# Hardness of Multi-Agent Path Finding on Trees: A Unified Approach

**Tzvika Geft**
Rutgers University, New Brunswick, NJ, USA

## Abstract

This paper presents a simple framework that settles the complexity of Multi-Agent Path Finding (MAPF) on trees across standard objectives – distance, makespan, and flowtime – for both labeled and colored variants. In MAPF, agents occupy the vertices of a graph and must move to target vertices without collisions while optimizing a given objective. In the labeled case, the agents are distinct and have respective targets; in the colored case, agents of the same color are interchangeable. While many MAPF variants are known to be intractable, several basic cases on trees have remained open. We prove NP-hardness on trees for both labeled and 2-colored MAPF under all three objectives. In particular, we resolve the classical Pebble Motion problem, where one pebble moves at a time to an adjacent empty vertex and the goal is to minimize the total number of moves. Despite being one of the most basic discrete motion models, its complexity on trees had remained open for several decades. Moreover, for colored Pebble Motion, we give the first hardness result on any graph class, already with two colors, which is tight.

All of these results are established through the hardness of Stack Rearrangement, itself posed as an open problem, which asks to optimally rearrange items stored in stacks, and which we also prove to be NP-hard. Notably, the connection to stacks yields hardness already on very simple trees – subdivided stars – across all problems. Together, these results reveal a common tractability barrier that permeates several fundamental motion models, thereby unifying and strengthening prior hardness results.

**2012 ACM Subject Classification** Theory of computation → Problems, reductions and completeness

**Keywords and phrases** Pebble Motion on Trees, Coordinated Motion Planning, Multi-Agent Path Finding, Stack Rearrangement, NP-Hardness

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2026.159

## 1 Introduction

*Pebble Motion on Graphs* (PMG), a generalization of the classical $(n^2-1)$-puzzle, asks for a shortest sequence of moves that takes a collection of labeled pebbles on the vertices of a graph from a start configuration to a target configuration, where a move relocates a single pebble to an adjacent empty vertex. Its parallel-motion variant, *Multi-Agent Path Finding* (MAPF), also known as *Coordinated Motion Planning*, allows many pebbles – now called *agents* – to move simultaneously, and is typically cast as an optimization problem under objectives such as *makespan*, the number of time steps until all agents reach their goals, and *flowtime* (or *sum-of-costs*), the sum of all the agents' individual arrival times.

In some cases, agents (or pebbles) can be naturally grouped into teams of equivalent agents rather than viewing each agent as unique. In the *$k$-colored* variant of PMG and MAPF [19, 6, 27], agents are partitioned into $k$ teams, each with its own set of targets, and any assignment of agents within a team to its targets is valid provided all targets are eventually occupied. The case where $k$ is the number of agents corresponds to the original labeled problem, while the $k=1$ case is also known as *unlabeled* or *anonymous*.

Together, PMG, MAPF, and their variants serve as a prominent abstraction for multi-robot path planning and have driven a large body of algorithmic work motivated by warehouse automation and related applications [28, 26, 11, 23, 14].


The *feasibility* problem for PMG – finding any solution or reporting that none exists – is classically solvable in polynomial time for the labeled case [21], as well as the colored case [19]. The feasibility algorithms can produce solutions cubic in $n$, the number of vertices, while practical applications call for high quality solutions. In turn, the complexity of finding optimal solutions has received significant attention, with hardness results on general graphs, planar graphs, and 2D grids [18, 25, 29, 32, 5, 10, 17, 16]. However, the picture on trees, the focus of this work, has not yet been settled.

Several recent works investigate motion on trees or tree-like graphs. Ardizzoni et al. [3] give an algorithm that finds feasible (not necessarily optimal) *Pebble Motion on Trees* (PMT) solutions with $O(Nnc + n^2)$ moves, for $n$ vertices, $N$ pebbles, and $c$ being the maximum corridor length in the tree, refining the classical $O(n^3)$ bound [21]. Nakamigawa and Sakuma [24] establish sharper upper bounds on the length of shortest PMG solutions, both on trees and on general graphs, in terms of graph diameter and cycle structure. MAPF variants have been recently approached through parameterized complexity, where treewidth – a measure of how tree-like a graph is – has been used with other parameters to develop parameterized algorithms [12, 8, 9]. On the hardness side, Fioravantes et al. [12, 13] prove NP-hardness of makespan-optimal MAPF on trees under various structural restrictions while Aichholzer et al. [2] recently resolved the complexity of token swapping on trees – a structurally distinct setting where a move swaps two adjacent pebbles, which is not allowed in PMG or MAPF.

**Stack Rearrangement.** One example of a motion planning problem that can be abstracted using PMT is *Stack Rearrangement* (SR). The problem asks to find the shortest sequence of pop-and-push moves that reconfigures items distributed across a collection of bounded-capacity stacks to a target arrangement. That is, each move removes an item from the top of one stack and places it at the top of another stack that has available capacity. Han et al. [20] introduce the problem to model robotic rearrangement in stack-like containers (e.g., stackable parking or retail shelves). They observe that SR can be cast as a PMT problem and conjecture optimal SR to be intractable, but leave its complexity open.

## Our Results

We present a simple framework, rooted in the hardness of Stack Rearrangement, that settles the complexity of MAPF on trees across the standard objectives of total distance, makespan, and flowtime, for both labeled and colored variants (see Section 2 for formal definitions). All variants are shown to be NP-hard on subdivided stars,[1] among the simplest nontrivial trees, while all colored variants are shown to be NP-hard already for $k = 2$. To our knowledge, this is the first work to jointly establish the intractability of both labeled and colored MAPF across these three common objectives. The results are summarized in Table 1.

On trees, MAPF under the total distance objective, which minimizes total edge traversals, corresponds to the PMT optimization problem. In this case the framework closes two particularly notable gaps:

- **Pebble Motion on Trees (PMT).** The complexity of optimal PMT has been implicitly open since the 1980s, when work on PMG emerged [21]. It was posed by Auletta et al. [4], which provides linear-time feasibility testing on trees, and recently re-raised by Deligkas et al. [8].

[1] A *subdivided star* is a graph consisting of any number of paths all joined at a single endpoint vertex.

**Table 1** Comparison to representative prior MAPF NP-hardness results that capture the state of the art. Each entry specifies the optimization objective, the labeled vs. colored variant, and graph class. Note that the table is not a complete survey and that works subsumed by the shown results at the graph type level are generally omitted.

| Graph type | Work | Labeled | | | Colored ($k = 2$ unless specified) | | |
|---|---|---|---|---|---|---|---|
| | | Distance | Makespan | Flowtime | Distance | Makespan | Flowtime |
| General | [32] | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Planar | [30] | ✓ | ✓ | ✓ | | ✓ | ✓ |
| 2D subgrids | [17] | ✓ | | | | | |
| | [16] | | ✓ | ✓ | | ✓ | ✓ |
| Trees | [12, 13] | | ✓ | | | $k = 6$ | |
| **Subdivided stars** | **This work** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Max. degree 3 trees** | | ✓ | | | ✓ | | |

- **Colored PMG.** The complexity of optimal colored PMG has been open for *any graph class* since at least Goraly and Hassin [19], who showed that feasibility is in P. Prior attempts, such as the systematic studies of the complexity of MAPF [32, 30], which consider standard objectives on general and later planar graphs, have left this case open. This work provides the first hardness result for colored PMG, already on trees and for $k = 2$ (Theorem 10).

Beyond subdivided stars, both the labeled and 2-colored PMT hardness results hold on comb-like trees of maximum degree 3 that naturally embed as 2D subgrids. The result for labeled PMT therefore recovers and strengthens the previously tightest 2D grid-based NP-hardness result [17].

On the time-optimal side, we obtain the first NP-hardness result for flowtime on trees, as well as the first NP-hardness of 2-colored MAPF on trees for both makespan and flowtime, which was only known for more general graph classes. Existing hardness results on trees are known only for the makespan objective, under various structural restrictions: maximum degree 5 [12], 11 leaves [13], or 9 internal vertices [13]. Sharpening the makespan landscape, we simultaneously bound depth to 3 and the number of branching vertices, i.e., vertices of degree at least 3, to 1.

For all considered objectives, the problems are efficiently solvable in the unlabeled (1-colored) case via a reduction to network flow [31]. Thus, the 2-colored hardness results form the exact tractability threshold.

**Stack rearrangement as a unifying source of hardness.** All of the above results are established through reductions from Stack Rearrangement, whose own complexity we also settle:

- Labeled Stack Rearrangement is NP-hard, even for stacks of depth 3 (Theorem 6).
- 2-colored Stack Rearrangement is NP-hard, even for deciding whether every item can be moved at most once (Theorem 7).

Here too hardness for two colors is tight, as the unlabeled case is straightforward to solve efficiently.

The connection to stacks yields hardness on subdivided stars across MAPF variants: each stack becomes a path from a shared central vertex, and the serial bottleneck at that vertex tightly couples the stack and tree formulations.

As NP membership holds for all studied problems [21, 19, 20], we only prove NP-hardness.

**Concurrent work.** During the review of this paper, we learned of independent work establishing hardness for labeled MAPF on trees under the total distance and flowtime objectives [22]. These concurrent results use different techniques and do not establish hardness on subdivided stars.

## 2 Problem Definitions and Notation

Let $G = (V, E)$ be a connected undirected graph. Let $d(u, v)$ denote the shortest-path distance between vertices $u$ and $v$ in $G$. We consider $N \leq |V|$ uniquely labeled pebbles, also called agents, $p_1, \ldots, p_N$. The pebbles' locations are specified by a *configuration* $S$, which is an injective mapping assigning each pebble $p$ to a unique vertex $S(p) \in V$.

▶ **Definition 1** (Pebble Motion on Graphs (PMG)). *Let $G = (V, E)$ be a connected undirected graph. A* move *consists of transferring a single pebble from its current vertex to an adjacent unoccupied vertex. Given two configurations $S$ (start) and $D$ (target/goal), the PMG problem asks for a minimum-length sequence of moves that transforms $S$ into $D$.*

▶ **Definition 2** (Pebble Motion on Trees (PMT)). *A PMT instance is a PMG instance $I = (T, S, D)$ where $T$ is a tree.*

▶ **Definition 3** (Multi-Agent Path Finding (MAPF)). *A MAPF instance is given by a graph $G = (V, E)$ and two configurations $S$ and $D$ as in PMG. Time is discretized and at each time step an agent can either wait at its current vertex or move to an adjacent vertex: A* timed path *for an agent $p$ is a sequence of vertices $(u_0, u_1, u_2, \ldots)$ with $u_0 = S(p)$, where each consecutive pair is either identical (a* wait*) or adjacent in $G$. A MAPF* solution *is a set of timed paths, one per agent, such that every agent eventually reaches and remains at its goal vertex, and the paths are* conflict-free*: no two agents coincide at a vertex at the same time step (*vertex conflict*), and no two agents traverse the same edge in opposite directions at the same time step (*edge conflict*). In particular, an agent may enter a vertex $v$ in the same time step in which another agent leaves $v$, allowing agents to move in a train-like manner.*

***Optimization objectives.*** *The* completion time *of an agent $p$ is the first time step $\tau_p$ at which $p$ reaches $D(p)$ and stays there. The* makespan *of a solution is $\max_p \tau_p$, i.e., the time it takes the last agent to reach its goal. The* flowtime *(also known as* sum-of-costs*) of a solution is $\sum_p \tau_p$, i.e., the sum of agent completion times. The* total distance *of a solution is the total number of edge traversals over all timed paths.*

**Remark.** As MAPF with total distance objective on trees corresponds to PMT, we generally refer to the latter in the sequel.

We now introduce the stack-based formulation relevant to this work.

▶ **Definition 4** (Labeled Stack Rearrangement (LSR) [20]). *We are given a set of $N$ distinct objects, $\{o_1, o_2, \ldots, o_N\}$, and $w + 1$ available stacks, each with a maximum capacity (depth) of $d$ such that $N \leq wd$. In this problem, objects can be moved from the top of one stack to the top of another, via a* pop-and-push action*, with the constraint that objects cannot be*

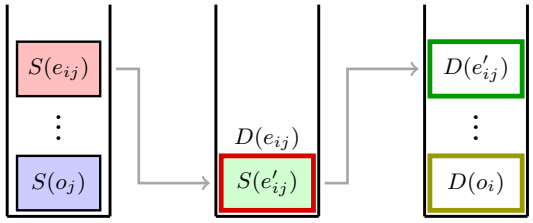



**Figure 1** The dependency created by an edge $(i, j)$. (Left) Start stack for $o_j$; (middle) the stack for $(i, j)$; (right) the goal stack for $o_i$.

*moved from an empty stack or to a full stack. An object's position in the stacks is defined by a 2-tuple, indicating its stack and its depth within that stack. A* configuration *specifies the locations of all objects in the stacks. Given an initial configuration, the objective of the LSR problem is to rearrange the objects to match a goal configuration using the minimum number of pop-and-push moves.*

We also consider the *colored* variant of each of the labeled problems defined above. In the colored variant, objects are partitioned into color classes and are interchangeable within each class. Whenever the terms labeled or colored are not used, we default to the labeled case.

▶ **Definition 5** (Colored variants: $k$-colored PMG, $k$-colored MAPF, $k$-colored Stack Rearrangement ($k$-colored SR)). *In the* $k$-colored *variant, the objects (or pebbles/agents) are partitioned into $k$ color classes. The goal specifies, for each target position, only the required color, and any object of that color may occupy it.*

## 3 Hardness of Stack Rearrangement

We begin with the hardness of Stack Rearrangement variants, which will serve as base problems for all the other reductions.

▶ **Theorem 6.** *Labeled Stack Rearrangement (LSR) is NP-hard, even when restricted to stacks of depth 3.*

**Proof.** To establish NP-hardness, we perform a reduction from the Minimum Feedback Arc Set (MFAS) problem, which asks to find the smallest set of edges in a directed graph whose removal results in a directed acyclic graph (DAG). We use the fact that MFAS remains NP-hard for maximum degree 3 [15]; that is, we can assume the in-degree and out-degree of each vertex in $G$ are at most 2.

Given a directed graph $G = (V, E)$ with vertices $V = \{1, 2, \ldots, n\}$ and $m$ edges, we construct a corresponding LSR instance $I := I(G)$ as follows: For each vertex $i$ in $G$, we introduce a corresponding object $o_i$. For each edge $e = (i, j)$ in $G$, we create two objects denoted by $e_{ij}$ and $e'_{ij}$. For each $o_i$, we designate a start stack where $o_i$ is initially at the bottom and a goal stack where it needs to end up at the bottom. For an edge $(i, j)$, we place the start position of $e_{ij}$ in the start stack of $o_j$, anywhere above $o_j$, and set its goal to be at the bottom of its own stack. That is, the order of the start positions of the $e_{ij}$'s is arbitrary but they are all initially above $o_j$. The start position of $e'_{ij}$ coincides with the goal of $e_{ij}$, and the goal of $e'_{ij}$ is in the goal stack of $o_i$, anywhere above $o_i$'s goal. See Figure 1. Note that an edge $(i, j)$ creates a dependency in an LSR solution with a lower bound cost (where every object is moved once), where $o_i$ must be moved to its goal before $o_j$.

Finally, add $m$ empty stacks. Therefore, the total number of objects is $n + 2m$, and there are $2n + 2m$ stacks. Since every vertex in $G$ has in-degree and out-degree at most 2, a stack depth of 3 suffices.

We now prove the correctness of the reduction. We show that $G$ has a feedback arc set $E' \subseteq E$ with $\ell$ edges if and only if $I$ has a solution with $n + 2m + \ell$ moves.

Let $G'$ be the DAG obtained by removing the edges in $E'$ from $G$. Similarly, let $I'$ denote the instance obtained by removing all objects corresponding to $E'$ from $I$. We can use the topological ordering of $G'$ to solve $I'$ with exactly one move per object as follows. For each vertex $i$ according to the topological ordering:

1. For each incoming edge $(j, i)$, move $e_{ji}$. These objects are moved one by one per the ordering imposed by their start positions in the start stack of $o_i$.
2. Move $o_i$.
3. Then, for each outgoing edge $(i, j)$, move $e'_{ij}$ according to the ordering in the goal stack of $o_i$.

The moves in step 1 are valid because when we handle vertex $i$, all the $e'_{ji}$'s have already been moved as part of handling step 3 in previous vertices. Moves in steps 2 and 3 are valid as no object blocks them.

We can now solve $I$, the original instance, based on the solution above. In step 1, any $e_{ji}$ that corresponds to an edge in $E'$ is moved to an empty stack instead of its goal (thereby breaking the dependency of the edge). Finally, after performing all the moves above, we can move all such $e_{ji}$'s to their goals one by one. This solution introduces $\ell$ additional moves beyond the lower bound for the number of moves, resulting in a total of $n + 2m + \ell$ moves, as required.

Conversely, suppose we have a solution with $n + 2m + \ell$ moves. Let $E'$ denote the set of edges for which either $e_{ij}$ or $e'_{ij}$ moves more than once. We have $|E'| \leq \ell$. We claim that $E'$ is a feedback arc set for $G$: Order the vertices according to the order in which the corresponding objects make their final moves. We claim that this results in a topological ordering of $G'$, the graph where the edges of $E'$ are removed. Indeed, let $(i, j)$ be an edge of $G'$. Both $e_{ij}$ and $e'_{ij}$ move exactly once. Since $e'_{ij}$ reaches its goal above $o_i$, it must move after the final move of $o_i$. Moreover, $e'_{ij}$ must move before $e_{ij}$, which in turn must move before the final move of $o_j$, since $e_{ij}$ initially lies above $o_j$. Thus $o_i$ makes its final move before $o_j$, i.e., vertex $i$ appears before $j$ in the ordering. Hence the ordering is topological for $G'$, and $E'$ is a feedback arc set. ◀

▶ **Theorem 7.** *2-colored SR is NP-hard, even for deciding whether there is a solution in which every item moves at most once.*

**Proof.** We reduce from 3-Partition: given $3m$ positive integers $a_1, \ldots, a_{3m}$ with each $a_i \in (K/4, K/2)$ and $\sum a_i = mK$, decide whether they can be partitioned into $m$ triples each summing to $K$. This problem is strongly NP-complete [15].

**Construction.** Given a 3-Partition instance, we construct the following stacks, as shown in Figure 2. We use red and blue as the two colors:

- For each $a_i$, create an *item stack*, containing $a_i$ red items initially and requiring $a_i$ blue items in the target configuration.
- Create $m - 1$ *checkpoint stacks*, each containing $K$ blue items initially and requiring $K$ red items in the target configuration.
- Create one additional *red target stack*, initially empty and requiring $K$ red items in the target configuration.
- Create one *filler stack*, initially containing $K$ blue items and empty in the target configuration.

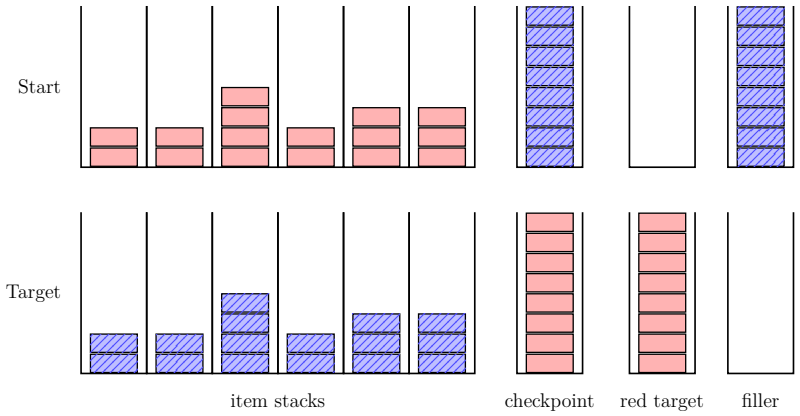


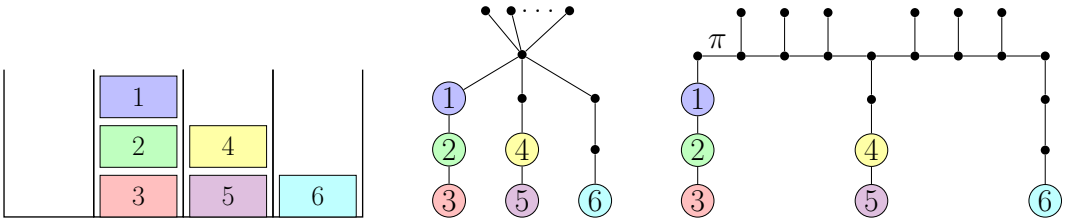

**Figure 2** The 2-colored SR construction for the 3-PARTITION instance with $K = 8$ and items $(2, 2, 4, 2, 3, 3)$: $S$ with $D$ below. Blue items are shown with a hatching pattern.

The construction uses $3m + (m-1) + 2 = 4m + 1$ stacks of depth $K$. It contains $mK$ red items and $mK$ blue items. Since 3-PARTITION is strongly NP-complete, the construction has polynomial size.

**Structural observation.** We say that a stack is *cleared* when every item originally in that stack has moved out of it. In any solution where each item moves at most once, every item stack and every checkpoint stack must be cleared before any item is placed in it. Indeed, any incoming item placed in such a stack earlier would permanently trap wrong-colored items that need to be removed. We therefore track available red and blue target capacity using two counters: let $r$ (resp. $b$) denote the number of currently accessible red (resp. blue) target positions. Initially $r = K$ (due to the empty red target stack) and $b = 0$. Clearing item stack $i$ decreases $r$ by $a_i$ and increases $b$ by $a_i$; clearing a checkpoint stack decreases $b$ by $K$ and increases $r$ by $K$. At any point in a solution we have $r \geq 0$ and $b \geq 0$.

**Correctness.** Given a valid 3-partition $S_1, \ldots, S_m$, we construct a one-move-per-item SR solution that operates in $m$ phases: For $j = 1, \ldots, m$, clear item stacks corresponding to $S_j$, then clear a checkpoint stack, except for $j = m$ where the filler stack is cleared instead. At the start of each phase $r = K, b = 0$ and one target stack is available for red items: the red target stack for $j = 1$, or a checkpoint stack otherwise. Clearing the $S_j$ stacks brings us to $r = 0, b = K$ as the cleared stacks can now accept blue items. Clearing the checkpoint resets the state to $r = K, b = 0$.

Suppose a valid 2-colored SR solution exists in which every item is moved once. Let $Q_1, \ldots, Q_{m-1}$ denote the checkpoint stacks in the order in which they are cleared. Let $S_1$ be the set of item stacks cleared before the first checkpoint stack $Q_1$ is cleared, and let $q = \sum_{i \in S_1} a_i$. Before $Q_1$ is cleared, there are exactly $K$ red target positions available, which are in the initially empty red stack, so $q \leq K$. Since the $K$ blue items originating in $Q_1$ each move exactly once, they must occupy $K$ blue target positions in item stacks cleared before this time. Hence $q \geq K$. Therefore $q = K$, and since each $a_i \in (K/4, K/2)$, we have $|S_1| = 3$ and so $S_1$ corresponds to a valid triple. Furthermore, $q = K$ implies that when $Q_1$ becomes cleared, every checkpoint and item stack other than $S_1 \cup \{Q_1\}$ remains as in the initial configuration. The red target-only stack is filled to its capacity and $Q_1$ is now empty, and so the initial target capacity state $r = K, b = 0$ is restored.




**(a)** An LSR configuration **(b)** Subdivided star **(c)** Comb-like graph

**Figure 3** The reductions in Theorem 8. A 4-stack LSR configuration (a) and the corresponding configurations in the subdivided-star (b) and comb-like graph (c) PMT instances. The buffer leaves are the topmost vertices.

The same argument repeats for each subsequent $Q_j$: the item stacks cleared after $Q_{j-1}$ is cleared and before $Q_j$ is cleared correspond to a triple summing to $K$. Finally, the number of items in the last 3 item stacks, cleared after $Q_{m-1}$ is cleared, must be $K$, and so we obtain $m$ triples forming a valid 3-partition. ◀

## 4 Hardness of distance-optimal motion

This section reduces Stack Rearrangement to PMT variants and establishes the hardness of the latter.

▶ **Theorem 8.** *PMT is NP-hard, even when restricted to (i) subdivided stars of depth 3 or to (ii) trees of maximum degree 3.*

**Proof.** We perform reductions from LSR with a stack depth of 3, which is NP-hard by Theorem 6. See Figure 3.

For (i), an LSR instance is converted to a subdivided star by treating each stack as a path with length equal to the stack's depth and then joining the top vertices of the stacks to a single root vertex $c$. Each PMT vertex on a stack path corresponds to one LSR stack position. By Theorem 6 we can assume that there are as many buffer (initially empty) stacks as objects, so we add $N$ empty buffer stacks, which are leaves emanating from $c$.

Let $L$ denote the sum over all pebbles of the shortest-path distance from the pebble's start to its goal. Then $L$ is a lower bound on the number of moves of any PMT solution. We claim that the LSR instance admits a solution with at most $\ell$ relocations, i.e., moves to a temporary stack, if and only if the PMT instance admits a solution with at most $L + 2\ell$ moves.

Given an LSR solution with $\ell$ relocations, each non-relocated object corresponds to a pebble moved along its shortest path to its goal. Each of the $\ell$ relocated objects corresponds to a pebble that first detours into an (unoccupied) buffer leaf, adding exactly two moves, before proceeding to its goal. The ample buffers guarantee that a buffer is free at the time of each detour. The PMT solution has a total of $L + 2\ell$ moves.

Conversely, suppose we have a PMT solution with $L + 2\ell$ moves. Each move of a pebble from the center vertex $c$ to a stack path corresponds to a single push-and-pop LSR move. Since using a buffer in the PMT instance requires a detour, adding two moves to the motion of a pebble, the resulting LSR solution has at most $\ell$ relocations.

For (ii), we construct a comb-like graph. We use a central path $\pi$ and for each stack, we attach its corresponding stack path to a distinct vertex of $\pi$. For each pebble, we provide a unique empty leaf vertex that it can detour to and incur exactly two extra moves beyond its

shortest path. Specifically, for each pebble $p$, let $u_1$ and $u_2$ be the unique vertices of $\pi$ to which the start and target stack path of $p$, respectively, are attached. Subdivide an arbitrary edge on the subpath of $\pi$ between $u_1$ and $u_2$ by a new vertex $b$, and attach a new leaf to $b$. The construction adds $O(N)$ vertices and has a maximum degree 3.

As in (i), we claim that the LSR instance admits a solution with at most $\ell$ relocations if and only if the PMT instance admits a solution with at most $L + 2\ell$ moves.

It is straightforward to verify that the forward direction is analogous to (i), so we prove only the backward direction. This direction requires more careful analysis since the PMT solution can now result in multiple pebbles that are simultaneously located outside of a stack path.

Suppose we have a PMT solution on the comb graph with $L + 2\ell$ moves. We show it yields a valid LSR solution with at most $\ell$ relocations. Since each detour beyond a shortest path requires at least 2 extra moves, at most $\ell$ pebbles follow a non-shortest path to their goal. Call these pebbles *detoured* and the remaining pebbles *direct*. We present an LSR solution for the direct pebbles that does not involve relocations. Each detoured pebble can be easily incorporated into this LSR solution as an object $o$ that is relocated exactly once, as a post-processing step: Each such $o$ is moved to an empty stack (not shared with any other object) and then to its target position when it becomes available, without disturbing other moves. This accounts for at most $\ell$ relocations.

It remains to show that the direct pebbles yield a valid LSR solution without relocations. We remove all detoured pebbles and their moves from the PMT solution, and call the resulting solution $\mathcal{S}$.

▶ **Lemma 9.** *Let $p$ be a pebble whose motion in $\mathcal{S}$ follows a shortest path $v_1, v_2, \ldots, v_r$ in the comb graph from its start vertex to its goal vertex. Then moves in the solution $\mathcal{S}$ can be rearranged (without changing the total move count) so that $p$ moves from $v_1$ to $v_r$ in $r - 1$ consecutive steps, i.e., with no other pebble moving during this interval. We call such a traversal* atomic*.*

**Proof.** We use induction on $r$. The base case $r = 2$ is trivial. For the inductive step, assume that there is a solution $\mathcal{S}$' where $p$'s traversal of $v_1, \ldots, v_q$ is atomic. In $\mathcal{S}'$, some sequence of moves $M$ by other pebbles occurs between $p$'s arrival at $v_q$ and $p$'s move to $v_{q+1}$. Since $p$ occupies $v_q$ throughout $M$, no move in $M$ crosses $v_q$, so each move lies entirely in one component of $T \setminus \{v_q\}$. This yields a partition of $M$: $M_{\text{next}}$, the moves in the component containing $v_{q+1}$, and $M_{\text{rest}}$, the moves in the remaining component(s).

We modify the solution as follows: perform the moves of $M_{\text{next}}$ right before $p$'s atomic block, then $p$ traverses $v_1, \ldots, v_{q+1}$ atomically, then perform the moves of $M_{\text{rest}}$. Subsequent moves remain the same.

This reordering is valid: First, since $p$ follows a shortest path in a tree, $v_{q+1} \notin \{v_1, \ldots, v_q\}$, so the component of $T \setminus \{v_q\}$ containing $v_{q+1}$ does not contain $v_1, \ldots, v_{q-1}$. The moves of $M_{\text{next}}$ and $p$'s atomic traversal of $v_1, \ldots, v_q$ thus act on disjoint vertex sets, so $M_{\text{next}}$ can be performed before $p$'s atomic block without affecting validity. Second, the moves of $M_{\text{rest}}$ lie in components of $T \setminus \{v_q\}$ not containing $v_{q+1}$, so they do not interact with $p$'s move from $v_q$ to $v_{q+1}$ and can be deferred. ◀

Apply the lemma to each pebble in turn. Processing pebbles in sequence does not invalidate previously established atomic blocks, as any such block within $M$ lies entirely in one component of $T \setminus \{v_q\}$ and is shifted as a unit and never split. The result is a solution consisting of atomic start-to-target traversals, each corresponding to a pop-and-push move in LSR, as required. ◀

▶ **Theorem 10.** *2-colored PMT is NP-hard, even when restricted to (i) subdivided stars or to (ii) trees of maximum degree 3.*

**Proof.** We reduce from 2-colored SR, which is NP-hard by Theorem 7.

For (i), apply the corresponding reduction used in the proof of Theorem 8 to yield a subdivided star. By Theorem 7, it is NP-hard to decide whether a 2-colored SR instance can be solved without relocations. Since a zero relocation SR solution corresponds to a PMT solution with cost exactly $L$ (the shortest-path lower bound), and vice versa, it is NP-hard to decide if a 2-colored PMT instance admits a solution of cost $L$. Notice that any assignment of pebbles to targets always results in the same distance lower bound of $L$.

For (ii), we apply the comb graph reduction used in the proof of Theorem 8, i.e., take $T$ to be a central path $\pi$ to which all the stack paths are connected. The stacks are ordered along $\pi$ as follows: all $3m$ item stacks on the left, then all $m-1$ checkpoint stacks, the empty red target stack, and the blue filler stack on the right. No additional empty stacks are used. It is straightforward to verify that this arrangement of the stacks results in the same distance lower bound $L$ for any valid assignment of pebbles to targets. A zero-relocation SR solution corresponds to a PMT solution of cost $L$. Conversely, a cost-$L$ PMT solution forces every pebble to move via a shortest path, and so we apply Lemma 9 to make each pebble's motion atomic to yield a zero-relocation SR solution. Lemma 9 is valid for the 2-colored case because a solution fixes a unique target and path for each pebble. ◀

## 5 Hardness of time-optimal objectives

In this section, we prove NP-hardness for time-optimal MAPF objectives, namely makespan and flowtime.

▶ **Theorem 11.** *MAPF is NP-hard for the makespan objective, even when restricted to a tree that is a subdivided star of depth 3.*

**Proof.** We reduce from LSR with stack depth 3, which is NP-hard by Theorem 6. Given an LSR instance $I$ with $N$ items, we construct a subdivided star $T$ as in Theorem 8(i): each stack of depth $d$ becomes a path $v_1, \ldots, v_d$, and the vertices $v_1$ of all paths are joined to a single central vertex $c$. In the initial configuration, the items of each stack containing $a$ items are placed at vertices $v_1, \ldots, v_a$, packed toward $c$. The target configuration is defined analogously from the target stack configuration: if a stack contains $a$ items in the target configuration, then those items occupy $v_1, \ldots, v_a$, again packed toward $c$.

We claim that the LSR instance $I$ has a solution with at most $M$ moves if and only if the resulting MAPF instance on $T$ admits a schedule with makespan at most $M+1$.

Suppose $I$ has a solution with $M$ moves. Each SR move, which pops the top item of one stack and pushes it onto another stack, is realized in $T$ by moving the corresponding agent through $c$. We schedule these $M$ transits in consecutive time steps, so that $c$ is occupied at times $2, \ldots, M+1$. More precisely, at each intermediate time step, the agent currently at $c$ moves to the destination stack of the current SR move, while the agent for the next SR move enters $c$ from the source stack of that move. Within each source stack path, when the agent at $v_1$ moves to $c$, the agents at $v_2, v_3, \ldots$ shift one step toward $c$ in the same time step, so that $v_1$ is always occupied for a non-empty stack. Similarly, when an agent moves from $c$ into a destination stack path, the existing agents in that path shift one step away from $c$, so the arriving agent can occupy $v_1$. Thus the first transit enters $c$ at time 1, and the last transit leaves $c$ at time $M+1$. Hence the MAPF schedule has makespan $M+1$.

Suppose the MAPF instance on $T$ admits a solution with makespan $M+1$. Since $c$ is empty initially, there are at most $M$ time steps at which an agent located at $c$ goes to a stack path. Each such time step corresponds to a pop-and-push move of an item in $I$, hence these time steps define at most $M$ SR moves. Along each stack path, agents cannot pass one another, so these induced moves respect the LIFO order. Therefore, we obtain a valid LSR solution with at most $M$ moves. ◀

▶ **Theorem 12.** *2-colored MAPF is NP-hard for the makespan objective, even when restricted to a tree that is a subdivided star.*

**Proof.** We reduce from 2-colored SR, which is NP-hard by Theorem 7, exactly as in Theorem 11. Since we can track moves of the colored case the same way as in the labeled case, the conversions between solutions are identical. ◀

We turn to the flowtime objective, where we begin with hardness for the 2-colored case, where the argument is simpler because the reduction only needs to distinguish zero SR relocations from any relocations. In the labeled case, on the other hand, the reduction must more carefully control the relationship between flowtime and the number of relocations.

▶ **Theorem 13.** *2-colored MAPF is NP-hard for the flowtime objective, even when restricted to a tree that is a subdivided star.*

**Proof.** We reduce from 2-colored SR, which is NP-hard by Theorem 7. From the construction in Theorem 7, we may assume that every item starts in a non-goal stack and that it is NP-hard to decide whether an instance with $N$ items admits a solution using exactly $N$ moves.

Given such a 2-colored SR instance $I$ with $N$ items, we construct a subdivided star $T$ with central vertex $c$, together with start and target configurations, exactly as in Theorem 11.

Set $\Delta \coloneqq \sum_{v \in D} d(c, v)$, where $D$ denotes the set of target vertices in $T$. We claim that $I$ admits a solution with exactly $N$ moves if and only if the resulting MAPF instance on $T$ admits a solution with flowtime of $N(N+1)/2 + \Delta$.

Fix an optimal MAPF solution. For an agent $a$, let $t_a$ denote the timestep at which $a$ reaches $c$ for the last time, and let $\delta_a$ be the distance from $c$ to the target vertex eventually occupied by $a$. After traversing $c$ for the last time, the shortest possible remaining path for $a$ is a descent of length $\delta_a$ into its target stack, so the completion time of $a$ is $t_a + \delta_a$. Hence, the flowtime satisfies $F = \sum_a (t_a + \delta_a) = \sum_a t_a + \Delta$. As the values $\{t_a\}_a$ are distinct positive integers, $F$ is minimized with $F = N(N+1)/2 + \Delta$ if and only if $\{t_a\}_a = \{1, 2, \ldots, N\}$.

Suppose $I$ admits a solution with exactly $N$ moves. Each SR move is converted to a single transit of the corresponding agent through $c$, scheduling the $N$ transits to reach $c$ at consecutive timesteps $1, 2, \ldots, N$. This is done by having all agents in a stack move up simultaneously as the topmost agent leaves. Each agent $a$ descends directly into its target vertex without waiting after visiting $c$, resulting in a flowtime of $N(N+1)/2 + \Delta$.

Suppose the MAPF instance on $T$ admits a solution with flowtime at most $N(N+1)/2 + \Delta$. That is, we have $\{t_a\}_a = \{1, \ldots, N\}$. The equality is only possible if each agent visits $c$ once, which corresponds to exactly one valid move per object in the SR solution. ◀

For the labeled case under the flowtime objective, repeating the reduction above would not work since the order in which relocations occur affects flowtime: earlier relocations correspond to increased agent completion times. The following reduction normalizes such variations by adding many agents that would be delayed by relocations regardless of when they occur in an LSR solution.

▶ **Theorem 14.** *MAPF is NP-hard for the flowtime objective, even when restricted to a tree that is a subdivided star.*

**Proof.** We reduce from LSR, which is NP-hard by Theorem 6. Given an LSR instance $I$ with a relocation bound $\ell$, we construct a subdivided star $T$ with center vertex $c$ as in Theorem 8(i): each stack is represented by a path joined to $c$.

We modify $T$ as follows for each stack path $\sigma$: Replace the edge connecting $\sigma$ to $c$ by a path containing $\ell + 1$ new vertices $b^1_\sigma, \ldots, b^{\ell+1}_\sigma$. Introduce $\ell + 1$ *seal agents* $s^1_\sigma, \ldots, s^{\ell+1}_\sigma$, with target vertices $b^1_\sigma, \ldots, b^{\ell+1}_\sigma$, respectively. Introduce a *gate agent* $g_\sigma$. Initially the seal agents are located in a new stack path $H_\sigma$ of length $\ell + 2$, above $g_\sigma$.

Finally, introduce $K$ (whose value is specified later) *padding agents* in a new start stack path, ordered from top to bottom as $q_1, \ldots, q_K$. The gate agents and padding agents have a common new target stack $Z$, where the gate agents must occupy the deepest positions, in an arbitrary fixed order, and the padding agents occupy the positions above them, ordered by $q_1, \ldots, q_K$ from bottom to top.

Let $N'$ be the total number of agents in $T$. Let $F_0 \coloneqq N'(N' + 1)/2 + \Delta$ be the flowtime lower bound, corresponding to a solution with no relocations, as defined in the proof of Theorem 13. We claim that $I$ admits a solution with at most $\ell$ relocations if and only if the MAPF instance on $T$ admits a solution with flowtime at most $B \coloneqq F_0 + \ell N'$.

Suppose that $I$ admits a solution with at most $\ell$ relocations. We simulate each LSR move by a corresponding transit through $c$ as in the 2-colored case. That is, $c$ is occupied at every timestep except the last and after visiting $c$ for the last time, each agent descends into its target without waiting. After all original LSR moves have been simulated, move the seal agents ("sealing" the original stacks) and then the gate agents directly to their respective targets. Finally, move the $K$ padding agents directly to their targets. The LSR relocations correspond to at most $\ell$ *extra transits* through $c$, i.e., an agent passing through $c$ but not for the last time. Each such transit delays at most all $N'$ final transits by one time step. Hence, the MAPF solution has a flowtime $F \leq F_0 + \ell N' = B$, as required.

Conversely, suppose there is a MAPF solution with a flowtime $F \leq B$. Let $\tau$ be the last timestep at which a gate agent transits through $c$. Each padding agent's final transit through $c$ occurs after $\tau$, since the gate agents need to occupy the deepest vertices in $Z$. Hence, each extra transit through $c$ occurring before $\tau$ precedes the $K$ padding agents' final transits. Let $r$ be the number of extra transits before $\tau$. We have $F \geq F_0 + rK$ since at least $K$ agents' final transits are delayed by $r$ timesteps compared to the lower bound. Let $m$ denote the number of non-padding agents, which is fixed for a given $I$. Choose $K$ such that $K > \ell m$. If $r \geq \ell + 1$, then $rK \geq (\ell + 1)K > \ell K + \ell m = \ell N'$, so $F > F_0 + \ell N' = B$, which is a contradiction. Thus $r \leq \ell$.

Fix a stack path $\sigma$. Each seal agent $s^t_\sigma$ lies above $g_\sigma$ in $H_\sigma$, so it leaves $H_\sigma$ before $g_\sigma$ does, hence before $\tau$. If every seal agent of $\sigma$ made an extra transit upon leaving $H_\sigma$, these transits would give $\ell + 1$ extra transits before $\tau$, contradicting $r \leq \ell$. Hence, some seal agent of $\sigma$ moves from $H_\sigma$ to $b^t_\sigma$ using a single transit through $c$. From then on $b^t_\sigma$ is occupied, so no original agent, i.e., corresponding to an LSR object, can enter or leave $\sigma$. Since this occurs before $\tau$, no original agent enters or leaves $\sigma$ after $\tau$. Since this holds for every $\sigma$, no original agent transits through $c$ after $\tau$, so every extra transit of an original agent is among the $r \leq \ell$ transits before $\tau$. We can therefore obtain a corresponding LSR solution, which is extracted by ignoring seal, gate and padding agents, with at most $\ell$ relocations, as required. ◀

## 6 Conclusion

We have shown that stack-based motion is a simple source of hardness across various fundamental motion coordination problems, settling the decades-old question of the complexity of pebble motion on trees. A key feature of the framework is that the same underlying reduction structure holds for distinct objectives, i.e., distance, makespan, and flowtime, which have typically required dedicated reductions or more involved gadget adaptations in previous work. There is therefore potential for a reusable basis for further hardness results in highly restricted tree-like settings. At the same time, any positive results must now account for these basic hardness results, which can serve as a baseline obstruction that could inform, e.g., approximation results.

The results are tight along natural axes: For the colored variants, hardness already appears with two colors, whereas the unlabeled motion is efficiently solvable [31]. Another boundary is established with respect to the maximum degree of the tree: our PMT reductions give hardness for maximum degree 3, while maximum degree 2 restricts the tree to a path, where the motion can be optimized efficiently.

On the labeled side, the LSR construction uses stacks of depth 3, which is also the depth of the subdivided stars for distance and makespan. We believe that the LSR construction can be tightened to stacks of depth 2; this refinement is left to a subsequent version.

Our analysis also identifies a tractable boundary case, which may be developed further to improve PMT algorithms. By Lemma 9, if a PMT solution attains the shortest-path lower bound, then its moves can be reordered so that each pebble performs its entire shortest-path traversal using consecutive moves. Consequently, for labeled PMT, one can test whether the lower bound is attainable by searching for an ordering of such atomic traversals. This can be done in polynomial time since monotone motion along fixed paths reduces to cycle detection on a dependency graph [7, 1]. This is also noteworthy because the analogous lower-bound attainability problem is NP-hard on 2D grids [17].

A natural next step is to identify the exact boundary between NP-hardness and polynomial-time solvability on trees. In particular, can the hardness results be further tightened by reducing parameters such as depth and the maximum degree? Are there efficient algorithms for special cases?

## References


1 Manuel Abellanas, Sergey Bereg, Ferran Hurtado, Alfredo García Olaverri, David Rappaport, and Javier Tejel. Moving coins. *Comput. Geom.*, 34(1):35–48, 2006. doi:10.1016/J.COMGEO.2005.06.005.

2 Oswin Aichholzer, Erik D. Demaine, Matias Korman, Anna Lubiw, Jayson Lynch, Zuzana Masárová, Mikhail Rudoy, Virginia Vassilevska Williams, and Nicole Wein. Hardness of token swapping on trees. In *ESA*, LIPIcs, pages 3:1–3:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.3.

3 Stefano Ardizzoni, Irene Saccani, Luca Consolini, Marco Locatelli, and Bernhard Nebel. An algorithm with improved complexity for pebble motion/multi-agent path finding on trees. *J. Artif. Intell. Res.*, 79:483–514, 2024. doi:10.1613/JAIR.1.15249.

4 Vincenzo Auletta, Angelo Monti, Mimmo Parente, and Pino Persiano. A linear-time algorithm for the feasibility of pebble motion on trees. *Algorithmica*, 23(3):223–245, 1999. doi:10.1007/PL00009259.

5 Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. Intractability of time-optimal multirobot path planning on 2D grid graphs with holes. *IEEE Robotics Autom. Lett.*, 2(4):1941–1947, 2017. doi:10.1109/LRA.2017.2715406.

6 Roman Barták, Marika Ivanová, and Jirí Svancara. From classical to colored multi-agent path finding. In *SOCS*, pages 150–152. AAAI Press, 2021. doi:10.1609/SOCS.V12I1.18566.

7 Stephen J. Buckley. Fast motion planning for multiple moving robots. In *ICRA*, pages 322–326. IEEE Computer Society, 1989. doi:10.1109/ROBOT.1989.100008.

8 Argyrios Deligkas, Eduard Eiben, Robert Ganian, Iyad Kanj, and M. S. Ramanujan. Parameterized algorithms for coordinated motion planning: Minimizing energy. In *ICALP*, LIPIcs, pages 53:1–53:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ICALP.2024.53.

9 Argyrios Deligkas, Eduard Eiben, Robert Ganian, Iyad Kanj, and M. S. Ramanujan. Parameterized algorithms for multiagent pathfinding on trees. In *AAMAS*, pages 584–592. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2025. doi:10.5555/3709347.3743574.

10 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM J. Comput.*, 48(6):1727–1762, 2019. doi:10.1137/18M1194341.

11 Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. Adding heuristics to conflict-based search for multi-agent path finding. In *ICAPS*, pages 83–87. AAAI Press, 2018. URL: https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17735.

12 Foivos Fioravantes, Dusan Knop, Jan Matyás Kristan, Nikolaos Melissinos, and Michal Opler. Exact algorithms and lowerbounds for multiagent path finding: Power of treelike topology. In *AAAI*, pages 17380–17388. AAAI Press, 2024. doi:10.1609/AAAI.V38I16.29686.

13 Foivos Fioravantes, Dusan Knop, Jan Matyás Kristan, Nikolaos Melissinos, Michal Opler, and Tung Anh Vu. Solving multiagent path finding on highly centralized networks. In *AAAI*, pages 23186–23193. AAAI Press, 2025. doi:10.1609/AAAI.V39I22.34484.

14 Graeme Gange, Daniel Harabor, and Peter J. Stuckey. Lazy CBS: implicit conflict-based search using lazy clause generation. In *ICAPS*, pages 155–162. AAAI Press, 2019. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3471.

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

16 Tzvika Geft. Fine-grained complexity analysis of multi-agent path finding on 2D grids. In *SOCS*, pages 20–28. AAAI Press, 2023. URL: https://ojs.aaai.org/index.php/SOCS/article/view/27279/27052.

17 Tzvika Geft and Dan Halperin. Refined hardness of distance-optimal multi-agent path finding. In *AAMAS*, pages 481–488. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2022. URL: https://ifaamas.org/Proceedings/aamas2022/pdfs/p481.pdf.

18 Oded Goldreich. Finding the shortest move-sequence in the graph-generalized 15-puzzle is NP-hard, 1984. Laboratory for Computer Science, Massachusetts Institute of Technology, unpublished manuscript.

19 Gilad Goraly and Refael Hassin. Multi-color pebble motion on graphs. *Algorithmica*, 58(3):610–636, 2010. doi:10.1007/S00453-009-9290-7.

20 Shuai D. Han, Nicholas M. Stiffler, Kostas E. Bekris, and Jingjin Yu. Efficient, high-quality stack rearrangement. *IEEE Robotics Autom. Lett.*, 3(3):1608–1615, 2018. doi:10.1109/LRA.2018.2800116.

21 Daniel Kornhauser, Gary L. Miller, and Paul G. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *FOCS*, pages 241–250. IEEE Computer Society, 1984. doi:10.1109/SFCS.1984.715921.

22 Daniel Koyfman, Dor Atzmon, Shahaf Shperberg, and Ariel Felner. Tree-MAPF: On the complexity of optimizing multi-agent path finding on tree graphs. In *SOCS*, 2026. To appear.

**23** Edward Lam, Pierre Le Bodic, Daniel Damir Harabor, and Peter J. Stuckey. Branch-and-cut-and-price for multi-agent pathfinding. In *IJCAI*, pages 1289–1296. ijcai.org, 2019. doi:10.24963/IJCAI.2019/179.

**24** Tomoki Nakamigawa and Tadashi Sakuma. On the order of the shortest solution sequences for the pebble motion problems. *CoRR*, abs/2503.20550, 2025. doi:10.48550/arXiv.2503.20550.

**25** Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.

**26** Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, 219:40–66, 2015. doi:10.1016/J.ARTINT.2014.11.006.

**27** Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. *Int. J. Robotics Res.*, 35(14):1750–1759, 2016. doi:10.1177/0278364916672311.

**28** Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *SOCS*, pages 151–159. AAAI Press, 2019. doi:10.1609/SOCS.V10I1.18510.

**29** Pavel Surynek. An optimization variant of multi-robot path planning is intractable. In *AAAI*. AAAI Press, 2010. doi:10.1609/AAAI.V24I1.7767.

**30** Jingjin Yu. Intractability of optimal multirobot path planning on planar graphs. *IEEE Robotics Autom. Lett.*, 1(1):33–40, 2016. doi:10.1109/LRA.2015.2503143.

**31** Jingjin Yu and Steven M. LaValle. Multi-agent path planning and network flow. In *WAFR*, volume 86 of *Springer Tracts in Advanced Robotics*, pages 157–173. Springer, 2012. doi:10.1007/978-3-642-36279-8_10.

**32** Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI*. AAAI Press, 2013. doi:10.1609/AAAI.V27I1.8541.