low-cost node. By contrast, LaCAM*-IDA* was only slightly better than LaCAM*-BnB. Without scattering, the f-value in DFBnB did not often increase above C^* so NENs were not common.

Comparing IDA* variants. In the midsize maps (Warehouse, Game), LaCAM*2-IDA* was better than LaCAM*-IDA*, mainly because LaCAM*2's low level chooses configurations that scatter the agents to sparse areas, and a solution can be quickly found. On the larger (and sparser) map (City), there was no need for scattering and, in many cases, LaCAM*-IDA* outperformed LaCAM*2-IDA*.

To summarize: On small dense maps, h is inaccurate; CBSm should be used. On larger, sparser maps, h is often perfect; LaCAM*2-IDA* is recommended. But, for very sparse maps, LaCAM*-IDA* is also recommended.

4 Conclusions and Future Work

We explain why LaCAM* can quickly optimally solve MAPF for MKS and suggest using different search strategies for it. We showed that on large maps, LaCAM* with IDA* is the state-of-the-art optimal MAPF solver for MKS.

Future work can (1) study CBS and LaCAM* more deeply; (2) employ other search methods for LaCAM*; (3) adapt LaCAM* for other objectives, e.g., fuel (Koyfman et al. 2025; Fine, Atzmon, and Agmon 2023); (4) improve LaCAM* using geometric constraints (Atzmon et al. 2023).

Acknowledgments

This work was supported by the Israel Science Foundation (ISF) grant #909/23 awarded to Ariel Felner and grant #1511/25 awarded to Dor Atzmon. This work was also supported by Israel's Ministry of Innovation, Science and Technology (MOST) grant #6908 (Czech-Israeli cooperative scientific research) awarded to Dor Atzmon and by a MOST grant awared to Ariel Felner.

References

Atzmon, D.; Bernardini, S.; Fagnani, F.; and Fairbairn, D. 2023. Exploiting Geometric Constraints in Multi-Agent Pathfinding. In *ICAPS*, 17–25.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *SoCS*, 19–27.

Boyarski, E.; Chan, S.; Atzmon, D.; Felner, A.; and Koenig, S. 2022. On Merging Agents in Multi-Agent Pathfinding Algorithms. In *SoCS*, 11–19.

Dechter, R.; and Pearl, J. 1985. Generalized Best-First Search Strategies and the Optimality of A^*. *J. ACM*, 32(3): 505–536.

Fine, G.; Atzmon, D.; and Agmon, N. 2023. Anonymous Multi-Agent Path Finding with Individual Deadlines. In *AAMAS*, 869–877.

Gange, G.; Harabor, D.; and Stuckey, P. 2019. Lazy CBS: implicit Conflict-based Search using Lazy Clause Generation. In *ICAPS*, 155–162.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1): 97–109.

Koyfman, D.; Atzmon, D.; Shperberg, S.; and Felner, A. 2025. Minimizing Fuel in Multi-Agent Pathfinding. In *SoCS*, 83–91.

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2022. Branch-and-cut-and-price for multi-agent path finding. *Computers & Operations Research*, 144: 105809.

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 *AAAI*, 10256–10265.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *IJCAI*, 442–449.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021. Pairwise symmetry reasoning for multi-agent path finding search. *AIJ*, 301: 103574.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. In *AAAI*, 12353–12362.

Maliah, A.; Atzmon, D.; and Felner, A. 2025. Minimizing Makespan with Conflict-Based Search for Optimal Multi-Agent Path Finding. In *AAMAS*, 1418–1426.

Okumura, K. 2023a. Improving LaCAM for scalable eventually optimal multi-agent pathfinding. In *IJCAI*, 243–251.

Okumura, K. 2023b. LaCAM: search-based algorithm for quick multi-agent pathfinding. In *AAAI*, 11655–11662.

Okumura, K. 2024. Engineering LaCAM*: Towards Real-time, Large-scale, and Near-optimal Multi-agent Pathfinding. In *AAMAS*, 1501–1509.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *AIJ*, 310: 103752.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *AIJ*, 219: 40–66.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *SoCS*, 151–159.

Surynek, P. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *AAAI*, 1261–1263.

Švancara, J.; Atzmon, D.; Strauch, K.; Kaminski, R.; and Schaub, T. 2024. Which Objective Function is Solved Faster in Multi-Agent Pathfinding? It Depends. In *ICAART*, 23–33.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI*, 1444–149.