Minimizing Fuel in Multi-Agent Pathfinding

Daniel Koyfman[1], Dor Atzmon[2], Shahaf Shperberg [1] and Ariel Felner [1]

[1]Ben-Gurion University of the Negev, Israel
[2]Bar-Ilan University, Israel
koyfdan@post.bgu.ac.il, dor.atzmon@biu.ac.il, {shperbsh,felner}@bgu.ac.il

Abstract

The multi-agent pathfinding problem (MAPF) of finding conflict-free paths for multiple agents has attracted a large number of researchers in the past. The cost of the solution is commonly measured by the sum-of-costs (SOC) cost function or, less commonly, by Makespan. In this paper, we focus on the Fuel cost function, which is the number of physical steps the agents traverse. While Fuel was mentioned in many previous papers, our paper is the first to deepen into it. We introduce an A*-based algorithm and a CBS-based algorithm for Fuel. We study Fuel theoretically, showing that it is (perhaps non-intuitively) can be more complex than SOC. Finally, we experimentally compare both algorithms against each other and against their SOC counterparts, studying their advantages and disadvantages.

1 Introduction

Multi-agent pathfinding (MAPF) is a widely studied problem where a set of agents needs to travel through collision-free paths from a set of start points to a set of goal points (Stern et al. 2019). There are several cost functions for MAPF solutions, as follows. The most common is *Sum-of-Costs* (SOC), which is the sum of the time that took each agent to reach its goal. Another cost function, albeit slightly less common, is *Makespan*, which is the time at which all agents reach their goal. Another cost function, often mentioned, is *Fuel* (also known as *total traveled distance*), and it is the sum of physical travels done by the agents. The difference between SOC and Fuel is whether *wait* actions have a cost. In SOC, wait actions cost like move actions but in Fuel wait actions are free of charge. In their survey, Stern et al. (2019) mention all three cost functions—SOC, Makespan, and Fuel—but state that SOC and Makespan are more common in the MAPF literature.

Notably, even those papers that mentioned Fuel did this in passing, and a deep treatment of Fuel was never presented. The *conflict-based search* algorithm (CBS) (Sharon et al. 2015), for example, is a prominent common optimal (minimal cost) MAPF algorithm, and it spawned many variants (Barer et al. 2014; Švancara et al. 2019; Li et al. 2020; Ma et al. 2017; Wan et al. 2018; Morag et al. 2024) and enhancements (Boyarski et al. 2015; Felner et al. 2018; Li et al. 2019a,b, 2021; Zhang et al. 2020; Shen et al. 2023) but none of them focused on Fuel.

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Fuel was discussed in several papers in the past. Yu and Rus (2015) prove the feasibility of MAPF on undirected graphs is a polynomial problem. They describe a MAPF variant as a pebble motion on graphs which is essentially the same definition as MAPF for Fuel. However, their algorithm only proves feasibility and does not solve the problem optimally. Geft and Halperin (2022) prove that optimizing Fuel is NP-hard, even when restricted to grid maps. However, no algorithms were presented specifically for minimizing Fuel. Okumura (2023) presents LaCAM*, an anytime approach to solve MAPF problems that eventually converges to optimality. But, finding optimal solutions was not the focus of that paper, and no experiments were performed for Fuel. The above list is just a representative sample. In general, Fuel was never deeply studied. A possible reason for this is that Fuel could be seen as a simple twist on SOC and even as a simpler problem to solve because wait actions are free. Additionally, tie breaking in the last search layer ($f = C^*$) and specifically with 0-cost edges was studied in (Asai and Fukunaga 2017). Note that, there is a difference between executing A* on graphs containing zero-cost edges and executing A* on graphs deriving from the joint action space of $MAPF_f$, where there are no zero-cost edges, as it is forbidden for all agents to wait at the same time.

In this paper, we deepen into Fuel. We consider two possible objectives. (1) Fuel: where wait actions are completely ignored. (2) Fuel_w: where wait actions are used as a secondary objective (we seek the solution with minimal waits among all minimal Fuel solutions). We present two algorithms specifically designed for Fuel. The first algorithm is an adaptation of the A* algorithm (Hart, Nilsson, and Raphael 1968) to treat the Fuel cost function. This adaptation is quite straightforward as it mainly requires adjusting the prioritization of the search for the required objective. The second algorithm is a CBS-based algorithm. Notably, the original CBS paper (Sharon et al. 2015) briefly discussed two algorithmic modifications that will allow CBS to minimize Fuel. However, as we explain, these modifications are correct but not practical. Instead, we introduce a new conflict, termed *wait conflict*, occurring when all agents wait at the same time, and provide a method that resolves such conflicts. We then provide a theoretical study of the state-

space of Fuel and, in contrast to a possible initial intuition, we show that zero-cost wait actions complicate the situation and, thus, Fuel is, arguably, more complex than SOC. Finally, we provide a systematic experimental comparison between our two algorithms on various maps. We also compare these algorithms to their SOC counterparts and show that modifying the algorithms to the Fuel objective causes a substantial increase in runtime and node expansion.

This paper is organized as follows. In Section 2, we formally define the problem of MAPF for Fuel. In Section 3, we provide background information on SOC solvers. In Section 4, we propose MAPF solvers for Fuel and Fuel$_w$. In Section 5, we theoretically analyze MAPF for Fuel. In Section 6, we experimentally study the different cost functions and proposed algorithms. Lastly, in Section 7, we conclude this paper and suggest possible directions for future work.

2 Definitions

The *MAPF* problem receives as input a graph $\mathcal{G} = (V, E)$, and three lists of: k agents $A = (a_1, \dots, a_k)$, start vertices $S = (s_1, \dots, s_k) \subseteq V$, and goal vertices $G = (g_1, \dots, g_k) \subseteq V$. In MAPF, time is discrete. A *path* π in graph $\mathcal{G}$ is a finite sequence of vertices representing the agent's movement. Let $\pi[t]$ denote the vertex at time step t according to π. Between any two time steps t and $t+1$, path π is composed of *move* actions ($(\pi[t], \pi[t+1]) \in E$) and *wait* actions ($\pi[t] = \pi[t+1]$). Let $|\pi|$ denote the number of actions in π, $|\pi|_m$ denote the number of move actions in π, and $|\pi|_w$ denote the number of wait actions in π (consequently, $|\pi| = |\pi|_m + |\pi|_w$).

A path π_i for agent $a_i \in A$ starts at $s_i \in S$ ($\pi[0] = s_i$) and ends at $g_i \in G$ ($\forall t \geq |\pi| : \pi[t] = g_i$). A *conflict* $\langle a_i, a_j, x, t\rangle$ between two paths π_i and π_j of agents a_i and $a_j (i \neq j)$ exists when either one of the following occurs:

1. $\pi_i[t] = \pi_j[t] = x$
2. $(\pi_i[t], \pi_i[t+1]) = (\pi_j[t+1], \pi_j[t]) = x$

The first conflict is referred to as *vertex conflict* and the second as *edge conflict*, where x is either a vertex or edge.

A *solution* $\Pi = (\pi_1, \dots, \pi_k)$ to MAPF is a list of paths where path π_i belongs to agent a_i and any two paths do not conflict. We also add a constraint over all agents from being allowed to perform a wait action simultaneously ($\forall t\ \exists \pi_i \in \Pi : \pi_i[t] \neq \pi_i[t+1]$). The following functions are defined on solutions:

Fuel is the sum of move actions in Π:

$$C_{Fuel}(\Pi) = \sum_{\pi_i \in \Pi} |\pi_i|_m$$

Wait is the sum of wait actions in Π:

$$C_{Wait}(\Pi) = \sum_{\pi_i \in \Pi} |\pi_i|_w$$

Sum-of-Costs (SOC) is the sum of costs of the paths in Π:

$$C_{SOC}(\Pi) = \sum_{\pi_i \in \Pi} |\pi_i|$$

By definition, $C_{Wait}(\Pi) + C_{Fuel}(\Pi) = C_{SOC}(\Pi)$. A solution is *optimal* if it minimizes some cost function. In this paper, we focus on the optimization of (1) SOC, which minimizes C_{SOC}; (2) Fuel, which minimizes C_{Fuel}; and (3) Fuel$_w$, which minimizes C_{Fuel} as a primary objective and C_{Wait} as a secondary objective. Trivially, any optimal Fuel$_w$ solution is also an optimal Fuel solution. However, in Fuel, as wait actions are free, agents may wait a large number of times. Fuel$_w$ aims to prevent this by taking the paths with minimal wait actions among all paths with minimal fuel actions. The problem depicted in Figure 1 contains two agents a_1 and a_2, and has different solutions for the above objectives. The optimal SOC solution is $\pi_1 = (s_1, A, B, g_1)$ and $\pi_2 = (s_2, E, D, C, g_2)$ with $C_{SOC}(\Pi) = C_{Fuel}(\Pi) = 7$. However, in an optimal Fuel solution, for example, agent a_2 has a different path of $\pi_2 = (s_2, s_2, s_2, B, A, g_2)$ where $C_{SOC}(\Pi) = 8$ but $C_{Fuel}(\Pi) = 6$. This solution is also optimal for Fuel$_w$. Notably, a solution where agent a_2 waits once more at s_2 is also optimal for Fuel (but not for Fuel$_w$).

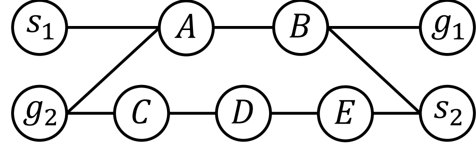

Figure 1: An instance with different SOC, Fuel, and Fuel$_w$ solutions.

In general, the two measures C_{Fuel} and C_{Wait} can be seen as two competing objectives forming a *bi-objective problem* (Salzman et al. 2023). That is, a solution Π is Pareto-optimal if there is no other solution Π' such that (1) $C_{Fuel}(\Pi') < C_{Fuel}(\Pi)$ and $C_{Wait}(\Pi') \leq C_{Wait}(\Pi)$ or (2) $C_{Fuel}(\Pi') \leq C_{Fuel}(\Pi)$ and $C_{Wait}(\Pi') < C_{Wait}(\Pi)$. The Fuel$_w$ solution is an extreme Pareto-optimal solution with the minimal C_{Fuel} and maximal C_{Wait} among all Pareto-optimal solutions. Likewise, one can consider the opposite cost function (Wait$_f$), which minimizes C_{Wait} as a primary objective and C_{Fuel} as a secondary objective. The Wait$_f$ solution represents the other extreme Pareto-optimal solution and can be used for cases where waiting should be minimized, ideally to zero. For example, when starting an engine incurs a significant cost or a case where you drive a baby in your car and the baby may wake up when the car stops moving (this is a real-life scenario often experienced by one of the authors). Also, any optimal SOC solution is a Pareto-optimal solution. Studying other Pareto-optimal solutions, besides SOC and Fuel$_w$, is beyond the scope of this paper. Likewise, we do not deal in this paper with the Makespan cost function (Maliah, Atzmon, and Felner 2025).

3 Background: SOC Solvers

A* (Hart, Nilsson, and Raphael 1968) is a well-known best-first search algorithm that chooses to expand nodes n with minimal $f(n) = g(n) + h(n)$ where $g(n)$ is the cost of a path to reach n from the start node and $h(n)$ estimates the cost from n to the goal. If h is *admissible*, i.e., it never overestimates the cost to the goal, then A* is guaranteed to return an optimal solution.

3.1 A* Solvers for SOC

To solve MAPF, A* is executed on the *composite search space*, which is defined as follows. Any node n represents a set of vertices for all agents, i.e., $n \in (V \times \cdots \times V)$. Start node n_s and goal node n_g represent the start and goal vertices of the agents, i.e., $n_s = S$ and $n_g = G$ (S and G are defined in Section 2). Each neighbor $n' = (v'_1, \ldots, v'_k) \in N(n)$ of node $n = (v_1, \ldots, v_k)$ represents move and wait actions for the agents, i.e., $\forall i : ((v_i, v'_i) \in E) \vee (v_i = v'_i)$, but excluding the combination where all agents wait at the same time (i.e., $n \neq n'$). Likewise, nodes that result in conflicts are invalid and are not generated.

A few enhancements were proposed for A* for solving MAPF. Standley (2010) proposed *independence detection* (ID) and *operator decomposition* (OD). In ID, agents are partitioned into groups, initially in individual groups. A path is planned for each group individually with A*. If there are conflicts between agents in different groups, groups of conflicting agents are merged, and A* replans for the agents in the merged group. In OD, only some agents perform an action in neighboring nodes (known as intermediate nodes). This allows A* to avoid generating surplus nodes whose f-value is larger than the cost of the optimal solution. *M** (Wagner and Choset 2015) improves ID by first resolving conflicts locally and delaying the full merge of agents. Recently, Okumura (2023) introduces the *LaCAM**, which performs an anytime depth-first search on the composite search space, converging eventually to the optimal solution.

3.2 CBS Solvers for SOC

A commonly-used algorithm for solving MAPF is *conflict-based search* (CBS) (Sharon et al. 2015). Constraints are the building blocks of CBS and are defined as $\langle a_i, x, t\rangle$ with an agent a_i, a vertex or edge x, and a time step t. This constraint prohibits agent a_i from occupying vertex $x \in V$ at time step t or from traversing edge $x \in E$ between time steps t and $t+1$. CBS is a two-level algorithm. The low level returns an individual path for each agent, which must satisfy the constraints imposed by the high level. Calculating such a path is possible, e.g., with *Space-Time A** (Silver 2005), where each node represents a pair of a vertex and a time step, and nodes that violate constraints are not generated. The high-level search is performed on a *constraint tree* CT. Each CT node N contains a set of constraints ($N.constraints$), paths for all agents that satisfy the constraints ($N.\Pi$), and the total cost (measured by SOC) of these paths ($N.cost$). The CT nodes are ordered in OPEN by their cost.

A root CT node with an empty set of constraints is inserted into OPEN. Then, iteratively, a CT node N with the lowest cost is extracted from OPEN. If $N.\Pi$ is conflict-free, it is returned as a solution. Otherwise, a conflict $\langle a_i, a_j, x, t\rangle$ is chosen and resolved be generating two CT child nodes N_i and N_j which inherit the constraints $N.constraints$, and add the additional constraints $\langle a_i, x, t\rangle$ to N_i and $\langle a_j, x, t\rangle$ to N_j. CBS is proven to return an optimal solution.

Many enhancements were proposed for CBS to reduce the size of the CT and reach a solution faster. Boyarski et al. (2015) prioritize conflicts in any CT node N such that resolving them increases the cost in the new CT nodes compared to the cost of N; Felner et al. (2018) and Li et al. (2019a) add admissible heuristics to the costs of CT nodes; Li et al. (2021), Zhang et al. (2020), and Shen et al. (2023) reason about constraints that can resolve multiple conflicts together; and Li et al. (2019b) impose a different type of constraints to resolve conflicts, which disjointly splits the optimal solution.

As a first paper on Fuel, we focus on the basic versions of A* and CBS, show how to adjust them for Fuel, and study them theoretically. Taking the above A* and CBS's enhancements that were done for SOC and adopting them for Fuel is a non-trivial task that is left for future papers by the current authors or by others in the community.

4 MAPF Solvers for Fuel and Fuel$_\text{w}$

4.1 A* Solvers for Fuel

We next adjust A*, presented above for SOC, for Fuel and Fuel$_\text{w}$. For both objectives, we execute A* on the composite search space, as defined in Section 3.1 for SOC. However, here, the cost of wait actions is zero. For Fuel, A* can be executed while exploring nodes with the lowest C_{Fuel} cost first. Any tie-breaking policy can be applied to enhance the search process. However, for Fuel$_\text{w}$, among the nodes with the lowest C_{Fuel}, the one with the lowest C_{Wait} must be chosen for expansion in order to obtain the required optimality. Thus, the only modification we need in A* variants is to change the priority function from C_{SOC} to C_{Fuel} or to C_{Fuel_w}.

For a heuristic, similarly to SOC, it is possible to use the sum of individual shortest path costs (SIC) as an admissible heuristic. For each agent. this heuristic assumes that no other agent exists and, therefore, wait actions are never needed. In fact, the SIC heuristic is more accurate for Fuel than for SOC. The accuracy of a heuristic $h(n)$ for a node n can be approximated by the fraction $h(n)/h^*(n)$ where $h^*(n)$ is the real cost of the shortest path from n to the goal. Notably, $h^*(n)$ for Fuel is *less than or equal to* $h^*(n)$ for SOC because the optimal SOC solution is a, possibly suboptimal, solution for Fuel. The SIC heuristic only counts moves, as individual agents never need to wait (as they ignore the existence of other agents altogether). Therefore, since SIC is identical for both SOC and Fuel, it is more accurate for Fuel than for SOC.

4.2 CBS Solvers for Fuel

In contrast to A* for Fuel (and Fuel$_\text{w}$), adjusting CBS requires several algorithmic modifications. A (quite obvious) modification, which was also mentioned by Sharon et al. (2015), is that nodes in the low and high levels of CBS must be prioritized by Fuel (or Fuel$_\text{w}$). However, there are cases where this modification may not be enough. Next, we will further modify CBS's low and high levels.

CBS's Low-Level. Consider the problem instance presented in Figure 2(a) where the low level is required to find the lowest-cost path (Fuel) for agent a_1 from s_1 to g_1 and

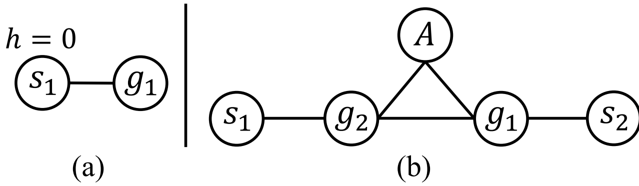

Figure 2: Examples of infinite wait actions on (a) CBS's low-level, and (b) CBS's high-level.

assume that $h(s_1) = 0$. Recall that the low level is executed on a state space where each node contains a vertex and a time step (Section 3.2). The low level starts by expanding the root node $(s_1, 0)$ and generating two successors $(s_1, 1)$ and $(g_1, 1)$. Now, because $f((s_1, 1)) = 0$ and $f((g_1, 1)) = 1$, the low level keeps generating nodes with $f((s_1, t)) = 0$ for any time step t, and a solution is never found. Therefore, for minimizing Fuel, (1) we *must* have a perfect heuristic, which estimates the exact cost assuming no constraints. As mentioned above, such a heuristic (SIC) is common in MAPF. But, note that having such a heuristic in CBS for SOC is not mandatory for the algorithm to work. However, even with a perfect heuristic (where $h(s_1) = 1$), the low level may prefer to always expand the node (s_1, t) for any time step t as $f((s_1, t)) = f((g_1, 1)) = 1$. Therefore, after prioritizing nodes according to the f-value of Fuel, (2) we *must* force the agent to prefer moving over waiting. For Fuel, we use the lowest h-value as a secondary priority, and for Fuel_w, we use the minimum number of wait actions as a secondary priority and the lowest h-value as a tertiary priority. Again, such an additional priority is not mandatory in CBS for SOC.

CBS's High-Level. Consider the problem instance presented in Figure 2(b) containing two agents a_1 and a_2. Here, the optimal Fuel solution costs 5, where one of the agents must not follow its lowest-cost path, which costs 2, and go through vertex A, which costs 3. In fact, each of the agents has an infinite number of lowest-cost (Fuel) paths, where it waits various number of time steps at each of the first two vertices on its path. Now, consider the CT constructed with CBS for Fuel. When a conflict occurs, the agents block each other, e.g., an edge conflict where the agents want to exchange their locations (g_2 and g_1). When this conflict is resolved, the low level adds more zero-cost waits to the lowest-cost paths of the agents (as it cannot add another move to the path of an agent, which increases the cost). Thus, the agents keep conflicting. For example, consider the branch in the CT where the agents are constrained alternately. This branch is infinitely long, and each CT node in this branch has a cost of 4. Thus, as CT nodes are prioritized by C_{Fuel}, each such CT node must be expanded.

This issue was already raised in the CBS paper (Sharon et al. 2015), and the following modification was considered. Yu and Rus (2015) showed that, for any solvable problem instance, the total number of actions required is $\leq O(|V|^3)$. Therefore, Sharon et al. (2015) suggested applying CBS for Fuel by setting a bound of $|V|^4$ on the number of actions allowed and pruning CT nodes exceeding that number. Consequentially, the size of the CT becomes finite, and the optimal Fuel will eventually be found. However, while this method is correct, it is not practical. Consider our example of Figure 2(b) and observe that $|V|^4 = 625$. Therefore, many CT nodes must be expanded before finding the optimal solution with cost 5. That is, the CT branch with cost 4, described above, can itself reach a depth of 625, making this method impractical. We experimented with the above method on the problem instance of Figure 2(b), which only contains five vertices and two agents, and it was solved only after more than 60 seconds, which is the time limit in our experiments below. We next propose an alternative modification, which identifies and resolves a new type of conflict, *wait conflicts*.

Wait Conflict: When Standard CBS Fails. A *wait conflict* emerges when all agents perform a wait action at the same time step. This includes agents that are already located in their goal location. It is defined as follows.

Definition 1 (Wait Conflict $\langle t \rangle$). *For a list of paths Π and time step t, a wait conflict exists if $\forall \pi_i \in \Pi : \pi_i[t] = \pi_i[t+1]$ (all agents wait at time step t).*

When such a conflict occurs, it means that the agents perform redundant wait actions which can be removed altogether. Thus, we want to prohibit all agents from waiting at the same time step. When this conflict is detected and chosen to be resolved in a CT node N, then N is split to k $(= |A|)$ new CT nodes which are generated as its children, instead of the usual split to two nodes as done by CBS for SOC to resolve conflicts. In each new child node, we set a *wait constraint* $\langle a_i, t \rangle$ on a different agent $a_i \in A$, which is prohibited from performing a wait action at time step t. This is a novel type of constraint where an agent is constrained not to use an action (wait in our case) regardless of its location in the graph.

Atzmon et al. (2020) defined the notion of *sound pair of constraints* when splitting a node to two children. This definition is needed for proving that CBS does not lose solutions by splitting. We extend this definition to sets of constraints.

Definition 2 (Sound Sets of Constraints). *For resolving a conflict at CT node N, the sets of constraints $C_1, \ldots, C_n$ are sound iff every optimal solution that satisfies $N.constraints$ also satisfies at least one set of constraints C_i $(1 \leq i \leq n)$.*

Let N be a CT node containing a wait conflict that is resolved by the above wait constraints, and let $N_1, \ldots, N_k$ be the new CT nodes created as part of this conflict resolution. Trivially, $N_1.constraints, \ldots, N_k.constraints$ are sound sets of constraints as they only eliminate solutions where all agents simultaneously wait. Algorithm 1 presents the high level structure of CBS, which can be used for different objective functions, e.g., SOC, Fuel_w, Makespan, etc. The algorithm begins by initializing a *Root* CT node containing an empty set of constraints and inserting it into OPEN (lines 1-5). Then, repeatedly, CBS extracts the lowest-cost CT node N from OPEN (lines 6-7). If $N.\Pi$ is conflict-free, it is returned as a solution (lines 8-9). Otherwise, the algorithm selects a conflict (with any priority order), conf, occurred in

Algorithm 1: Conflict-Based Search (CBS)

Input: $\mathcal{G}, A, S, G$
Output: A plan for all agents from S to G

1: Init OPEN, $Root$; $Root.constraints = \{\}$
2: **for** $a_i \in A$ **do**
3: Plan path $Root.\Pi.\pi_i$
4: $Root.cost = C(N.\Pi)$
5: Insert $Root$ into OPEN
6: **while** OPEN is not empty **do**
7: Extract best N from OPEN //Lowest solution cost
8: **if** $N.\Pi$ is conflict-free **then**
9: **return** $N.\Pi$
10: //Choose a conflict in N
11: conf = GetConflict(N)
12: //All agents involved in conf
13: **for** $a_i \in$ conf.agents **do**
14: //Get a single constraint from the current conflict
15: cons = conf.GetConstraint(a_i)
16: N_{child} = GenerateChild(N, cons)
17: Insert N_{child} to OPEN
18: **return** *No Solution*

N (line 11). For each agent involved in the conflict, a corresponding constraint, const, is generated and a new child node N_{child} is created with the added constraint. Each such child is then inserted into OPEN (lines 13–17). If no solution is found and OPEN is empty, No Solution is returned (line 18).

5 Theoretical Analysis for Fuel MAPF

Optimizing Fuel is NP-hard, even for grid maps as shown in (Geft and Halperin 2022). However, we next compare the search tree that is spanned in Fuel to that in SOC and show that in many cases Fuel requires many more node expansions.

The composite search space of a given instance under the Fuel and SOC objectives (described in Sections 3.1 and 4.1) is identical in terms of nodes and edges. However, the costs associated with edges differ between the two objectives due to the cost of wait actions. We next characterize which nodes must be expanded in Fuel but not in SOC, and vice versa.

Let n be a node in the composite search space, corresponding to a collection of paths $\Pi = (\pi_1, \ldots, \pi_k)$. Let $g_m(n)$ denote the number of *move* actions in Π ($= C_{Fuel}(\Pi)$), and let $g_w(n)$ denote the number of *wait* actions in Π ($= C_{Wait}(\Pi)$). Consequently, the g-value of nodes according to SOC is $g_{SOC}(n) = g_m(n) + g_w(n)$, whereas the g-value of nodes in Fuel is denoted as $g_{Fuel}(n) = g_m(n)$. Additionally, we denote the optimal solution for SOC and for Fuel by C^*_{SOC} and C^*_{Fuel}, respectively. It follows directly that $C^*_{Fuel} \leq C^*_{SOC}$.

Under the standard assumptions, to prove the optimality of solutions, all nodes n with $f(n) < C^*$ *must be expanded* (Dechter and Pearl 1985). Such nodes are called *must expand nodes*. For SOC, this means all nodes n with:

$$g_m(n) + g_w(n) + h(n) < C^*_{SOC} \quad (1)$$

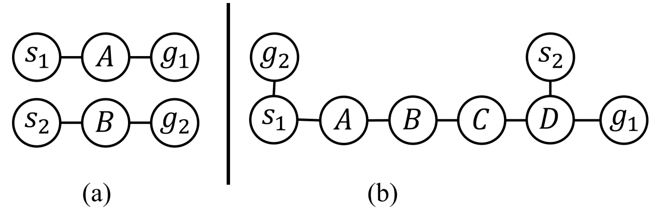

Figure 3: Examples of cases where (a) $C^*_{SOC} = C^*_{Fuel}$, and (b) $C^*_{\texttt{diff}}$ is $O(k \cdot C^*_{Fuel})$.

while for Fuel, this means nodes with:

$$g_m(n) + h(n) < C^*_{Fuel} \quad (2)$$

Using these conditions, we divide nodes into groups:

- ME_{SF}: Nodes that must be expanded in both SOC and Fuel. These nodes satisfy both inequalities.
- $\text{ME}_{\overline{SF}}$: Nodes that must be expanded in neither SOC nor Fuel. These nodes satisfy neither of the inequalities. Their cost is $\geq C^*$ for both measures.
- $\text{ME}_{S\overline{F}}$: Nodes that must be expanded in SOC but not in Fuel. These nodes satisfy Inequality 1 but not Inequality 2.
- $\text{ME}_{\overline{S}F}$: Nodes that must be expanded in Fuel but not in SOC. These nodes satisfy Inequality 2 but not Inequality 1.

A* for both Fuel and SOC must expand all the nodes in ME_{SF}. They differ in the nodes that are in $\text{ME}_{S\overline{F}}$ vs. nodes in $\text{ME}_{\overline{S}F}$. There are cases where $|\text{ME}_{S\overline{F}}| > |\text{ME}_{\overline{S}F}|$ and cases where $|\text{ME}_{S\overline{F}}| < |\text{ME}_{\overline{S}F}|$. Assume we solve the problem instances in Figures 1 and 2(b) with A* using the SIC heuristic function. For Figure 1, $|\text{ME}_{S\overline{F}}| = \{(s_1, s_2), (A, B)\}$ and $\text{ME}_{\overline{S}F} = \emptyset$ (here, $f(n) = g_m(n) + g_w(n) + h(n) = g_m(n) + h(n) = 6$, while $C^*_{SOC} = 7$ and $C^*_{Fuel} = 6$). Thus, $|\text{ME}_{S\overline{F}}| > |\text{ME}_{\overline{S}F}|$. By contrast, for Figure 2(b), one of the agents must take a detour via vertex A, resulting in $C^*_{SOC} = C^*_{Fuel} = 5$. Consequently, the nodes (s_1, s_2) and (g_2, g_1) have $g_m(n) + g_w(n) + h(n) = g_m(n) + h(n) = 4$, thus, they are both in ME_{SF}. All other nodes have $g_m(n) + g_w(n) + h(n) \geq 5$, as one of the agents must either wait or further move. Thus, only these two nodes must be expanded in SOC. Nonetheless, for Fuel, agents can wait without increasing the f-value, thus, there are other nodes with $g_m(n) + h(n) < 5$. Overall, $|\text{ME}_{S\overline{F}}| = \emptyset$ and $\text{ME}_{\overline{S}F} = \{(s_1, g_1), (s_1, g_2), (g_2, s_2), (g_1, s_2)\}$, and, thus, $|\text{ME}_{S\overline{F}}| < |\text{ME}_{\overline{S}F}|$.

As in standard A*, any node n with $f(n) = C^*$ may or may not be expanded. This often depends on the tie-breaking rule used by the algorithm when $f(n)$ is equal. Here, we only focus on nodes that must be expanded by the algorithm. We next quantify which nodes are in $\text{ME}_{S\overline{F}}$ and in $\text{ME}_{\overline{S}F}$.

5.1 Nodes in $\text{ME}_{S\overline{F}}$

Nodes that are in $\text{ME}_{S\overline{F}}$ have $g_m(n) + g_w(n) + h(n) < C^*_{SOC}$, thus, it also holds that $g_m(n) + h(n) < C^*_{SOC}$. Since these nodes do not satisfy Inequality 2, $g_m(n) + h(n) \geq C^*_{Fuel}$. Consequently, the size of $\text{ME}_{S\overline{F}}$ is correlated with

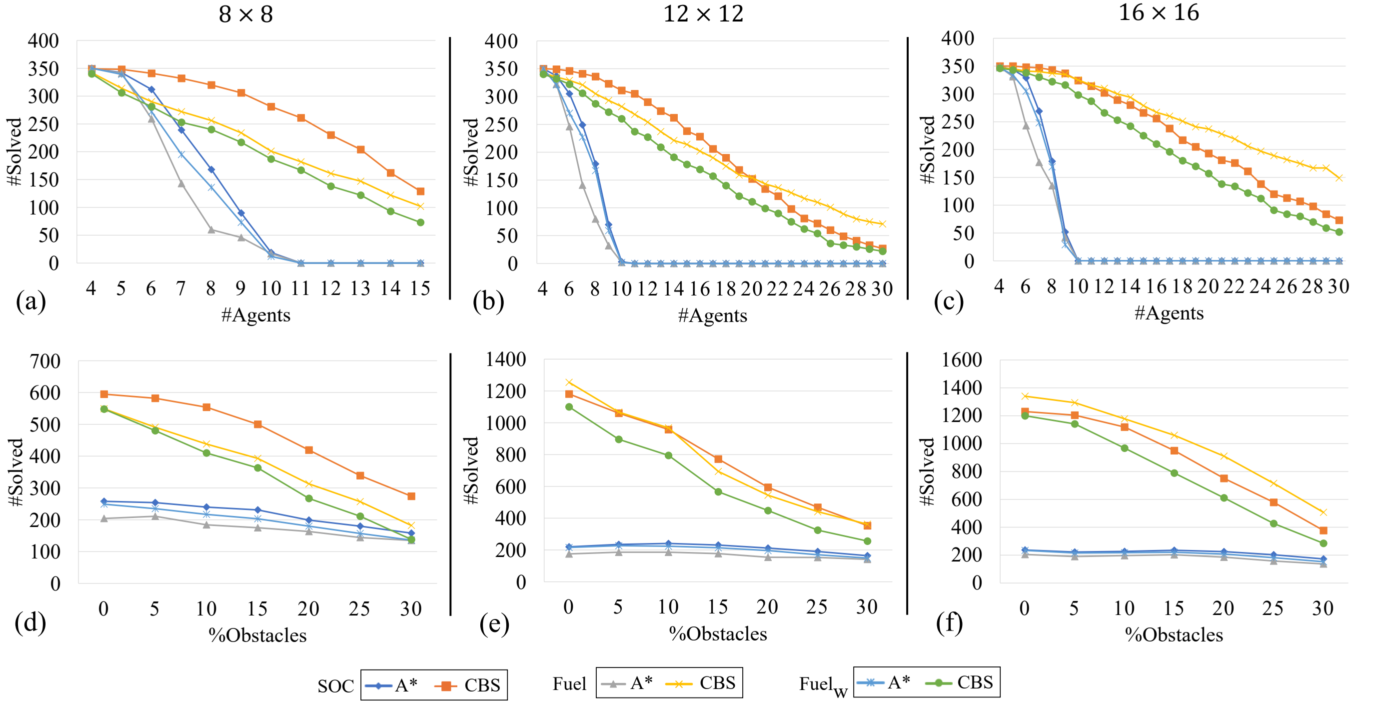

Figure 4: Success rate for A* and CBS while optimizing SOC, Fuel, and Fuel$_w$ on 8×8 (a,d), 12×12 (b,e), and 16×16 (c,f) grids.

the difference $C^*_{\texttt{diff}} = C^*_{SOC} - C^*_{Fuel}$, as well as the heuristic strength.

Lower bound on ME$_{S\overline{F}}$. If $C^*_{SOC} = C^*_{Fuel}$, ME$_{S\overline{F}}$ is an empty set, as in this case, every node that must be expanded under SOC must also be expanded under Fuel. Figure 3(a) shows an example in which all agents can take a shortest path to their goals without having to wait. Thus, in this example, $C^*_{SOC} = C^*_{Fuel}$. Therefore, in this case, ME$_{S\overline{F}} = \emptyset$.

Upper bound on ME$_{S\overline{F}}$. On the other extreme, where $C^*_{SOC} > C^*_{Fuel}$, ME$_{S\overline{F}}$ include additional nodes based on the difference between the two costs. If, for example, h is the (admissible) zero heuristic (i.e., $\forall n : h(n) = 0$), ME$_{S\overline{F}}$ will include a number of nodes that is exponential in $C^*_{\texttt{diff}}$. Specifically, in this case, each of the k agents can take b move actions, where b is the branching factor of the underlying graph, plus an additional wait action (total branching factor of $(b+1)^k$). However, the joint action in which all agents wait is forbidden (thus $(b+1)^k - 1$). Consequently, $|\text{ME}_{S\overline{F}}|$ is bounded from above by $((b+1)^k - 1)^{\frac{C^*_{\texttt{diff}}}{k}}$, where $\frac{C^*_{\texttt{diff}}}{k}$ is the depth, as every action of every agent costs 1.

The maximum value of $C^*_{\texttt{diff}}$ is $O(k \cdot C^*_{Fuel})$, as in the worst-case optimal solution for C^*_{SOC}, only one agent moves at each time step while the remaining $k-1$ agents wait. In this case, the total number of agent actions is C^*_{Fuel}, with each time step incurring $k-1$ additional wait actions. An example of such a case can be seen in Figure 3(b). In this example, the agents need to cross a single-spaced wide corridor from opposite directions. This means that agents move over the entire corridor one at a time, while all other agents that haven't reached their goal yet must wait until it is their turn to move. Thus, in the problem instance, $C^*_{\texttt{diff}} = O(k \cdot C^*_{Fuel})$. As a result, with the zero heuristic, $|\text{ME}_{S\overline{F}}| \leq ((b+1)^k - 1)^{C^*_{Fuel}}$, where C^*_{Fuel} can be arbitrarily large (e.g., by increasing the size of the corridor).

5.2 Nodes in ME$_{\overline{S}F}$

Nodes that are in ME$_{\overline{S}F}$ satisfy Inequality 2, but not Inequality 1. Thus, ME$_{\overline{S}F}$ include all nodes n for which (i) $g_m(n) + h(n) < C^*_{Fuel}$ and (ii) $g_w(n) \geq C^*_{SOC} - g_m(n) - h(n)$.

Lower bound on ME$_{\overline{S}F}$. When the heuristic is perfect, it holds that for all nodes n, $g_m(n) + h(n) = C^*_{Fuel}$. Thus, in this case, ME$_{\overline{S}F} = \emptyset$.

Upper bound on ME$_{\overline{S}F}$. Since $g_m(n) + h(n) < C^*_{Fuel}$ and $g_w(n) \geq C^*_{SOC} - g_m(n) - h(n)$ for all nodes $n \in$ME$_{\overline{S}F}$, we can substitute $-g_m(n) - h(n)$ in the second inequality with $-C^*_{Fuel}$ and obtain that $g_w(n) > C^*_{\texttt{diff}}$.

Next, we count the nodes that do not include any wait action, i.e., they only include move actions. There are at most $(b^k)^{\frac{C^*_{Fuel}}{k}} = b^{C^*_{Fuel}}$ such nodes with $g_m(n) + h(n) < C^*_{Fuel}$. We now bound the number of wait actions that can be added to each node. The minimal number of waits is $C^*_{\texttt{diff}}$ as $g_w(n) > C^*_{\texttt{diff}}$, and the maximal number is $(k-1) \cdot g_m(n)$ wait actions, as at most $k-1$ agents can wait at every time step. Notably, $(k-1) \cdot g_m(n)$ is bounded by $(k-1) \cdot C^*_{Fuel}$. When aiming to place m wait actions on a path of length l, the number of possible placements is equivalent to a stars and

bars problem, where we need to choose m elements out of l items, with replacement, which has $\binom{l+m-1}{m}$ possibilities. However, we need to exclude wait allocations in which all agents wait at the same time step. Due to stars and bars, the number of excluded wait allocations is expressed as: $\binom{l+m-1-k}{m-k}$. Overall, the upper bound on nodes in $ME_{\overline{SF}}$ is:

$$b^{C^*_{Fuel}} \cdot \sum_{m=C^*_{\mathtt{diff}}}^{(k-1)\cdot C^*_{Fuel}} \left(\binom{C^*_{Fuel}+m-1}{m} - \binom{C^*_{Fuel}+m-1-k}{m-k} \right)$$

To find an upper bound for the expression, we start by analyzing both the inner sum. Consider the inner term $\binom{C^*_{Fuel}+m-1}{m} - \binom{C^*_{Fuel}+m-1-k}{m-k}$. Generally, $\binom{C^*_{Fuel}+m-1}{m}$ dominates $\binom{C^*_{Fuel}+m-1}{m} - \binom{C^*_{Fuel}+m-1-k}{m-k}$. Thus, the term can be roughly bounded by $e^{m\cdot C^*_{Fuel}}$.

Next, we consider the sum:

$$\sum_{m=C^*_{\mathtt{diff}}}^{(k-1)\cdot C^*_{Fuel}} \left(\binom{C^*_{Fuel}+m-1}{m} - \binom{C^*_{Fuel}+m-1-k}{m-k} \right)$$

Since each term is bounded by $e^{C^*_{Fuel}\cdot m}$ and the sum of an exponential series is bounded by its last term, the sum is bounded by $O(e^{(k-1)\cdot {C^*_{Fuel}}^2})$. Multiplying the result by the outer term $b^{C^*_{Fuel}}$ we get:

$$O\left(b^{C^*_{Fuel}} \cdot e^{((k-1)\cdot {C^*_{Fuel}}^2)}\right)$$

Note that this bound is considerably larger than the upper bound on $ME_{\overline{SF}}$, indicating that the Fuel problem may be more complex than the SOC problem.

5.3 Limitation of the Analysis

While this analysis holds under standard assumptions, specific properties of MAPF may allow algorithms to avoid expanding certain must-expand nodes. For example, CBS exploits the fact that finding paths jointly can be decomposed into solving for each agent's lowest-cost path individually, with information (constraints) shared only in the case of conflicts. This allows CBS to avoid dealing with some must-expand nodes (in the composite search space). However, a formal characterization of how CBS factorization affects the definition of must-expand nodes, and whether other MAPF properties can further restrict them, remains under-researched in the literature, even for SOC, and presents an interesting future research direction.

6 Experimental Study

To compare the different cost functions with the A* and CBS solvers, we generated 50 maps for three grid sizes: 8×8, 12×12, and 16×16. Each such map contains 30% randomly placed obstacles. Each map has six additional variants, where we decrease the number of obstacles by 5% down to 0 (25,20,15,10,5,0). For each instance, we varied the number of agents between 4 and 15 in 8×8 and between 4 and 30 in 12×12 and 16×16. Six algorithms were executed for each instance: A* and CBS, varying in their objective: SOC, Fuel, and Fuel_w. We ran our experiments with a 1 min timeout on Linux virtual machines in a cluster of AMD EPYC 7763 CPUs, with 12GB RAM each.
Code – https://github.com/bernuly1/MAPF-fuel

6.1 Number of Solved Instances

Figure 4(a)-(c) shows the results for the three different grids. The x-axis represents the number of agents, while the y-axis indicates the number of solved instances within the 1-minute timeout. As expected, the A* variants solve problem instances with up to 10 agents, regardless of the cost function, but have a significantly lower success rate than the CBS algorithms. This can be explained by the large branching factor of A* exhausting the runtime. By contrast, the CBS-based algorithms have a much higher success rate. A similar trend can be seen in Figure 4(d)-(f), where we increase the map and count the successfully completed runs as a function of the obstacle percentage, averaged over all the different number of agents (1–30). Clearly, more obstacles reduce the number of solved instances within the time limit. An intriguing phenomenon emerges when analyzing the performance of the two CBS variants as the map size increases. On 8×8 maps, CBS for SOC consistently solves more problems than CBS for Fuel across all agents and obstacle configurations. However, on 12×12 maps, a shift occurs. Specifically, with fewer agents, CBS for SOC outperforms CBS for Fuel, but as the number of agents increases, the trend reverses, with CBS for Fuel taking the lead. Similarly, when the percentage of obstacles is low, CBS for Fuel tends to outperform CBS for SOC, but this advantage diminishes as the percentage of obstacles rises. On 16×16 maps, CBS for Fuel almost always dominates CBS for SOC. This shift in performance can be attributed to the behavior of the SIC heuristic. On larger maps, there are more instances where the SIC heuristic is perfect for Fuel but not for SOC. When the heuristic is perfect, solving Fuel instances becomes relatively straightforward as the root CT node already has the optimal cost. However, even slight imperfections in the heuristic can significantly increase the difficulty of solving these instances. Additionally, as the number of agents grows, C^*_{Fuel} is far less likely to increase compared to C^*_{SOC}. This is because, in many cases, additional agents can simply wait for others to move and still follow the optimal path to their objectives. Conversely, when the percentage of obstacles increases, particularly on smaller maps, the accuracy of the SIC heuristic decreases. This makes it more likely for C^*_{Fuel} to increase.

6.2 Impact of SIC Accuracy

To further support this explanation, we next present results on the accuracy of SIC in estimating the initial cost for each objective across all solved instances. Table 1 presents the results. The columns separate the instances into 3 buckets. Each bucket represents a range of ratio values. The ratio r is calculated by dividing $h_{SIC}(root)$ by the optimal cost. In each map, the columns contain the instances with ratios of $r < 0.9$, with $0.9 \leq r < 1$, and with $r = 1$. The table shows that all solvers perform significantly better on instances where $h_{\text{start}} = h^*$ ($r = 1$) across various grid sizes. For SOC, both A* and CBS were able to solve instances even when the heuristic was imperfect. In contrast, for Fuel, very few instances with an imperfect heuristic were

Cost	Solver	8×8			12×12			16×16		
		[0,0.9)	[0.9,1)	1	[0,0.9)	[0.9,1)	1	[0,0.9)	[0.9,1)	1
SOC	A*	112	581	827	16	356	1121	5	291	1226
	CBS	664	1609	990	114	3339	1932	20	3574	2616
Fuel	A*	4	56	1156	0	14	1158	0	3	1273
	CBS	0	7	2616	0	0	5330	0	0	7004
$Fuel_w$	A*	4	47	1326	0	11	1385	0	1	1431
	CBS	1	1	2415	0	0	4386	0	0	5421

Table 1: Success rate comparison by buckets across map sizes (8×8, 12×12, 16×16). Each bucket represents the ratio h_{start}/h^*.

%Obs.	8×8						12×12						16×16					
	SOC		Fuel			#Ins.	SOC		Fuel			#Ins.	SOC		Fuel			#Ins.
	M	W	M	W	W_{opt}		M	W	M	W	W_{opt}		M	W	M	W	W_{opt}	
0	46.40	0.45	46.14	4.57	1.01	541	118.18	0.50	117.64	9.66	1.34	1081	175.48	0.31	175.34	7.82	0.58	1200
15	42.79	1.25	41.76	8.65	3.04	357	88.07	0.80	86.88	10.20	3.17	562	136.88	0.72	135.85	11.58	2.51	764
30	36.92	1.96	35.52	9.35	4.65	138	73.25	2.27	72.15	10.27	4.77	253	101.07	1.76	99.65	11.27	4.79	280

Table 2: Average number of move (M), wait (W), and optimal wait (W_{opt}) actions for the different cost functions in each map size.

solved, particularly for $r < 0.9$. Notably, in Fuel, the accuracy of the heuristic had a greater impact on CBS than A*. For Fuel, while CBS solved significantly more problems than A* when the heuristic was perfect, it solved substantially fewer problems than A* when the heuristic was imperfect. Furthermore, as the map size increases, the number of instances solved with imperfect heuristics decreases. This aligns with the theoretical analysis, as the bounds on necessary expansions—which serve as a measure of problem difficulty (though less accurately for CBS)—grow with the optimal solution cost.

6.3 Number of Move and Wait Actions

Lastly, Table 2 compares C^*_{Fuel} and C^*_{SOC} by showing the number of the different actions taken for each objective averaged over problem instances that were solved for all cost functions. M and W indicate the average number of move and wait actions, receptively. W_{opt} indicates the average number of wait actions received from the $Fuel_w$ cost function. The data is shown across the different grid sizes and over obstacle percentages. In all three maps, as the obstacle percentage increases, so does the number of wait actions required to solve the problem optimally. As expected, the solution cost of SOC is larger than that of Fuel, albeit the difference is relatively small, especially in move actions. An interesting observation is the large number of wait actions for Fuel compared to $Fuel_w$. This suggests that the reason why CBS for Fuel solved more instances than CBS for $Fuel_w$ (as shown in Figure 4) is due to the larger solution space of Fuel. In contrast, $Fuel_w$ restricts CBS from adding any number of wait actions, and a solution is found only after every lower number of wait actions (and the same move cost) has been explored. This means that it is easier to find solutions where agents can wait many times than where wait actions are minimized. Future work can utilize the idea of adding more wait actions instead of trying different paths or adding constraints by optimal and sub-optimal algorithms for Fuel.

7 Conclusion and Future Work

This paper is the first to deeply study the Fuel cost function for MAPF. We considered two variants, where only the number of move actions is minimized (simply Fuel) and where the number of wait actions is also minimized as a secondary objective ($Fuel_w$). We then proposed A*-based and CBS-based algorithms for Fuel, studied Fuel MAPF theoretically, and compared the two algorithms experimentally.

There are many possible directions for future work. Future work can:

1. Adjust enhancements of A* and CBS for Fuel.
2. Explore the entire set of Pareto-optimal solutions considering the two measures C_{Fuel} and C_{Wait}.
3. Propose other algorithms for Fuel, e.g., based on *ICTS* (Sharon et al. 2013), *BCP* (Lam et al. 2022), or *Lazy CBS* (Gange, Harabor, and Stuckey 2019)
4. Extend this work to other problems, such as online/lifelong MAPF (Morag et al. 2022; Švancara et al. 2019; Li et al. 2020; Ma et al. 2017; Wan et al. 2018), where new agents/tasks arrive over time.

Acknowledgments

This work was supported by the Israel Science Foundation (ISF) grant #909/23 awarded to Shahaf Shperberg and Ariel Felner, by Israel's Ministry of Innovation, Science and Technology (MOST) grant #1001706842, in collaboration with Israel National Road Safety Authority and Netivei Israel, awarded to Shahaf Shperberg, by BSF grant #2024614 awarded to Shahaf Shperberg, by BSF grant #2021643 awarded to Ariel Felner, and by MOST grant #6908 (Czech-Israeli cooperative scientific research) awarded to Dor Atzmon. We thank Shao-Hung Chan for providing the initial CBS implementation.

References

Asai, M.; and Fukunaga, A. 2017. Tie-breaking strategies for cost-optimal best first search. *Journal of Artificial Intelligence Research*, 58: 67–121.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2020. Robust multi-agent path finding and executing. *JAIR*, 67: 549–579.

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.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. E. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *IJCAI*, 740–746.

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

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2018. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In *ICAPS*, 83–87.

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

Geft, T.; and Halperin, D. 2022. Refined Hardness of Distance-Optimal Multi-Agent Path Finding. In *AAMAS*, 481–488.

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.

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.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019a. 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. *Artificial Intelligence*, 301: 103574.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019b. Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search. In *ICAPS*, 279–283.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2020. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *AAAI*, 11272–11281.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *AAMAS*, 837–845.

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

Morag, J.; Felner, A.; Stern, R.; Atzmon, D.; and Boyarski, E. 2022. Online Multi-Agent Path Finding: New Results. In *SoCS*, 229–233.

Morag, J.; Zhang, Y.; Koyfman, D.; Chen, Z.; Felner, A.; Harabor, D.; and Stern, R. 2024. Prioritised Planning with Guarantees. In *SoCS*, 82–90.

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

Salzman, O.; Felner, A.; Hernández, C.; Zhang, H.; Chan, S.; and Koenig, S. 2023. Heuristic-Search Approaches for the Multi-Objective Shortest-Path Problem: Progress and Research Opportunities. In *IJCAI*, 6759–6768.

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

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195: 470–495.

Shen, B.; Che, Z.; Li, J.; Cheema, M. A.; Harabor, D. D.; and Stuckey, P. J. 2023. Beyond Pairwise Reasoning in Multi-Agent Path Finding. In *ICAPS*, 384–392.

Silver, D. 2005. Cooperative Pathfinding. In *AIIDE*, 117–122.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 173–178.

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 *SoCS*, 151–158.

Švancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online multi-agent pathfinding. In *AAAI*, 7732–7739.

Wagner, G.; and Choset, H. 2015. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219: 1–24.

Wan, Q.; Gu, C.; Sun, S.; Chen, M.; Huang, H.; and Jia, X. 2018. Lifelong Multi-Agent Path Finding in A Dynamic Environment. In *ICARCV*, 875–882.

Yu, J.; and Rus, D. 2015. Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, 729–746. Springer.

Zhang, H.; Li, J.; Surynek, P.; Koenig, S.; and Kumar, T. K. S. 2020. Multi-Agent Path Finding with Mutex Propagation. In *ICAPS*, 323–332.