# Tree-MAPF: On the Complexity of Optimizing Multi-Agent Path Finding on Tree Graphs

**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

In its general form, Multi-Agent Path Finding (MAPF) is well known to be NP-hard for various optimization objectives. But determining the complexity boundary for restricted topologies remains a key theoretical challenge. This paper investigates the complexity of MAPF on tree topologies. While recent work has established that minimizing Makespan on trees is NP-hard, the complexity of other standard metrics has remained an open question. We prove that, even on trees, optimizing Fuel (total traveled distance) and the Sum of Costs each remain NP-hard, closing a significant theoretical gap. Conversely, we identify a polynomial-time solvable case: restricting the agents to their individual shortest paths and determining if a feasible solution exists by only adding wait actions, thus maintaining the optimal Fuel cost.

## Introduction

Multi-Agent Path Finding (MAPF) is a fundamental problem in artificial intelligence and robotics, where the goal is to coordinate the movement of multiple agents from their respective start locations to their destinations without collisions. This problem is applicable to automated warehouses, traffic management (Wurman, D'Andrea, and Mountz 2008; Dresner and Stone 2008), and route planning for autonomous vehicles (Atzmon, Diei, and Rave 2019). As the number of agents scales, finding optimal solutions becomes computationally prohibitive, and so, general MAPF is known to be NP-hard for the three primary optimization objectives: **Makespan** (the total time until the last agent reaches its destination), **Sum of Costs** (SOC, the cumulative timesteps taken by all agents), and **Fuel** (the total distance traveled, excluding wait actions) (Yu and LaValle 2013).

Restricting graph topology often reveals tractable subclasses. Trees are the simplest connected graphs: they contain no cycles and provide a unique simple path between every pair of vertices. Several classical problems that are NP-hard on general graphs, including Independent Set and Vertex Cover, can be solved efficiently on trees; more generally, many hard problems become tractable on bounded-treewidth graphs (Garey and Johnson 1979; Arnborg, Lagergren, and Seese 1991). For MAPF, however, trees still contain narrow corridors and high-degree junctions that force agents to coordinate at bottlenecks. Trees also capture constrained settings such as dead-end rail yards (Hanou, de Weerdt, and Mulderij 2023). Related pebble-motion abstractions have also been used for stack-like container rearrangement (Han et al. 2018). Trees therefore serve as an important theoretical "frontier" for complexity analysis (Fioravantes et al. 2023). If an optimization problem on graphs admits a polynomial-time algorithm, trees are a natural first place to seek one. Conversely, proving NP-hardness on trees shows that cycles and alternative routes are not necessary for MAPF intractability: the difficulty can arise from agent coordination at bottlenecks alone.

It was shown that the *feasibility* problem—finding *any* valid non-colliding plan—is solvable in polynomial time on trees (Auletta et al. 1999; Masehian and Nejad 2009). In addition, finding *optimal* Makespan has recently been proven NP-hard on trees (Fioravantes et al. 2023, 2025). Nevertheless, the complexity of optimizing SOC and Fuel has remained unresolved. In this paper, we close this theoretical gap by resolving the complexity status of these key MAPF objectives on trees.

Our contributions are as follows. 1) We prove that minimizing Fuel is NP-hard on trees. 2) In contrast to this general hardness, we identify a formally tractable subclass for Fuel optimization: we prove that determining whether a valid collision-free schedule on trees exists by restricting all agents *exclusively* to their individual shortest paths can be decided in polynomial time. 3) Finally, we prove that minimizing SOC is also NP-hard on trees, closing a gap in the understanding of this metric.

## Background

### Problem Definition

We consider the MAPF problem on an undirected and unweighted graph $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges. The system consists of $k$ agents, labeled $a_1, a_2, \ldots, a_k$. Each agent $a_i$ has a unique start vertex $s_i \in V$ and a unique goal vertex $g_i \in V$. Time is discretized into timesteps $t \in \mathbb{N}$. At each timestep, an agent occupies exactly one vertex. In a single timestep, an agent can either *wait* at its current vertex or *move* to an adjacent vertex. A *path* $\pi_i$ for agent $a_i$ is a sequence of vertices

$\pi_i = (v_i^0, v_i^1, \ldots, v_i^T)$, where $v_i^0 = s_i$, $v_i^T = g_i$, and for all $0 \leq t < T$, $(v_i^t, v_i^{t+1}) \in E$ (move action) or $v_i^t = v_i^{t+1}$ (wait action). The path implies that agent $a_i$ is at vertex $v_i^t$ at timestep $t$. As in the standard MAPF formulation, after reaching its goal at time $T$, the agent is assumed to remain at $g_i$ indefinitely, i.e., $v_i^t = g_i$ for all $t \geq T$.

A valid joint plan $\Pi = \{\pi_1, \ldots, \pi_k\}$ must satisfy two collision constraints: **(1) Vertex Conflict**: No two agents can occupy the same vertex at the same timestep. Formally, for all distinct agents $i, j$ and all timesteps $t$, $v_i^t \neq v_j^t$. **(2) Edge Conflict** (Swapping Conflict): No two agents can traverse the same edge in opposite directions at the same timestep. Formally, for all distinct agents $i, j$ and all timesteps $t$, if $v_i^t = v_j^{t+1}$ and $v_i^{t+1} = v_j^t$, a conflict occurs.

In accordance with standard MAPF conventions, we do not enforce *following* conflicts; that is, an agent is permitted to move into a vertex that was occupied by another agent in the immediately preceding timestep.

Finally, to facilitate our theoretical analysis later in this work, we formally define a structural property that can exist within $G$: a *chain* of vertices. A vertex chain is a simple path sequence $(v_1, v_2, \ldots, v_m)$ wherein every intermediate vertex $v_i$ (for $1 < i < m$) has a connecting degree of exactly 2 in $G$. A chain forms a strict 1D corridor: because no branching junctions exist within the sequence, agents are physically gridlocked from swapping positions and must traverse the entire chain sequentially—or retreat universally back to the terminal endpoints to grant the right-of-way.

## Tree Topologies

In this paper, we restrict the input graph $G$ to a *tree topology*. A tree is an undirected, connected, and completely acyclic graph, meaning that fundamentally, $|E| = |V| - 1$. Crucially, this implies there exists one unique simple (and shortest) path between any two vertices $u, v \in V$. This unique geometry rigidly alters MAPF dynamics: lacking any topological rings, $k$ agents cannot swap their locations in a $k$-vertex subgraph (also known as a cycle rotation).

## Cost Functions

Our primary focus is on optimization. We evaluate the quality of a MAPF solution using the two standard scalar cost functions central to our complexity analysis:

1. **Sum of Costs (SOC)**: the total timesteps consumed by all agents until they arrive at their goals. Let $T_i$ be the arrival time of agent $a_i$ (the earliest timestep $t$ such that $v_i^{t'} = g_i$ for all $t' \geq t$). SOC is the standard cost function, optimized by most algorithms (Stern et al. 2019).
2. **Fuel (Total Traveled Distance)**: the total physical distance traveled by all agents, excluding wait actions. Let $dist(\pi_i)$ denote the number of move actions (non-wait steps) in path $\pi_i$. Fuel has recently attracted growing interest (Geft and Halperin 2022; Koyfman et al. 2025), showing that Fuel optimization is a surprisingly difficult task, lacking scalable algorithmic solutions.

$$(1)\ C_{\text{SOC}}(\Pi) = \sum_{i=1}^{k} T_i \quad ; \quad (2)\ C_{\text{Fuel}}(\Pi) = \sum_{i=1}^{k} dist(\pi_i)$$

Another objective is **Makespan**, the arrival time of the last agent (Maliah, Atzmon, and Felner 2025; Idgar, Atzmon, and Felner 2026). Minimizing Makespan recently proven to be NP-hard on trees (Fioravantes et al. 2025). Thus, we analyze the complexity of Fuel and SOC.

Let TMAPF be the MAPF problem instance where the graph structure is a tree, and let the subscripts $f$ and $s$ denote Fuel and SOC objectives, respectively. We evaluate these objectives as parameterized decision problems:

- **TMAPF**$_f$: Given a TMAPF instance and an integer bound $\beta$, does there exist a valid joint plan $\Pi$ such that $C_{\text{Fuel}}(\Pi) \leq \beta$?
- **TMAPF**$_s$: Given a TMAPF instance and an integer bound $\beta$, does there exist a valid joint plan $\Pi$ such that $C_{\text{SOC}}(\Pi) \leq \beta$?

## Relationship to Pebble Motion

MAPF is deeply intertwined with the classic *Pebble Motion on Graphs* problem (PMG). Early foundational work by (Kornhauser, Miller, and Spirakis 1984) established that finding any sequence of moves for labeled pebbles (where only one pebble moves at each timestep)—most closely related to the Fuel objective— is in class P (i.e., solved in polynomial time) on general graphs. When restricted to trees (PMT), finding *any* feasible solution (without optimality guarantees) is highly tractable, solvable in linear $\mathcal{O}(|V|)$ time (Auletta et al. 1999). Auletta and Persiano (2001) showed that on trees, finding the optimal solution for moving a single pebble out of the $k$ pebbles to its destination is also in class P. This is a variation of the *MAPF with Unassigned Agents* problem (MAPFUA) (Felner and Stern 2026; Gabay et al. 2026) where some agents do not have a goal assignment but can move if this is helpful for the assigned agents (Morag et al. 2025). Later work by Surynek (2010); Yu and LaValle (2013) formally bridged multi-robot path planning with pebble motion, proving that finding optimal paths with respect to Makespan, SOC and Fuel is intractable for general graphs. The work of TMAPF explores the boundaries of tree topologies both in MAPF and pebble motion. For instance, Masehian and Nejad (2009) and Krontiris, Luna, and Bekris (2013) leverage the linear-time feasibility of trees as a baseline for determining solvability and guiding path planners. Other works adopt hierarchical decoupled approaches to achieve cubic time complexity at the cost of optimality (Masehian and Nejad 2010), or abstract practical applications like dead-end train shunting yards as pebble motion on trees, mapping out sharp complexity boundaries based on capacity (Hanou, de Weerdt, and Mulderij 2023).

It is important to distinguish between standard MAPF, which allows agents to move concurrently, and classical pebble motion, which restricts movement to one pebble at a time. However, specifically on trees, optimizing TMAPF$_f$ is equivalent to sequential pebble motion. Because the tree topology lacks cycles, synchronous rotations are impossible; thus, any concurrent TMAPF$_f$ moves can be decoupled and serialized without increasing the total traversal distance. Despite this equivalence, the complexity of optimizing Fuel (or

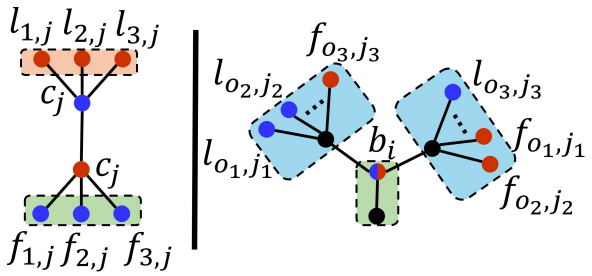


Figure 1: **Left:** Coat hanger Gadget. The top red box contains the arms of CH, and the green box contains the legs. The spine of CH is the two vertices connecting the two boxes. **Right:** Wings and tail gadget. The blue boxes are the right and left wings of WaT. The green box is the tail. The red vertices are the start vertices, and the blue are the goal vertices. The half red, half blue vertex denotes both the start and goal vertex of $b_i$.

the equivalent pebble motion) on trees has remained open. The following section resolves this gap.

## Optimizing TMAPF$_f$

In this section, we prove that optimizing Fuel on trees is NP-hard by reducing from the E3-SAT problem.

**Definition (E3-SAT).** The E3-SAT problem (Garey and Johnson 1979) is an NP-complete variant of the Boolean satisfiability problem. The input formula $F$ is in *conjunctive normal form* (CNF), consisting of $n$ variables $x_i$ and $m$ clauses $c_j$, with the strict constraint that each clause contains exactly three distinct literals (a literal cannot appear more than once in a clause). A valid output must determine if $F$ is satisfiable.

**Theorem 1.** *Optimizing TMAPF$_f$ is NP-hard.*

Before detailing the main TMAPF$_f$ reduction construction, we introduce two specialized graph gadgets underpinning its topological structure:

1. **The coat hanger (CH) gadget**. This gadget (illustrated in Figure 1(left)) structurally separates distinct paths into a coat-hanger topology via two primary components: the top and the bottom. The bottom component contains a central vertex designated as the *bottom spine*, which is adjacent to at most three *leg* vertices. An edge connects the bottom spine directly to a *top spine* vertex, which is subsequently connected to three distinct vertices known as the *arms*.
2. **The wings and tail (WaT) gadget**. The structurally symmetrical WaT gadget (illustrated in Figure 1(right)) divides flow choices into three main physical branches: the left wing, the right wing, and the tail. Each wing contains a *base* vertex, to which all localized wing extremities (called *feathers*) are uniquely connected. The tail serves as a conduit and consists of two sequentially connected vertices (the top and bottom of the tail). Finally, the top of the tail is connected to both the left and right wing bases, forming a Y-junction.

### High-Level Reduction Structure and Intuition

**Agents:** To map an E3-SAT formula into a TMAPF$_f$ instance, our reduction populates the constructed tree graph with four types of agents: *block agents* ($b_i$) corresponding to the variables, *clause agents* ($c_j$) corresponding to the clauses, *clause literal agents* ($l_{o,j}$) representing the literals, and *filler agents* ($f_{o,j}$) acting as physical counterparts to the literals. These agents navigate through the newly defined graph structures, which serve specific logical functions:

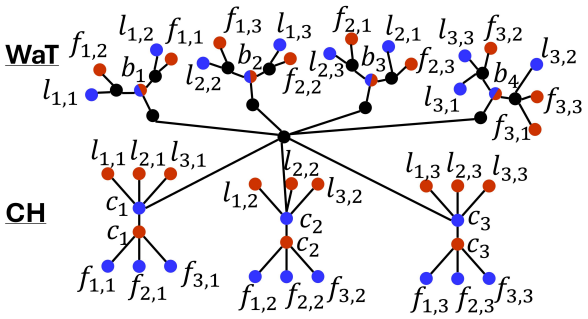


Figure 2: TMAPF$_f$ construction example for the E3-SAT instance $\langle\{x_1, x_2, x_3, x_4\}; \{(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)\}\rangle$. The red vertices are the start vertices and the blue are the goal vertices.

**The WaT gadget** acts as a *variable assignment switch*. A resident *block agent* begins at the tail, blocking *literal agents* trying to exit. To unblock the junction, the block agent must detour into either the left or right wing base, mimicking a rigid True or False assignment. Pinning itself to a specific base opens the path for the literal agents associated with that exact branch to safely pass into the central bottleneck.

**The CH gadget** acts as a *clause verifier*. A resident *clause agent* guards the spine, blocking external *filler agents* from reaching the legs. To let the filler agents pass, the clause agent must detour into one of the arms. Critically, it can only enter an arm if that arm has already been vacated by its assigned *literal agent* (representing a True literal), thus permanently enforcing that at least one literal in the clause must be structurally satisfied.

Intuitively, the reduction creates a central bottleneck vertex such that all paths of the literal and filler agents contain it. Since wait actions are free under Fuel but detours are not, the cost bound permits each block agent to visit both wings exactly once before returning to its goal. Any additional traversal between the wings exceeds the bound.

*Proof.* **Construction.** (Example in Figure 2) Given an E3-SAT problem instance, we create a corresponding MAPF$_f$ decision problem with $n + m + 3m + 3m = n + 7m$ agents ($n$ is the number of variables and $m$ is the number of clauses in the E3-SAT formula) on an undirected tree $G$:

$$A = \{b_1, \ldots, b_n, c_1, \ldots, c_m, l_{1,1}, \ldots, l_{3,m}, f_{1,1}, \ldots, f_{3,m}\}$$

The $b_i$ agents are called *block agents* (an agent for each variable), $c_j$ agents are called *clause agents* (there is an agent for each clause). Let $o$ be an integer $o \in \{1, 2, 3\}$. The $l_{o,j}$ agents are called *clause literal agents*, and the $f_{o,j}$ agents are called *filler agents* (we have each of those for every literal in the clauses). Next, the following steps are taken:

1. We create a single vertex and name it *the central vertex*.
2. Below, we build several CH and WaT gadgets. We connect the top spine vertex of each coat gadget and the bottom tail vertex of each wing and tail gadget to the central vertex, resulting in a tree.
3. For each clause, we construct a CH gadget and set the start vertex of the *clause agent*, $c_j$, as the bottom spine vertex. The goal vertex of $c_j$ is set as the top spine vertex. The start and goal vertices are neighbors.
4. We assign the start vertex of the *literal agents* of clause $c_j$ (or their negations)—$l_{1,j}$, $l_{2,j}$, and $l_{3,j}$—as the three arm vertices of the CH gadget corresponding to clause $c_j$. In addition, we set the goal vertex of the *filler agents* —$f_{1,j}$, $f_{2,j}$, and $f_{3,j}$— as the legs of the gadget. Note that each filler $f_{o,j}$ agent also corresponds to the literals of $c_j$ (or their negations).
5. For each variable, we create a WaT gadget and set both the start and goal vertices of the *block agent* $b_i$ to be the top of the tail vertex. Each $l_{o,j}$ is associated with a feather vertex which serves as its goal vertex and is connected to the corresponding wing base: If $l_{o,j}$ corresponds to a variable $x_i$ and is set in clause $c_j$ without negation (resp. with negation), then the vertex is connected to the left (resp. right) base wing of the gadget with $b_i$ in its tail. After connecting the goal vertex $l_{o,j}$ to a wing, we create a start vertex for $f_{o,j}$ and connect it to the base vertex of the opposite wing.
6. Lastly, we set the desired Fuel cost $\beta = 4n + 3m + 18m + 21m = 4n + 42m$. We aim to find a TMAPF Fuel solution with cost $\leq \beta$. This is explained next.

**Cost Bounds.** We next show the minimum number of move actions for each type of agent given the construction above. This will bound the Fuel cost as the wait actions are zero-cost and thus ignored for this purpose.

1. Every clause literal agent $l_{o,j}$ travels a fixed distance of 6, while the filler agents $f_{o,j}$ traverse a fixed distance of 7. For identically $3m$ literals and $3m$ filler agents, their absolute minimal Fuel contribution strictly equals $18m + 21m = 39m$.
2. The *clause agents* $c_j$ start and goal vertices are located on the optimal path of the filler agents. Since the filler agents $f_{o,j}$ do not have any other path that does not contain the start and goal vertices of $c_j$ (due to construction), the clause agents are forced to extend their path. The clause agents' minimum-length sequence is to climb the top spine, divert into the arm (2 steps), wait for $f_{o,j}$ to pass, and then return to the top spine goal (1 step), yielding precisely a path of length 3. Wait actions cost 0. Over $m$ gadgets, this giving a total cost of $3m$.
3. To allow transit through the WaT gadget, resident *block agents* $b_i$ must detour into one wing base, wait, then traverse completely across to the opposite wing base. Otherwise, the filler agents will be blocked in their wing, and the literal agents will not be able to enter the wing. The minimal path from the top tail vertex, into the left (resp. right) wing, directly back across the top tail vertex to the opposite wing, and terminating back at the goal tail inherently requires exactly 4 physical Fuel steps. Any further back-and-forth toggling exceeds this bound. For $n$ block agents, the cost is precisely $4n$.

Summing these lower bounds yields $4n+42m$. Therefore, any solution with Fuel cost at most $\beta = 4n + 42m$ must attain every lower bound with equality: each literal and filler agent follows its unique shortest path, each clause agent uses exactly three move actions, and each block agent uses exactly four move actions.

**Polynomial Construction.** The reduction can be computed in polynomial time. It creates $n + 7m$ agents, one constant-size CH gadget for each clause, and one constant-size WaT gadget for each variable, with one additional pair of vertices for each literal occurrence. Therefore, the number of agents and vertices is $\mathcal{O}(n + m)$, and the bound $\beta = 4n + 42m$ can also be computed in polynomial time.

**E3-SAT $\Rightarrow$ TMAPF$_f$.** Assume the E3-SAT formula is satisfiable with assignment $\widetilde{x}$. For each variable $x_i$, move $b_i$ to the right (resp., left) wing base if $\widetilde{x}_i$ is true (resp., false), opening the wing containing the goals of the true literal agents. Route these literal agents to their goals. Since every clause contains a true literal, at least one arm in each CH gadget is now vacant. Move each clause agent $c_j$ into such an arm, clearing the spine, and route the filler agents from the open wings through the CH gadgets. Next, move each block agent across the tail to the opposite wing base and route the remaining literal and filler agents. Finally, return each clause agent to its top-spine goal and each block agent to its tail goal. Every literal and filler agent follows its shortest path, each clause agent uses three moves, and each block agent uses four moves. Thus, the total Fuel cost is exactly $4n + 42m$.

**E3-SAT $\Leftarrow$ TMAPF$_f$.** Suppose the TMAPF$_f$ instance yields a solution with cost $\leq 4n + 42m$. To avoid exceeding this strict Fuel bound, each block agent $b_i$ can toggle between wing bases at most once. Initially, it must move to exactly one wing base, acting as a binary switch.

Once $b_i$ occupies a specific wing base (e.g., the right wing), the opposite (left) wing becomes unblocked. To avoid compounding detours (see the 3rd item in the cost bound), all filler agents $f_{o,j}$ starting in the unblocked wing move to their respective goals at the CH legs. Otherwise, these fillers become permanently trapped after the block agent returns from the opposite wing. These migrating fillers correspond to the negation of the literals whose goals are trapped in the blocked wing. Because the fillers route through the bottom spine of the CH gadget corresponding to their clause $c_j$, the resident clause agent $c_j$ is forced to detour (see the 2nd item in the cost bound).

Consequently, to preserve the $4n+42m$ bound, the clause agent will move to one of the vacant arms. A vacant arm is available iff a corresponding literal agent has moved to its goal vertex, as any vertex along the path overlaps with the corresponding filler agent path. Since the filler agents correspond to the negation of the literal agents in the unblocked wing, a literal and its negated literal cannot move before the block agent has moved to the opposite wing.

As a result, the clause agents $c_j$ can safely detour into these vacant arms, clearing the bottom spine without incurring excess cost. This occurs for every clause gadget before the block agent toggles to the opposite wing to release the remaining agents. By mapping the block agent's initial wing choice to a variable assignment (e.g., assigning $x_i$ = true when $b_i$ pins the right wing, and false when it pins the left), every CH gadget is guaranteed at least one unblocked literal arm. This establishes a valid, satisfying assignment for the E3-SAT formula. □

## Polynomial class of TMAPF$_f$

A central mechanism in our NP-hardness reductions is the forced deviation of auxiliary agents from their individual shortest paths. This raises a natural theoretical question: *What is the computational complexity of TMAPF$_f$ if every agent is strictly restricted to its unique shortest path?* Under this restriction, agents can only avoid collisions by utilizing wait actions, which do not incur a Fuel penalty. Thus, the theoretical optimal Fuel cost is preserved.

A polynomial-time feasibility test for shortest-path solutions can greatly help optimal (and suboptimal) MAPF solvers such as CBS (Sharon et al. 2015; Koyfman et al. 2025): (1) If the polynomial algorithm returns a solution, there is no need to execute an exponential algorithm like CBS. (2) If the polynomial algorithm returns that there is no solution, the algorithm can start the search with an increased lower bound, potentially pruning large chunks of the search space without exhaustively exploring unavoidable collisions.

**Theorem 2.** *Given a MAPF instance on a tree graph $G$ (TMAPF), determining whether a valid global plan $\Pi$ exists where every agent's path $\pi_i \in \Pi$ is strictly restricted to the spatial sequence of its unique shortest path is in class P.*

*Proof.* For each agent $a_i$, let $S_i$ be the sequence of vertices comprising its unique shortest path in $G$. Švancara et al. (2023) introduced a directed graph structure known as the *wait-graph*, denoted $\mathcal{W} = (A, E_W)$, to analyze such path restrictions. In $\mathcal{W}$, the vertex set $A$ corresponds to the set of agents. A directed edge $(a_i, a_j) \in E_W$ exists if and only if one of the following occurs:

1. The sequence $S_i$ contains the start vertex $s_j$ of agent $a_j$.
2. The sequence $S_j$ contains the goal vertex $g_i$ of agent $a_i$.

Švancara et al. (2023) demonstrated that if $\mathcal{W}$ is acyclic, a valid global plan $\Pi$ strictly utilizing the vertex sequences $\{S_1, \ldots, S_k\}$ is guaranteed to exist (by adding only wait actions which in the Fuel case are free). Constructing $\mathcal{W}$ and verifying whether it is acyclic evaluates in polynomial time. However, in general graph topologies, the presence of a cycle in $\mathcal{W}$ does not strictly imply unsolvability, as agents may resolve cyclic dependencies by moving simultaneously (e.g., rotating synchronously along a topological ring). We prove that the strict topological properties of a tree graph fundamentally preclude this, rendering any wait-graph cycle unconditionally unsolvable.

Let $C = \langle a_1, a_2, \ldots, a_m, a_{m+1} = a_1 \rangle$ be a directed cycle in $\mathcal{W}$. For each edge $(a_i, a_{i+1})$ in $C$, the dependency is witnessed by a concrete start or goal vertex: either $s_{i+1} \in S_i$, or $g_i \in S_{i+1}$. Let $u_i$ be such a witness vertex. For each agent $a_i$, let $P_i$ be the subpath of $S_i$ connecting $u_{i-1}$ to $u_i$. Concatenating these forced subpaths, $CW = P_1 \oplus P_2 \oplus \cdots \oplus P_m$, constructs a closed walk in $G$.

Since $G$ is a tree, any non-empty closed walk must traverse some edge in both directions (Diestel 2017). Because each $S_i$ is a simple shortest path, no individual subpath $P_i$ retraces its own steps. Thus, the backtracking in $CW$ is caused by distinct agents whose forced subpaths reuse part of the tree in opposite directions.

A directed cycle in $\mathcal{W}$ has no sequential execution order. In a general graph, such a dependency cycle may be resolved by a synchronized rotation around an actual graph cycle. The closed walk $CW$ shows why this is impossible on a tree: it reuses part of the tree in opposite directions, rather than forming a true cycle. Since agents are restricted to shortest paths, they cannot bypass the reused subpath; hence the cyclic precedence constraints cannot all be satisfied without either leaving the shortest paths or causing a collision.

Consequently, a valid global plan restricted to individual shortest paths exists on trees if and only if the wait-graph $\mathcal{W}$ is acyclic. Since $\mathcal{W}$ can be constructed in polynomial time and performing cycle detection via depth-first search is bounded by $\mathcal{O}(|A| + |E_W|)$ time, the decision problem of whether such a global plan $\Pi$ exists can be resolved in polynomial time. Thus, the problem is in class P. □

# Optimizing TMAPF$_s$

**Theorem 3.** *Optimizing TMAPF$_s$ is NP-hard.*

In this section, we prove Theorem 3. We achieve this by reducing from the 2P1N-SAT problem (Yoshinaka 2005).

**Definition (2P1N-SAT).** 2P1N-SAT is defined on a CNF formula with $n$ variables and $m$ clauses, like in E3-SAT. However, in 2P1N-SAT, each variable appears exactly 3 times in the clauses—twice as an unnegated literal and once as a negated literal (making the total number of literals exactly $3n$). Note that, unlike E3-SAT, 2P1N-SAT may contain any number of literals in each clause and not exactly 3. We also assume that $m > 2$, as $m \leq 2$ can be solved in polynomial time.

**Graph Construction: The Delay Gadget.** In contrast to Fuel, in SOC, both move and wait actions contribute one unit of cost. To restrict waiting agents from moving and to tightly synchronize their traversal schedules, we define a new parametrized graph structure: the delay gadget, denoted as $dg(d, e_{back}, e_{front})$.

$dg$ consists of a start chain and a goal chain both connected to a central *passing vertex*. The start chain begins with a single vertex $da_1$, followed by $e_{back}$ ($e_{back} \in \mathbb{N}$) intermediate vertices, then a sequence of $d-1$ vertices, and finally $e_{front}$ ($e_{front} \in \mathbb{N}$) vertices. The very last vertex in this sequence connects directly to the passing vertex. The goal chain consists of $2d-1$ sequentially connected vertices, with the $d$-th vertex connected to the passing vertex. We demonstrate $dg$ via Figure 3(a). The start chain is the

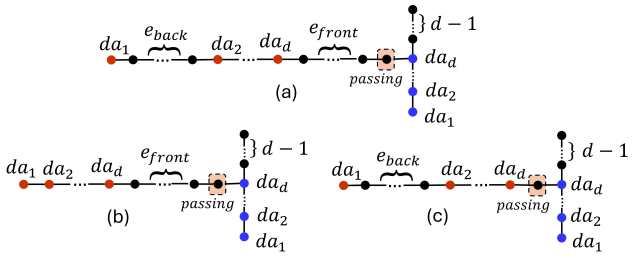


Figure 3: (a) Delay gadget (b) The window mode (c) The capacity mode. The red vertices are the start vertices and the blue are the goal vertices. The red box is the passing vertex.

horizontal chain (red). The goal chain is the bottom (blue) vertical chain. The additional chain (black) continues vertically from the last blue vertex.

**Agents and Semantic Modes.** We populate this gadget with $d > 0$ *delay agents*, denoted as $da_1 \dots da_d$. Agent $da_1$ starts at the first vertex of the start chain. Agents $da_2 \dots da_d$ start on the sequence of $d-1$ vertices located after the $e_{back}$ gap. Their respective goal vertices correspond to the first $d$ sequentially chained vertices in the goal chain such that the goal vertex of $da_d$ is connected to the passing vertex and the goal of $da_1$ is at the start of the chain.

We functionally deploy this gadget in two distinct semantic modes within our reduction:

• **Window Mode** $dg_{win}(d, e_{front})$: We set $e_{back} = 0$ (equivalent to $dg(d, 0, e_{front})$). This configuration, shown in Figure 3(b), enforces a tight schedule window, ensuring that at most $e_{front}$ external agents can safely pass through the passing vertex *before* the delay agents arrive and seal the path.

• **Capacity Mode** $dg_{cap}(d, e_{back})$: We strictly set $e_{front} = 0$ (equivalent to $dg(d, e_{back}, 0)$). This configuration, depicted in Figure 3(c), acts as a controlled filter, permitting exactly $e_{back}$ external agents to weave through the passing vertex *while* the sequence of delay agents is transiting, thereby placing a strict capacity limit on the passage.

**Lemma 1.** *The optimal path traversal of a delay gadget yields a minimal sum of costs of* $d(2d + e_{front} + e_{back})$ *for its* $d$ *delay agents. Functionally, the above modes govern this optimal threshold as follows:*

• *A Window Mode gadget* $dg_{win}(d, e_{front})$ *maintains this minimum cost if and only if at most* $e_{front}$ *external agents cross the passing vertex* before $da_d$ *reaches it. Any agent arriving late forces* $da_1$ *to wait, compounding the cost by* $+d$ *per delayed timestep.*

• *A Capacity Mode gadget* $dg_{cap}(d, e_{back})$ *maintains this minimum cost if and only if at most* $e_{back}$ *external agents go through the passing vertex* during *the transit gap between* $da_1$ *and the rest of the delay agents. Any agent attempting to cross outside this narrow transit window forces* $da_1$ *to wait, identically compounding the cost by* $+d$ *per delayed timestep.*

*Proof.* The bounds follow directly from the sequential linkage of the delay gadget graph:

1. The shortest uninterrupted path of the trailing delay agent $da_1$ is exactly $d + e_{front} + e_{back} + d = 2d + e_{front} + e_{back}$ steps. At this precise timestep, all $d$ delay agents can simultaneously reach their sequentially chained goals, yielding the theoretical minimum sum of costs. To achieve this, the delay agents must not be blocked at the passing vertex.
2. In a Window Mode gadget ($dg_{win}$), $da_1$ traverses the initial sequence to reach the passing vertex at exactly timestep $d + e_{front}$. Because $e_{front} > 0$, external agents have exactly $e_{front}$ timesteps to cross the passing vertex before $da_d$ arrives and seals the intersection.
3. In a Capacity Mode gadget ($dg_{cap}$), $e_{front} = 0$, so $da_d$ is adjacent to the passing vertex. The $e_{back}$ empty vertices between $da_1$ and $da_2$ create a transit gap before $da_1$ arrives. Thus, at most $e_{back}$ external agents can cross the passing vertex without delaying any delay agent.
4. Any external crossing outside the permitted timesteps postpones the earliest arrival of $da_1$. Since the delay agents preserve their order and can reach their assigned goals only as a contiguous sequence, the arrival of the trailing agent $da_1$ determines their earliest common completion time. Postponing $da_1$ by one timestep therefore postpones the arrival of all $d$ delay agents by one timestep, increasing the sum of costs by $d$.

□

**High-Level Reduction Structure and Intuition.** With the delay gadgets establishing rigid timing constraints, we inherit the bottleneck concept used in Fuel Proof, where all literal agents must pass sequentially through a central vertex. Because SOC penalizes wait actions, agents must be tightly synchronized.

We use an extended WaT gadget in which a block agent forces a choice between wings, simulating variable assignments. Only a valid satisfying assignment guarantees that literal agents can navigate the extended WaT and smoothly cross the delay gadgets' passing vertices within their strict $e_{front}$ or $e_{back}$ allowances, reaching their destinations without incurring additional cascading delay penalties.

*Proof.* **Construction.** (Example in Figure 4) Given a 2P1N-SAT problem instance, we create a corresponding TMAPF$_s$ decision problem with $n + 3n + m(m-1) + n(m+3) + n(m+4) + n(3n+7) + n(3n+8)$ agents on a tree $G$. The set of agents $A$ includes $n$ $b$-agents, $3n$ $l$-agents and five groups of $da$-agents as follows:

$$A = \{b_1, \dots, b_n, l_{1,1}, \dots, l_{3,n}, da_{1,1,1}, \dots, da_{1,m,m-1},\\ da_{2,1,1}, \dots, da_{2,n,m+3}, da_{3,1,1}, \dots, da_{3,n,m+4},\\ da_{4,1,1}, \dots, da_{4,n,3n+7}, da_{5,1,1}, \dots, da_{5,n,3n+8}\}$$

The graph is built as follows:

1. We create a single vertex and name it the central vertex.
2. For each clause $c_j$, we create 2 vertices (top and bottom) connected by an (undirected) edge. The top vertex is connected to the central vertex, and the bottom vertex is connected to $|c_j|$ additional vertices, where $|c_j|$ is the number of literals in clause $c_j$. Each of the additional $|c_j|$ vertices is the start vertex of the corresponding literal agent

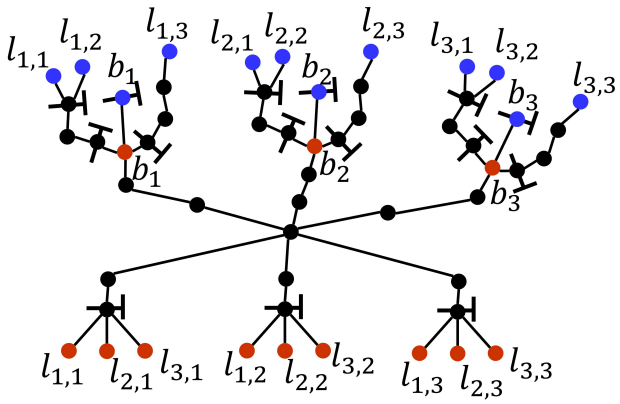


Figure 4: TMAPF$_s$ construction example for the 2P1N-SAT instance $\langle\{x_1, x_2, x_3\}; \{(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)\}\rangle$. The red vertices are the start vertices and the blue are the goal vertices. The $\dashv$ sign encapsulates a delay gadget $dg$. Each bottom $dg$ is $dg_{win}(2, 1)$. Each WaT gadget contains only $dg_{cap}$ (from left to right): $dg_{cap}(16, 2), dg_{cap}(6, 2), dg_{cap}(17, 0), dg_{cap}(7, 1)$.

(or its negation) that is set in the current clause. Lastly, the bottom vertex is a passing vertex of a Window Mode delay gadget $dg_{win}(m-1, 1)$. This is illustrated at the bottom of Figure 4. The $\dashv$ sign on the bottom vertex encapsulates a delay gadget $dg_{win}$.

3. For each variable $x_i$, we create a modified wings and tail gadget (illustrated at the top of Figure 4). We construct a chain of three vertices (top, middle and bottom) and call it the *tail*. The bottom vertex of this tail is explicitly connected to the central vertex to allow literals to exit the bottleneck. Then, we construct two chains of three vertices (top, middle and bottom) and call each chain the *base* of the (right or left) wing. The top vertex of the tail is connected to the bottom vertex of the left and right base vertices. We connect a new vertex (the feather of the wing) to the top base vertex for each literal occurrence that represents the $x_i$ variable in the following way: two vertices connect to the top vertex of the left base (representing the 2 unnegated occurrences). One vertex is connected to the top vertex of the right base.
4. For each left wing base, the bottom vertex is the passing vertex of a Capacity Mode delay gadget $dg_{cap}(m+3, 2)$ and the top vertex is the passing vertex of another $dg_{cap}(3n+7, 2)$.
5. For each right-wing base, the bottom vertex is the passing vertex of a Capacity Mode delay gadget $dg_{cap}(m+4, 1)$.
6. For each variable $x_i$, as in the Fuel reduction, we add a blocking agent $b_i$ and set the start vertex of $b_i$ to be the top vertex of the tail of the corresponding wings and tail gadget.
7. For each block agent $b_i$, we create a capacity delay gadget $dg_{cap}(3n+8, 0)$ and connect its passing vertex to the start vertex of $b_i$. We set the passing vertex of this $dg_{cap}$ to also be the goal vertex of $b_i$.
8. Lastly, let $\Phi = n(3n+9) + \frac{3n}{2}(3n+19) + m(m-1)(2m-1) + n(m+3)(2m+8) + n(m+4)(2m+9) + n(3n+7)(6n+16) + n(3n+8)(6n+16)$. We set $\beta = \Phi$.

Figure 4 shows a construction of a TMAPF$_s$ instance.

**Cost Bounds.** Based on Lemma 1 and the defined construction, we establish the following optimal cost bounds:

1. The block agent $b_i$ cannot reach its goal (and stay there indefinitely) until all the delay agents cleanly pass the passing vertex in the $dg_{cap}$ component containing agent $b_i$'s goal. Thus, the smallest sum of costs that the $n$ block agents $b_i$ can achieve is $n(3n+8+1) = n(3n+9)$.
2. The $dg_{cap}$ gadgets that house the goal vertex of a block agent $b_i$ allow their delay agents to move uninterrupted to their goals, as no agent needs to pass through this gadget (except for $b_i$ after all delay agents have passed). Thus, the total sum of costs for $n$ such target-blocking gadgets is $n(3n+8)(6n+16)$.
3. The first time any literal agent can reach the central vertex is at timestep 3. Since all literal agents must pass through the central bottleneck, they must pass it one by one, forcing subsequent agents to wait incrementally. Additionally, the shortest path length of each literal agent is 10. Thus, the minimum sum of costs that the literal agents can collectively achieve is $10 + 11 + \ldots + (3n+8) + (3n+9) = \frac{3n}{2}(10 + 3n + 9) = \frac{3n}{2}(3n+19)$.
4. Since each $dg_{win}$ gadget located next to the start vertices of the literal clauses has exactly $e_{front} = 1$, exactly $m$ literal agents (strictly one from each clause) can sprint through the window at timestep 1 without increasing the $dg_{win}$ cost. These first $m$ representative agents will seamlessly occupy the central vertex from timestep 3 until timestep $m+2$. At timestep $m+1$, agent $da_1$ of the respective window gadgets will clear the passing vertex, maintaining perfect stride for the remaining literal agents to safely tail the initial $m$ representatives from timestep $m+3$ to timestep $3n+2$. Thus, the sum of costs of each clause gadget is optimally $(m-1)(2m-1)$, and for $m$ gadgets the total sum of costs is $m(m-1)(2m-1)$.
5. The bottom $dg_{cap}$ gadget in the right wing permits at most one external crossing without increasing its cost, while the corresponding gadget in the left wing permits at most two. These capacities allow either assignment choice. If $b_i$ initially blocks the right wing, it uses the right-wing slots and the two positive literal agents use the left-wing slots. If $b_i$ initially blocks the left wing, it uses one left-wing slot and the negative literal agent uses the right-wing slot. Operating essentially as capacity filters, their optimal uninterrupted sum of costs is $n(m+3)(2m+8)$ and $n(3n+7)(6n+16)$ for the bottom and top base vertices of the left wing, respectively. For the right wing, the optimal sum of costs is $n(m+4)(2m+9)$ for the bottom base vertex.

**Polynomial Construction.** The reduction can also be computed in polynomial time. Each delay gadget $dg(d, e_{back}, e_{front})$ contains $\mathcal{O}(d + e_{back} + e_{front})$ vertices and agents. All delay-gadget parameters used in the construction are linear in $n$ or $m$, and the construction contains

$\mathcal{O}(n+m)$ such gadgets. Consequently, the resulting number of vertices and agents is $\mathcal{O}(n^2+nm+m^2)$. The bound $\beta = \Phi$ is likewise computable in polynomial time.

**2P1N-SAT $\Rightarrow$ TMAPF$_s$.** To prove this direction, we need to show that if the 2P1N-SAT formula is satisfiable, then the constructed TMAPF$_s$ instance has a solution with cost $\leq \beta(= \Phi)$. Let $\widetilde{x}_i$ be the assignment for variable $x_i$ in the 2P1N-SAT solution. If $\widetilde{x}_i$ is true (resp., false), then let block agent $b_i$ first block the right (resp., left) middle base vertex of the corresponding wing. The initial movement of each block agent uses one available capacity slot in the bottom delay gadget of its chosen wing. More precisely, the schedule has the following phases:

1. At timestep 1, $b_i$ moves from the tail into the bottom base vertex of the selected wing.
2. At timestep 2, $b_i$ moves to the corresponding middle base vertex and remains pinned there through timestep $m+5$.
3. At timestep $m+6$, the last delay agent $da_1$ vacates the passing vertex of the uninterrupted bottom-base delay gadget. Since following movements are permitted, $b_i$ can simultaneously return to the bottom base vertex.
4. At timestep $m+7$, $b_i$ moves from the bottom base vertex into the adjacent delay gadget. This vacates the previously blocked wing and allows the remaining literal agents to reach their goals.
5. After the remaining literal agents have passed, $b_i$ exits the delay gadget and proceeds to its goal. At timestep $3n+9$, the block agents reach their goals simultaneously.

The base vertices of the unblocked left (resp., right) wing are thus free—having exactly the required number of empty vertices in the delay gadgets to avoid interruption—allowing all corresponding literal agents to pass to their goals one at a time. Since every clause contains at least one true literal, the respective literal agents can selectively pass the delay gadget at timestep 1 and reach their unblocked goals without interrupting any $dg$ along their optimal paths (excluding standard sequential wait actions). Because none of the permitted crossings increases the delay agents' sum of costs, and the first $m$ literal agents occupy the central vertex from timestep 3 to timestep $m+2$, the resulting schedule achieves the theoretical minimum sum of costs. Thus, applying the previously established cost calculations, the total cost is exactly $\beta = \Phi$.

**2P1N-SAT $\Leftarrow$ TMAPF$_s$.** If the TMAPF$_s$ construction has a solution with cost $\leq \beta(= \Phi)$, then the 2P1N-SAT formula must be satisfiable. From our earlier bounds, we know that all agents in the delay gadgets and all literal agents must achieve their absolute minimum possible sum of costs to meet the $\Phi$ threshold. We now establish that each block agent is forced to stably block exactly one wing while waiting for the first $m$ literals to reach their goals; this enforced constraint corresponds to a consistent variable assignment.

To permit literal agents to reach their goals, the block agents must initially move to one of the middle base vertices in their respective wings. These middle base vertices are the only locations where a block agent can passively wait without penalty while the first $m$ literal agents pass through. Crucially, if a block agent has moved to the middle vertex of the left (resp., right) wing, it cannot move to the opposite middle vertex during the first $m+6$ timesteps, because the bottom delay gadget on the opposite wing lacks sufficient empty vertices to absorb the interruption. Furthermore, the block agent cannot advance to the top base vertex (or beyond), as doing so would directly intercept and unnecessarily penalize the top delay gadget.

Thus, throughout the initial phase, each block agent $b_i$ remains pinned to the middle base vertex of one of its two wings. If $b_i$ is pinned to the right (resp., left) wing, set $\widetilde{x}_i =$ true (resp., false). During this phase, at least $m$ literal agents pass through the central bottleneck, including at least one representative from each clause. Since each block agent remains pinned to a single wing, the literals represented by these agents are consistent: they cannot include both a literal and its negation. Therefore, the extracted assignment satisfies every clause.

□

## Conclusion and Discussion

In this work, we addressed the computational complexity of optimizing Fuel and SOC for MAPF on tree topologies (TMAPF). Despite trees being the simplest cycle-free connected graphs, we proved that finding optimal schedules for both objectives remains strictly NP-hard. This closes a theoretical gap, complementing recent findings on Makespan intractability on trees. Our results demonstrate that the hardness of optimal MAPF stems from tight collision constraints and bottleneck resolution, rather than routing complexity or alternative paths found in general cyclic graphs.

Conversely, we identified a highly relevant tractable subproblem: determining if a collision-free schedule exists when all agents are restricted to their shortest paths can be solved in polynomial time. This boundary holds practical value for state-space search solvers like CBS. By applying this feasibility check, solvers can rapidly identify unsolvable zero-detour configurations. This allows them to bypass the massive, symmetric search spaces generated by zero-cost wait actions and safely increment cost lower bounds.

The intractability of optimal TMAPF implies that purely topological restrictions are insufficient to guarantee polynomial-time solvability for standard objectives. Consequently, future algorithmic research on constrained environments—such as dead-end rail networks or container rearrangement—should pivot toward approximation algorithms or bounded-suboptimal solvers.

Several promising directions remain open for future work. First, integrating our polynomial-time feasibility check as a fast pruning heuristic into state-of-the-art solvers offers a clear path to practical performance gains. Second, theoretical investigations should explore the parameterized complexity of TMAPF. Analyzing complexity bounds based on specific tree parameters—such as maximum junction degree, graph diameter, or the number of bottleneck chains—could reveal practical subclasses that scale efficiently in real-world infrastructure.

## Acknowledgements

The work of Shahaf Shperberg and Ariel Felner was supported by the Israel Science Foundation (ISF) grant #909/23. This work was also supported 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, and by BSF grant #2024614 awarded to Shahaf Shperberg. The work of Dor Atzmon was supported by the Israel Science Foundation (ISF) grant #1511/25. 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.

## References

Arnborg, S.; Lagergren, J.; and Seese, D. 1991. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2): 308–340.

Atzmon, D.; Diei, A.; and Rave, D. 2019. Multi-train path finding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 125–129.

Auletta, V.; Monti, A.; Parente, M.; and Persiano, P. 1999. A linear-time algorithm for the feasibility of pebble motion on trees. *Algorithmica*, 23(3): 223–245.

Auletta, V.; and Persiano, P. 2001. Optimal pebble motion on a tree. *Information and Computation*, 165(1): 42–68.

Diestel, R. 2017. *Graph Theory*. Heidelberg: Springer-Verlag, 5th edition.

Dresner, K.; and Stone, P. 2008. A multiagent approach to autonomous intersection management. In *Journal of artificial intelligence research*, volume 31, 591–656.

Felner, A.; and Stern, R. 2026. Blue-Sky: Multi-Agent Path Finding with Unassigned Agents (MAPFUA).

Fioravantes, F.; Knop, D.; Křištan, J. M.; Melissinos, N.; and Opler, M. 2023. Exact Algorithms and Lowerbounds for Multiagent Pathfinding: Power of Treelike Topology. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 11477–11484.

Fioravantes, F.; Knop, D.; Křišťan, J. M.; Melissinos, N.; Opler, M.; and Vu, T. A. 2025. Solving multiagent path finding on highly centralized networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 23186–23193.

Gabay, N.; Morag, J.; Felner, A.; and Stern, R. 2026. The Reachability Objective in Multi-Agent Path Finding. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems*, 1165–1173.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Geft, T.; and Halperin, D. 2022. Refined hardness of distance-optimal multi-agent path finding. *arXiv preprint arXiv:2203.07416*.

Han, S. D.; Stiffler, N. M.; Bekris, K. E.; and Yu, J. 2018. Efficient, high-quality stack rearrangement. *IEEE Robotics and Automation Letters*, 3(3): 1608–1615.

Hanou, I. K.; de Weerdt, M. M.; and Mulderij, J. 2023. Moving trains like pebbles: A feasibility study on tree yards. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, 482–490.

Idgar, O.; Atzmon, D.; and Felner, A. 2026. LaCAM* Variants for Minimizing Makespan in Multi-Agent Path Finding.

Kornhauser, D.; Miller, G.; and Spirakis, P. 1984. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium on Foundations of Computer Science*, 241–250. IEEE.

Koyfman, D.; Atzmon, D.; Shperberg, S.; and Felner, A. 2025. Minimizing Fuel in Multi-Agent Pathfinding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 18, 83–91.

Krontiris, A.; Luna, R.; and Bekris, K. 2013. From feasibility tests to path planners for multi-agent pathfinding. In *Proceedings of the international symposium on combinatorial search*, volume 4, 114–122.

Maliah, A.; Atzmon, D.; and Felner, A. 2025. Minimizing Makespan with Conflict-Based Search for Optimal Multi-Agent Path Finding.

Masehian, E.; and Nejad, A. H. 2009. Solvability of multi robot motion planning problems on trees. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5936–5941. IEEE.

Masehian, E.; and Nejad, A. H. 2010. A hierarchical decoupled approach for multi robot motion planning on trees. In *2010 IEEE International Conference on Robotics and Automation*, 3604–3609. IEEE.

Morag, J.; Gabay, N.; Koyfman, D.; and Stern, R. 2025. Should Multi-Agent Path Finding Algorithms Coordinate Target Arrival Times? In *Proceedings of the International Symposium on Combinatorial Search*, volume 18, 231–235.

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

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the international symposium on combinatorial search*, volume 10, 151–158.

Surynek, P. 2010. An optimization variant of multi-robot path planning is intractable. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, 1261–1263.

Švancara, J.; Tignon, E.; Barták, R.; Schaub, T.; Wanko, P.; and Kaminski, R. 2023. Multi-Agent Pathfinding with Predefined Paths: To Wait, or Not to Wait, That Is the Question. In *Proceedings of the International Symposium on Combinatorial Search*, volume 16, 185–186.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of autonomous vehicles in factories. *AI magazine*, 29(1): 9–19.

Yoshinaka, R. 2005. Higher-order matching in the linear lambda calculus in the absence of constants is NP-complete. In *International Conference on Rewriting Techniques and Applications*, 235–249. Springer.

Yu, J.; and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.