Parameterized Algorithms for Multiagent Pathfinding on Trees

Argyrios Deligkas
Royal Holloway, University of London
Egham, United Kingdom
argyrios.deligkas@rhul.ac.uk

Eduard Eiben
Royal Holloway, University of London
Egham, United Kingdom
eduard.eiben@rhul.ac.uk

Robert Ganian
TU Wien
Vienna, Austria
rganian@gmail.com

Iyad Kanj
DePaul University
Chicago, United States
ikanj@cdm.depaul.edu

M. S. Ramanujan
University of Warwick
Coventry, United Kingdom
R.Maadapuzhi-Sridharan@warwick.ac.uk

ABSTRACT

The classical Multiagent Pathfinding problem has been extensively studied not only within the artificial intelligence research community, but also by scholars in the areas of theoretical computer science and computational geometry. The problem asks for a minimum-makespan schedule that routes k agents (or robots) from their starting points to their destinations in a graph, while avoiding collisions, and is known to be NP-hard even on the fundamental class of trees. In this article we present two fixed parameter algorithms parameterized by k: the first yields a collision-free schedule on trees whose makespan deviates from the optimum by at most an additive polynomial function of k, and the second solves Multiagent Pathfinding optimally on the class of irreducible trees, i.e., trees with no vertices of degree 2. Both results rely on novel tools and insights into the properties of optimal schedules.

KEYWORDS

Multiagent Pathfinding; Trees; Parameterized Complexity

ACM Reference Format:
Argyrios Deligkas, Eduard Eiben, Robert Ganian, Iyad Kanj, and M. S. Ramanujan. 2025. Parameterized Algorithms for Multiagent Pathfinding on Trees. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 9 pages.

INTRODUCTION

Finding an efficient and collision-free routing of a set of agents (or, equivalently, robots) through a known environment is one of the most important and central tasks in multiagent planning, and arises naturally in a variety of real-world applications including, e.g., robotics [26], aircraft towing [19], warehouse management [18, 27] and digital entertainment [23]. The task is commonly formalized and studied as the Multiagent Pathfinding problem (MAPF)[1], where the environment is modeled as an n-vertex graph and each agent has a designated starting vertex and a designated destination vertex. At every time step, each agent can either wait or move from its current position in the graph to a neighboring vertex, but no vertex or edge can be used by more than a single agent at the same time step. In the most common formalization of MAPF, we seek a schedule which routes all the agents to their destinations while minimizing the number of required time steps, i.e., the makespan [9, 12, 28].

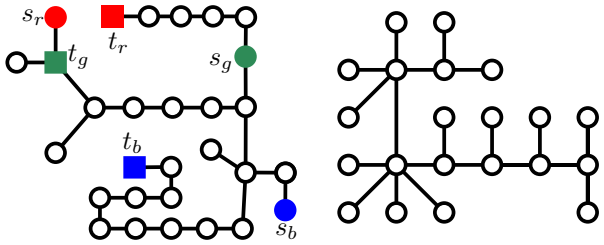

Figure 1: Left: An example instance of Multiagent Pathfinding on a tree T with three agents (marked in red, blue and green). Each agent's starting and destination vertex is marked with a full circle and full square, respectively. This particular instance admits a schedule with makespan 12. Right: An example of an irreducible tree.

The problem defined above has been extensively studied, and is considered highly intractable in the classical sense. Indeed, not only is the problem NP-hard on general graphs, but also on solid grids [6, 21] and trees [12]. In terms of applications, the need to solve MAPF in such settings arises not only when tasked with navigating robots through tight spaces such as caves and mines, but also, e.g., in entertainment (e.g., game challenges). Given the aforementioned general intractability of the problem, a number of heuristic approaches have been proposed, including SAT-based algorithms [25], scheduling techniques [2] and A*-based algorithms [13]—see also the dedicated survey [24]. An example instance of the problem on trees is provided in Figure 1 (left).

An alternative approach that can yield algorithms with guarantees on both running times and quality of identified solutions is the *fixed-parameter tractability* paradigm [4, 8]. There, one asks whether the problem can be solved in time $f(k) \cdot n^{O(1)}$ (FPT-time) where f is a computable function of some specified *parameter* k; algorithms with running time of this form are called *fixed-parameter algorithms*.

[1] Also known as the Coordinated Motion Planning problem [11].

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), A. El Fallah Seghrouchni, Y. Vorobeychik, S. Das, A. Nowe (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

The most obvious choice of the parameter for MAPF is the number of agents. Unfortunately, the vast majority of the techniques developed for designing fixed-parameter algorithms—such as bounded search trees, iterative compression and color coding—do not translate to this setting due to its temporal nature, and this is witnessed by the fixed-parameter intractability of the problem on general graphs (as established by Fioravantes et al. (2024); see also the closely-related work on temporal paths [16]). Hence, research in this direction has focused on establishing the fixed-parameter tractability of MAPF in natural and arguably simple settings, where the underlying graph is a tree or a solid grid [9, 11, 12, 14, 15].

Eiben, Ganian and Kanj (2023) employed a combination of problem-specific insights and integer programming techniques to design a fixed-parameter algorithm for MAPF on solid grids, but their approach strongly relies on the fact that large instances on solid grids provide ample "free space" for a small number of agents to navigate through. While subsequent works have obtained fixed-parameter algorithms for other objective functions [5] or parameterizations [12] of MAPF on tree-like networks, the existence of a fixed-parameter algorithm for the classical variant of the problem on trees was left open by all of the aforementioned works and has been explicitly posed as an open question in the latest of these [5, Section 7].

In this article, we obtain a fixed-parameter algorithm which, given an n-vertex tree instance of MAPF with k agents, computes in time $2^{O(k^2)} \cdot n^{O(1)}$ a schedule whose makespan is only an *additive* polynomial function of k away from the optimum. In a typical usage scenario where k is much smaller than both the makespan and the tree, this algorithm can be seen as "almost exact". At the same time, the fixed-parameter running time seems necessary to obtain such a "near-optimal" schedule: the recent NP-hardness result of Fioravantes et al. (2024) on trees is based on an approximation-preserving reduction from a token swapping problem on stars for which the existence of even multiplicative 2-approximation algorithms is a prominent open question [1]. As our second result, we obtain an exact algorithm for MAPF restricted to *irreducible trees*, i.e., trees with no vertices of degree 2 (see Figure 1 (right)).

Technical Contributions and Proof Overview

Both of the obtained algorithms are highly non-trivial and are, in fact, very different from each other: the former is easy to describe and implement but requires a highly non-trivial analysis to establish correctness, while the difficulty of the latter lies in the heavy algorithmic machinery that is used to find an exact schedule. Below, we provide an overview of each of these results, focusing on the main ideas, while leaving the technical details for the respective sections.

The First Result. A core ingredient in our fixed-parameter additive-approximation algorithm for MAPF on trees is a novel adaptation of conflict-based search—a technique that has been successfully used in heuristics for MAPF [3, 22]—to the parameterized setting. Informally, in conflict-based search one detects conflicts (i.e., collisions) between agents in advance and uses branching as well as search techniques to find an alternative schedule which avoids that collision. Our algorithm uses exhaustive branching techniques in order to obtain tight bounds on the optimality of the solution, and—crucially—only invokes conflict-based search in highly controlled situations in order to avoid exceeding our runtime bounds and guarantees.

On a high level, the algorithm can be described as follows. First, it processes the input tree T to compute the set of *havens*—connected subtrees centered at certain degree-3 vertices which allow for agents to move from any starting configuration in the haven to any target configuration in the same haven in at most $O(k^3)$-many time steps. While the notion of havens is not new [5], this is the first time they are used in the classical makespan-minimization setting; intuitively, every vertex of degree at least 3 is either the center of some haven, or it behaves essentially as a "leaf" in T.

Second, the algorithm computes an initial *pseudoschedule* for the agents, which may at this point contain conflicts. For agents whose starting position s and destination t are both near some haven[2], the pseudoschedule simply uses the unique s-t path in T. For each agent whose starting position (and/or destination) does not lie near to a haven—due to being in the middle of some long path of degree-2 vertices—we apply exhaustive branching to determine which of the at most 2 nearest havens it visits first (last) in a hypothetical optimal schedule. While there are some technical difficulties related to the aforementioned "leaf-like" vertices that require a more careful treatment of this branching step, the end result is a pseudoschedule that has correctly identified the first and last haven for each agent.

Third, the algorithm iteratively refines the initial pseudoschedule by following it until we detect a collision. If the collision occurs in the vicinity of a haven, we use the properties of the haven to resolve that collision while only incurring an additive loss of $f(k)$ time steps compared to a hypothetical optimal solution. If the collision does not occur near any haven, such a rerouting may be impossible—so instead, we exploit the structural properties of the instance to argue that in this case the collision occurred in the middle of a long path of degree-2 vertices and that a hypothetical solution must have one of the agents cross the path first. We then apply conflict-based branching to determine which of the agents will wait at one of the nearest havens, and alter the affected pseudoschedules accordingly. Crucially, in the proof of correctness, we show that for each pair of agents, the number of times we branch and the number of times we incur the aforementioned delay compared to a hypothetical optimal solution are both bounded by a constant.

The Second Result. Unfortunately, the approach outlined above does not translate to the computation of a schedule with *minimum* makespan. Indeed, the central idea used above is that collisions near havens are sufficiently "cheap" to resolve, but a truly makespan-optimal solution may in fact need to avoid such collisions by temporarily keeping some of the agents very far from the haven. Hence, the techniques we use to obtain our exact fixed-parameter algorithm differ from those used in the approximation setting.

For the following description, let f_1, f_2 and f_3 be some non-trivial, sufficiently large computable functions which will be specified later. In a first preparatory stage, the algorithm uses known polynomial bounds on the makespan of a hypothetical schedule [14] to iterate over the target makespan λ of the sought-after solution. It also applies a pruning argument to safely reduce the maximum degree of the irreducible input tree. The algorithm then carefully partitions

[2] i.e., having distance bounded by some function of the parameter

the agent set into *small-slack* and *large-slack* agents: the former contains agents which must follow the shortest (i.e., unique) path to their destination in all but at most $f_1(k)$ "deviating" time steps in order to achieve makespan λ, and for the latter we guarantee that they can deviate from their shortest path in at least $f_2(k)$ more time steps than any small-slack agent.

With these preparations in place, we prove that, assuming our guess of λ is correct, there must exist an optimal schedule with the property that at each time step $x \leq \lambda$, each agent is confined to an $f_3(k)$-sized ball centered around the x-th (or final) vertex on the unique path from its source to its destination. The algorithm uses this, along with the previously-established degree bound, to construct an FPT-sized "state-graph-like" representation of all possible positions of the agents at each time step, and finds the sought-after schedule by searching for a shortest path in this representation. As in the previous case, establishing this result requires not only formalizing the above ideas but also dealing with a range of additional technical difficulties—for instance, establishing the existence of an optimal schedule with the aforementioned property is far from trivial.

PRELIMINARIES

We assume basic knowledge of graph theory [7], approximation and the notion of *fixed-parameter tractability* [4, 8]. Unless specified otherwise, every graph considered in this work is simple and undirected. For two integers $i < j$, we use $[i, j]$ and $[i]$ as shorthand for the sets $\{i, i+1, \ldots, j\}$ and $[1, i]$, respectively.

A *tree* $\mathcal{T}$ is a connected acyclic graph, and a tree is *irreducible* if all of its non-leaf vertices have degree at least 3. For a subgraph H of $\mathcal{T}$ (where $H = \mathcal{T}$ is possible), we denote by $V(H)$ and $E(H)$ the vertex-set and edge-set of H, respectively. For two vertices $u, v \in V(\mathcal{T})$, denote by $\text{dist}(u, v)$ the length of the unique u-v path in $\mathcal{T}$. Let $u \in V(\mathcal{T})$ and H be a subgraph of $\mathcal{T}$ (H could be $\mathcal{T}$). Let $\deg_H(u)$ denote the degree of u in H. For $\rho \in \mathbb{N}$, we denote by $B(u, \rho)$, the *ball* centered at u and of radius ρ, that is, $B(u, \rho) = \{v \in V(\mathcal{T}) \mid dist(u, v) \leq \rho\}$. A path P in $\mathcal{T}$ is a *2-path* if P is an induced path of degree-2 vertices in $\mathcal{T}$ (see, e.g., the s_g-t_r path in Figure 1 (left)); observe that 2-paths do not occur in irreducible trees.

Problem Definition. We follow the standard problem definition for Multiagent Pathfinding restricted to the class of trees. Apart from the input tree $\mathcal{T}$, each instance is provided with a set $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ of k agents. Each $R_i \in \mathcal{R}$ is represented by a starting vertex s_i and a destination vertex t_i in $V(\mathcal{T})$, and we assume all the s_i's to be pairwise distinct and all the t_i's to be pairwise distinct (as otherwise the instance may be immediately rejected). At each time step, each agent may either move to an adjacent vertex or stay at its current vertex.

We use a discrete time frame $[0, t]$, $t \in \mathbb{N}$, to reference the sequence of moves of the agents and in each time step $x \in [0, t]$ every agent remains stationary or moves. The *shortest path* for an agent R_i is the unique s_i-t_i path in $\mathcal{T}$. A *route* for agent R_i is a sequence $W_i = (u_0, \ldots, u_t)$ of vertices in $\mathcal{T}$ such that u_0 is the starting vertex s_i of R_i, $u_t = t_i$, and for each $i \in [t]$ it holds that $u_{i-1} = u_i$ or $u_{i-1}u_i$ is an edge in $\mathcal{T}$. The vertex u_i is the position of R_i at time step i. For $1 \leq i < j \leq k$, two routes $W_i = (u_0, \ldots, u_t)$ and $W_j = (v_0, \ldots, v_t)$ are *conflicting* (or *in conflict*) if there exists some time step x such that $u_x = v_x$ or $u_{x-1}u_x = v_{x-1}v_x$. A *schedule* Γ for $\mathcal{R}$ is a set of pairwise non-conflicting routes $\{W_i \mid i \in [k]\}$, during a time interval $[0, \lambda]$. The integer λ is called the *makespan* of Γ. Using the introduced terminology, we formalize our problem of interest below.

Multiagent Pathfinding on Trees (Tree-MAPF)	
Input:	An n-vertex tree $\mathcal{T}$, an integer k and a set $\mathcal{R} = \{R_i = (s_i, t_i) \mid i \in [k]\}$ of agents.
Task:	Compute a schedule for $\mathcal{R}$ of minimum makespan (or decide that none exists).

Note that the feasibility component of Tree-MAPF (i.e., deciding whether a schedule exists at all) can be resolved in linear time [15] and that the minimum makespan is known to be upper-bounded by $O(n^3)$ [14]. However, a polynomial-time algorithm for Tree-MAPF is excluded by the NP-hardness of deciding whether there exists a schedule with makespan at most a given integer λ [12]. We remark that without loss of generality, we assume that at each time step there exists at least one agent that made a move.

FPT ADDITIVE APPROXIMATION

In this section, we obtain the first main result: a fixed-parameter algorithm that solves Tree-MAPF up to an additive error that depends purely on the parameter k (i.e., the number of agents).

Preparations and Setup

We start by recalling the notion of havens that was recently introduced and used to obtain an approximation algorithm for the variant of Multiagent Pathfinding, where the aim is to minimize the total distance traveled as opposed to the makespan. Intuitively, a haven is a subtree rooted at a high-degree vertex that allows agents to safely reconfigure themselves in a parameter-bounded number of moves.

Definition 1 (Deligkas et al. (2024)). *A vertex $w \in V(\mathcal{T})$ is* nice *if there exist three connected subtrees C_1, C_2, C_3 of $\mathcal{T}$ such that: (i) the pairwise intersection of the vertex sets of any pair of these subtrees is exactly w, and (ii) $|V(C_1)| \geq k+1$, $|V(C_2)| \geq k+1$, and $|V(C_3)| \geq 2$.*

If w is nice, let $x \in V(C_3)$ be a neighbor of w in $V(C_3)$, and define the haven *H_w of w to be the subtree of $\mathcal{T}$ induced by the vertices in $\{x\} \cup V(C_1) \cup V(C_2)$ whose distance from w is at most k.*

For a set $S \subseteq \mathcal{R}$ of agents and a subtree H, a *configuration* of S w.r.t. H is an injection $\iota : S \longrightarrow V(H)$. The following result captures the crucial property of havens.

Lemma 1 (Deligkas et al. (2024)). *Let w be a nice vertex, let C_1, C_2, C_3 be three subgraphs satisfying conditions (i) and (ii) of Definition 1, and let H_w be a haven for w. Let $S \subseteq \mathcal{R}$ be a set of agents forming a configuration $\iota(S)$ in H_w. Every configuration $\iota'(S)$ in H_w can be obtained from $\iota(S)$ via a sequence of $O(k^3)$ moves that take place in H_w.*

While the previously-introduced notion of havens and their central property play a role in our algorithm for the makespan-minimization task, the actual approach used to obtain it (and the difficulties faced) is very different from the one used in the article

which first introduced this notion [5]. In particular, when minimizing the total distance traveled, one can assume, without loss of generality, that the agents move *sequentially* and their conflicts can thus be resolved in a pairwise and localized manner, and always near a suitable haven. In our makespan-minimization setting, we do not have this luxury: several conflicts may need to be resolved simultaneously or in an overlapping manner, and—crucially—conflicts may occur very far from nice vertices and their havens.

To deal with the latter situation, we will need to obtain a better understanding of vertices which are not "nice" as per Definition 1. This is achieved in the following lemma.

Lemma 2. *Let v be a vertex of degree at least 3 in $\mathcal{T}$ which is not nice. Then there exists a neighbor u of v such that the connected component of $\mathcal{T} - uv$ containing v has size at most $3k$.*

Proof. Suppose that no neighbor u of v satisfies the property stated in the lemma, which asked for the connected component of $\mathcal{T} - uv$ containing v to have size at most $3k$. Equivalently, this means that for every neighbor u_i of v, the connected component D'_i of $\mathcal{T} - u_i v$ containing v has size at least $3k+1$. Let D_i be the connected component of $\mathcal{T} - u_i v$ not containing v, and let us assume that the neighbors $u_1, \ldots, u_\zeta$ of v are listed in ascending order based on the size of these components, i.e., $|D_1| \leq |D_2| \leq \cdots \leq |D_\zeta|$.

With the setup in place, we now construct a witness showing that v is nice. First, we set $C_3 := \{v, u_1\}$. Let j be the smallest index such that $|\bigcup_{2 \leq i \leq j} D_i| \geq k$, and set $C_2 := \{v\} \cup \bigcup_{2 \leq i \leq j} D_j$. Observe that this guarantees $|C_2| \geq k+1$. Next, we argue that $j < \zeta$. Indeed, if this were not the case, we would have $|\bigcup_{2 \leq i \leq j-1} D_i| \leq k$ (by the definition of j) and hence also $|D_1| \leq k$, and thus we would arrive at $|\bigcup_{1 \leq i \leq \zeta - 1} D_i| \leq 2k$—a direct contradiction with $D'_\zeta = \{v\} \cup \bigcup_{1 \leq i \leq \zeta - 1} D_i$ having size at least $3k$.

Since $j < \zeta$, we can proceed to defining $C_1 := \{v\} \cup \bigcup_{j+1 \leq i \leq \zeta} D_i$. To show that v is nice and complete the proof, it suffices to argue that $|C_1| \geq k+1$. First, observe that if $|D_j| \geq k$ then also $|D_\zeta| \geq k$, and in particular $|C_1| \geq k+1$ as desired. For the case where $|D_j| \leq k$, we recall that $|D'_j| \geq 3k+1$ and that D'_j is the disjoint union of v and all of the individual connected components D_i, $i \neq j$. The latter implies that $|D'_j| = |D_1| + |\big(\bigcup_{2 \leq i \leq j-1} D_i\big)| + |C_1|$. Since the first term is upper-bounded by $|D_j| \leq k$ and the second term is upper-bounded by k as well due to the definition of j, we obtain that $|C_1| \geq k+1$, as desired. □

Intuitively, Lemma 2 implies that—assuming k is sufficiently small compared to n—every higher-degree vertex which is not nice essentially acts as a "complicated leaf", as almost all vertices in the tree lie in the direction of precisely one of its incident edges. Using this fact, we can directly solve all instances which do not contain any nice vertex whatsoever. We provide this result separately, as our main algorithm will assume the existence of at least one nice vertex in the tree.

Lemma 3. Tree-MAPF *restricted to instances which do not contain any nice vertex can be solved optimally in time $k^{O(k)} \cdot n$, and in particular is fixed-parameter tractable.*

Proof Sketch. We use Lemma 2 to show that the tree $\mathcal{T}$ consists of a long 2-path P plus some small branching vertices near the endpoints p_1, p_2 of that path. Our algorithm exhaustively branches to decide, for each agent which starts (or has a destination) far from the endpoints of P, which out of p_1 and p_2 it will visit first (or last). In each branch, we can safely send the agents along the path until they reach that endpoint and essentially vacate the path P of all agents. At that point (or, more specifically, as soon as agents are only traversing P in a single direction), we construct a *pruned state graph* H which contains the following information about all the agents in $\mathcal{T}$ at a particular time step.

1. The exact positions of all agents which are at distance at most $3k$ from p_1 or p_2.
2. Whether P is being traversed by agents exclusively from p_1 to p_2, or from p_2 to p_1.
3. The exact position of the agent R_i that is traversing P and will reach the endpoint of P first (i.e., is the first among the group of agents traversing P).
4. For each other agent R_j traversing P, the exact distance between R_j and R_i.

Crucially, one can show that there exists a solution with the property that the maximum distance between R_j and R_i is upper-bounded by k^4, which provides an overall upper bound of $2^{O(k \log k)}$. n on the size of the state graph. The result then follows by finding a shortest path from the initial to the final configuration in H, obtained after the exhaustive branching described in the first paragraph. □

The First Algorithm

With the setup in place, we can proceed to the advertised algorithmic result.

Theorem 1. *There is an $2^{O(k^2)} \cdot n^{O(1)}$-time approximation algorithm for* Tree-MAPF *with an additive error of $O(k^5)$.*

Proof Sketch. Let $\mathcal{I} = (\mathcal{T}, \mathcal{R}, k)$ be an instance of Tree-MAPF and let $\mathcal{OPT}$ be an optimal schedule for $\mathcal{I}$. We begin by checking that $\mathcal{T}$ contains at least one nice vertex; if not, we solve the instance by invoking Lemma 3. For each $R_i \in \mathcal{R}$, we exhaustively branch over (i.e., "guess") the first and last degree-3 vertices that it visits and the first and last nice vertices that it visits in $\mathcal{OPT}$; denote these vertices by $\texttt{init}(R_i)$, $\texttt{final}(R_i)$, $\texttt{fnice}(R_i)$, and $\texttt{lnice}(R_i)$, respectively. We remark that, since $\mathcal{T}$ is a tree, for each of these four vertices there are only at most 2 possible choices, resulting in a branching factor of at most $2^{O(k)}$ at this stage.

We will describe the FPT approximation algorithm over FPT-many rounds. We define an initial route, for each R_i, denoted π_i, to be the walk: s_i-$\texttt{init}(R_i)$-$\texttt{fnice}(R_i)$-$\texttt{lnice}(R_i)$-$\texttt{final}(R_i)$-t_i. Observe that by Lemma 2, every degree-3 vertex between $\texttt{fnice}(R_i)$ and $\texttt{lnice}(R_i)$ on π_i must be a nice vertex. Define an *event* to be a time step at which two agents collide. We define a schedule Γ_{APX} for the agents iteratively as follows. Starting at time step 0, each R_i (continuously) follows its initial route π_i. If no collision occurs, then Γ_{APX} is already a valid schedule. Suppose now that two agents R_i and R_j collide (i.e., that there is an event), and let t be the first time step at which an event occurs. Let v_i and v_j be the two vertices at which R_i and R_j are located at time step $t-1$, i.e., before they collided. We distinguish the following cases and modify π_i accordingly.

Case (i). v_i and v_j are both on a 2-path P between two nice vertices w_i, w_j, such that v_i (and v_j) is at distance at least $k+2$ from w_i and w_j, and where R_i and R_j are going in opposite directions on this path.

Observe that in $\mathcal{OPT}$, one of the two agents R_i, R_j must traverse the whole path P before the other agent enters P. Therefore, we branch to determine which of the two agents traverses P before the other agent enters P. Without loss of generality, assume that R_i traverses P in the direction w_i-w_j before R_j enters P through w_j. Note that, if our guess is correct, then necessarily w_j appears on π_j before the location of this collision. We modify π_j so that R_j waits on P at the closest vertex to w_j such that no other agent waits there already. This waiting is until R_i collides with R_j at this vertex, which is resolved later. We note that any agent that needs to traverse P from w_j to w_i and enters P after R_j is also made to wait on P close to w_j until R_i arrives.

Case (ii). If v_i (or v_j) is at a distance at most k from some nice vertex n_i, define $B = B(n_i, k+3)$. Repeatedly, do the following: if there is an agent at distance at most k from a vertex in B, increase the radius of B by k. Observe that, since this repetition can occur at most $k-2$ times (as two agents are already in B), at the end of this repetition, B has radius at most k^2 and satisfies that every agent outside B is at distance at least $k+1$ from every vertex in B. Now, we modify the schedule as follows.

We freeze every agent located at a vertex outside of B. Let S_{wait} be the subset of agents in B that are currently waiting due to an agent outside of B as in Case (i) above.

We define a configuration ι for the agents in B as follows. (A) For each "waiting" agent $R_x \in S_{wait}$, ι places R_x at an arbitrary vertex that is not the destination of any other agent. (B) For each agent R_x whose destination is in B, ι places R_x on its destination. (C) For every other agent R_x in B, we process these agents in a decreasing order of the distance between n_i and their destination t_x (breaking ties arbitrarily). Starting from the agent R_x for which this distance is maximized, we let $b_x \in V$ be the unique vertex on π_x that has distance precisely $k-q$ from the boundary of B, where q is the index of the agent R_x w.r.t. the aforementioned order. This ensures that agents that need to leave B are safely placed outside of it, in the "direction" they need to go, and in an order which respects the distance to their destination.

We then apply Lemma 1 to achieve configuration ι. We do so by first moving all agents in B to within distance k from the nice vertex, applying Lemma 1, and then rearranging the agents as desired afterwards. This can be done in $O(k^3)$ many steps.

Case (iii). Since $\mathcal{T}$ contains at least one nice vertex, Lemma 2 guarantees that if v_i or v_j were to lie on *any* path P' between two nice vertices, every vertex of degree at least 3 on P' would be nice as well. Hence, the first two cases completely cover all situations except for the following Case (iii): Both v_i and v_j are in a subtree that is rooted at some nice vertex n_i and no other vertex of this subtree is nice.

While this case intuitively covers all of the "borderline cases" that arise from collisions occurring in the peripheries of $\mathcal{T}$, a formal treatment requires a careful case analysis that distinguishes four subcases. On a high level, the proof is based on arguing that either the necessary reconfiguration can be carried out without routing both agents to n_i or by showing that the agents must reach n_i in a hypothetical schedule. We then route the agents accordingly while performing the necessary reconfigurations along the way.

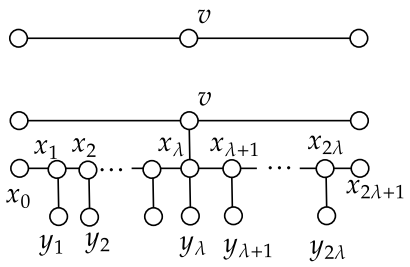

Figure 2: The caterpillar gadget used in Theorem 2. Top: the original vertex v of degree 2. Bottom: vertex v with the caterpillar gadget attached

The running time of the algorithm can be shown to be dominated by the branching along the 2-paths that happens in Case (i) to resolve conflicts between pairs of agents. Since the number of such pairs is $O(k^2)$, we can show that there will be at most one conflict per pair and the branching can be done in $2^{O(k^2)}$ time. In particular, a careful analysis of the running time over all cases shows that the algorithm can be implemented to run in $2^{O(k^2)}n$ time.

Finally, we indicate why the schedule Γ_{APX} obtained by the algorithm is within $O(k^5)$ additive delay from a hypothetical optimum. First, observe that no additional delay is incurred in Case (i). In Case (ii), each potential collision incurs a delay of $O(k^3)$ to resolve by Lemma 1 and the total number of Case-(ii) collisions among the agents can be upper-bounded by $O(k^2)$. Finally, the total additive delay incurred over all collisions handled under Case (iii) can be shown to be upper-bounded by $O(k^4)$. □

MAPF ON IRREDUCIBLE TREES

While the additive approximation error that occurs in Theorem 1 only depends on the parameter, it is still natural to ask whether the error can be avoided altogether. In this section, we provide an exact algorithm for Tree-MAPF when the input trees are irreducible. We first show that this problem restriction remains hard:

Theorem 2. Tree-MAPF *remains* NP*-complete when restricted to instances in which the tree is irreducible.*

Proof. We reduce from Tree-MAPF on general trees [12]. The idea is that we can append to each degree 2 vertex v a "caterpillar" whose middle vertex is adjacent to v and the main path of the caterpillar contains agents that need to go distance λ in the caterpillar and are always blocking the neighbor of v.

Caterpillar gadget. This gadget (refer to Figure 2), denoted $T_c(\lambda)$ consists of vertices $x_0, x_1, x_2, \ldots, x_{2\lambda}, x_{2\lambda+1}, y_1, y_2, \ldots, y_{2\lambda}$; the x-vertices form a path and, for each $i \in [2\lambda]$, y_i is connected with x_i. For every $i \in [\lambda]$, we create an agent with starting vertex x_i and destination $x_{\lambda+i}$. Observe that the makespan of $T_c(\lambda)$ is λ.

Therefore, given an instance T_g of Tree-MAPF on general trees that asks whether the makespan is λ, we construct an irreducible tree T_{ir} as follows. For every vertex v of degree 2, we create a

caterpillar gadget $T_c(\lambda)$ and we connect vertex x_λ with v. Observe that T_{ir} consists by T_g plus a union of caterpillar gadgets. Clearly, this construction can be performed in polynomial time.

We claim that T_{ir} has makespan λ, if and only if T_g has makespan λ. The correctness follows for the fact that no schedule that achieves makespan λ on T_{ir} can route any of the original agents inside any caterpillar gadget. Observe that if this happens, then there will be a "delay" for the agents of the corresponding gadgets that will increase the makespan by at least one. □

Preparations and Setup for the Exact Algorithm

The approach we use to obtain an exact fixed-parameter algorithm for Tree-MAPF on irreducible trees does not employ the notions of nice vertices and havens, as we cannot "afford" to use these to arbitrarily reconfigure agents in their vicinity—each such reconfiguration may lead to an irreversible loss of makespan compared to the true optimum.

Instead, our algorithm relies on showing the existence of a "canonical solution" for every yes-instance of the problem, which we can then compute in FPT-time. First, we assume that we are given a target makespan λ. This assumption follows from the known cubic upper-bound on the makespan of an optimal schedule [14], combined with standard enumeration of the makespan (possibly up to the upper bound), to reduce Tree-MAPF to a decision problem.

Second, we establish and employ the following reduction rule to bound the maximum degree of the input tree $\mathcal{T}$. We say that two instances are *equivalent* if every solution for one instance can be transformed, in polynomial time, into a solution for the other instance.

Proposition 1. *Let $\mathcal{I} = (\mathcal{T}, \mathcal{R}, k, \lambda)$ be an instance of* Tree-MAPF. *In P-time, we can reduce $\mathcal{I}$ to an equivalent instance in which the tree has maximum degree at most $3k$.*

Proposition 1 allows us to assume that the instance is provided with a makespan bound λ and that the input tree has degree at most $3k$. Next, we make use of this assumption to prove a sufficient condition on an optimal schedule that enables us to solve the problem in FPT-time. This result will be employed as a subroutine in our main algorithm, and much of the work therein will focus on ways to correctly restrict our attention to schedules satisfying this condition.

For a computable function ρ of k, let us call a Tree-MAPF instance $(\mathcal{T}, \mathcal{R}, k, \lambda)$ *$\rho(k)$-ball-restricted* if, for every time step $0 \leq x \leq \lambda$ and every agent R, the position of R at time step x is confined to $B(v^x, \rho(k))$ where $v^x \in V(\mathcal{T})$ can be computed from x in polynomial time.

Lemma 4. *Let $\rho(k)$ be a computable function. The restriction of* Tree-MAPF *to instances which are $\rho(k)$-*ball-restricted *can be solved in time $\lambda \cdot (3k)^{k \cdot \rho(k)} \cdot |\mathcal{T}|^{O(1)}$.*

Proof. By Proposition 1, we may assume that every vertex in $\mathcal{T}$ has degree at most $3k$. By assumption, for every agent R_i and every time step $0 \leq x \leq \lambda$, R_i is confined to a ball of size $f(k) = O((3k)^{\rho(k)})$, consisting of all vertices in $\mathcal{T}$ of distance at most $\rho(k)$ from a vertex in $\mathcal{T}$ (v^x) that can be determined in polynomial time. Therefore, at each time step, there are at most $f(k)^k$ many possible configurations for the agents, where a configuration is a tuple specifying the positions of all agents. Now, we can construct a directed configuration graph $G_{\mathcal{T}}$, whose vertices are pairs of the form (x, C), where $0 \leq x \leq \lambda$, is a time step and C is a possible configuration of the agents at time step x, and such that there is an edge from (x, C) to $(x{+}1, C')$, if C can yield C' in a single move of the agents. Deciding whether there is a schedule S for the agents from the starting positions to the destination positions with makespan λ, reduces to deciding whether a path from $(0, (s_1, s_2, \ldots, s_k))$ to $(\lambda, (t_1, t_2, \ldots, t_k))$ exists in $G_{\mathcal{T}}$. Note that the number of vertices of $G_{\mathcal{T}}$ is at most $\lambda \cdot f(k)^k$, and we can decide the existence of such a path in $O(|V(G_{\mathcal{T}})| + |E(G_{\mathcal{T}})|)$ time. □

Slack: A Useful Measure of Agent Urgency

On an intuitive level, our algorithm employs different treatments to agents that need to reach their destinations "urgently" and agents that can afford to "wait" for longer periods of time without violating the makespan constraint. To partition the agents into these groups, we will employ the notion of *slack* that was introduced for Multiagent Pathfinding in the setting of solid grids [9]. While the definition is precisely the same as the one proposed in the previous work, the way we use it is entirely different. The structure of solid grids guarantees plenty of "room" for large-slack agents to avoid small-slack agents and the movement of small-slack agents was handled by an encoding into an Integer Linear Program [9, Subsection 3.2]. On the other hand, on trees, large-slack agents have very little room to maneuver and their potential collisions with small-slack agents need to be handled using more careful, combinatorial insights.

Definition 2. *Let $\mathcal{I} = (\mathcal{T}, \mathcal{R}, k, \lambda)$ be an instance of* Tree-MAPF. *The* slack *of an agent $R_i \in \mathcal{R}$, denoted slack(R_i), is $\lambda - \mathrm{dist}(s_i, t_i)$.*

Proposition 2. *Let $\mathcal{I} = (\mathcal{T}, \mathcal{R}, k, \lambda)$ be an instance of* Tree-MAPF *and let $\sigma(k)$ be a function. If slack$(R_i) \leq \sigma(k)$, for every $i \in [k]$, then $\mathcal{I}$ can be solved in* FPT*-time.*

Proof. For any agent R_i and for any time step $0 \leq x \leq \lambda$, R_i is confined to the ball of radius $\sigma(k)$ centered at the unique vertex of distance $\min(x, \mathrm{dist}(s_i, t_i))$ from s_i on the unique s_i-t_i path of $\mathcal{T}$. Hence, Lemma 4 applies. □

Proposition 2 follows from Lemma 4 and allows us to restrict our attention to instances containing at least some large-slack agents.

We next formally identify an appropriate threshold for the slack that determines which agents are small-slack and which are large-slack. Note that given λ, the slack of every agent is known.

Lemma 5. *Given an instance $\mathcal{I} = (\mathcal{T}, \mathcal{R}, k, \lambda)$ of* Tree-MAPF *and a computable function $g : \mathbb{N}N \to$ natN, in time* FPT *in k, we can either decide the instance $\mathcal{I}$, or partition the agents in $\mathcal{R}$ into two sets S and L such that*

- *$L \neq \emptyset$,*
- *each agent in S has slack at most $g(k)^{k+1}$, and*
- *each agent in L has slack at least $g(k)$ times the slack of any agent in S.*

Proof. Let us sort the agents in a nondecreasing order of their slack. Let $i \in [k]$ be the smallest index (w.r.t. the sorted list) of an

agent with slack at least $g(k)^{i+1}$. If no such agent exists, then all agents have slack at most $g(k)^{k+1}$ and we can solve the instance in FPT time by Proposition 2. Otherwise, we let $S = \{R_1, R_2, \ldots, R_{i-1}\}$ (defined as the empty set if $i = 1$) and $L = \{R_i, R_{i+1}, \ldots, R_k\}$. Note that by the choice of i, all agents in S have slack at most that of R_{i-1}, which is at most $g(k)^{(i-1)+1} = g(k)^i$, and all agents in L have slack at least that of R_i, which is at least $g(k)^{i+1}$ and the lemma follows. □

In the above lemma, S is the set of small-slack agents and L is the set of large-slack agents. Our last two preparatory lemmas establish two important properties of small-slack agents. The first shows that every subtree with sufficiently many high-degree vertices contains many vertices that are not visited by any small-slack agent in some optimal solution. The main algorithm uses this result to guarantee the existence of "safe" waiting points for large-slack agents.

Lemma 6. *Let $\mathcal{I} = (\mathcal{T}, \mathcal{R}, k, \lambda)$ be a yes-instance of* Tree-MAPF *in which $\mathcal{T}$ is irreducible and let Γ be a solution to $\mathcal{I}$. Let $S \subseteq \mathcal{R}$ be a set of agents, each of slack at most $\sigma(k)$ in Γ, for some function σ, and let H be a subtree of $\mathcal{T}$ such that the number of vertices with degree at least 3 in H is at least q. Then at least $(q - k \cdot \sigma(k))$ vertices in H are not visited by any agent in S during Γ.*

The final preparatory lemma establishes that since small-slack agents must more-or-less follow the route to their destination, they cannot interfere with any part of a path in $\mathcal{T}$ for too long.

Lemma 7. *Let $S \subseteq \mathcal{R}$ be a set of agents each of slack at most $\sigma(k)$. Moreover, let P be a path in $\mathcal{T}$, T be a time interval and $|T|$ denote its length. Then there is a subinterval of T of length at least $|T|/(k \cdot (|P| + \sigma(k)) + 1)$ during which P is devoid of any agent in S.*

The FPT Algorithm on Irreducible Trees

We are now ready to establish our second main result:

Theorem 3. *There is a fixed-parameter algorithm which solves* Tree-MAPF *on irreducible trees to optimality.*

Proof. Let the given target makespan be λ. By Lemma 4, it suffices to show that every yes-instance of Tree-MAPF has a solution in which the position of each agent at every time step is confined to a ball whose center can be computed in polynomial time and whose radius is a computable function of k.

Let $\mathcal{I} = (\mathcal{T}, \mathcal{R}, k, \lambda)$ be a given yes-instance. By Proposition 1, we may assume that every vertex in $\mathcal{T}$ has degree at most $3k$. By Lemma 5, we may assume that $\mathcal{R}$ is partitioned into two sets, S and L, such that each agent in S has slack at most $\sigma(k)$ and each agent in L has slack at least $\rho(k) = g(k) \cdot \sigma(k)$, where $g(k)$ is "large-enough" computable function of our choice, to be determined later. Note also that the fact that L is nonempty implies that the overall makespan λ is large (at least $\rho(k)$). Without loss of generality, we will also assume that $|\mathcal{T}|$ is large (i.e., larger than any chosen function of k), as otherwise, the instance can be solved in FPT-time by brute force.

For $R_i \in \mathcal{R}$, denote by P_i the unique s_i-t_i path in $\mathcal{T}$. For any time step $0 \leq x \leq \lambda$, denote by $P_i(x)$ the unique vertex of P_i at distance $\min(x, |P_i|)$ from s_i. Clearly, for any small-slack agent $R_i \in S$ and any $0 \leq x \leq \lambda$, R_i is confined to $B(P_i(x), \sigma(k))$ at time step x. We will show next that there is a solution to $\mathcal{I}$ in which every large-slack agent $R_i \in L$, at time step $0 \leq x \leq \lambda$, is confined to $B(P_i(x), \rho(k))$. This will complete the proof.

Consider any solution Γ to $\mathcal{I}$. As mentioned above, every agent $R_i \in S$ at time step $0 \leq x \leq \lambda$ is confined to $B(P_i(x), \sigma(k)) \subseteq B(P_i(x), \rho(k))$, and its route in Γ will not be altered. We will now show how to modify Γ so that the agents in L satisfy that property as well. Choose a function $\beta(k) = \omega((k \cdot \sigma(k))^{k^2 \cdot \sigma(k)^2})$.

For any $R_i \in L$, $i \in [k]$, during the first (resp. last) $\beta(k)$ time steps in Γ, R_i covers positions that are within distance $\beta(k)$ from s_i (resp. t_i). Let $First_i$ and $Last_i$ be the two subtrees of $\mathcal{T}$ visited by R_i during the first and last $\beta(k)$ time steps in Γ, respectively. (The vertices/edge of $First_i$ and $Last_i$ may have been possibly visited multiple times.) Define E_i to be the *extension* of P_i, which consists of $P_i + First_i + Last_i$ plus, for each vertex v whose degree in $First_i + Last_i + P_i$ is 2, a unique neighboring vertex u of v that is not on $P_i + First_i + Last_i$ and the edge uv. The vertex u is called a *pendant vertex*. Note that E_i is well defined since $\mathcal{T}$ is an irreducible tree.

We will modify Γ so that the route of R_i, $i \in [k]$, is confined to E_i and satisfies that, at any time step $0 \leq x \leq \lambda$, the position of R_i in E_i is in the ball $B(P_i(x), \rho(k))$. For the rest of the proof, it helps to consider $\rho(k)$ to be much larger than $\beta(k)$. (In fact, $\rho(k) = \Omega(\beta(k))$ with a large multiplicative constant would suffice.)

Define a *safe* vertex to be a vertex in $\mathcal{T}$ that no agent in S visits during Γ. Observe that, for each $i \in [k]$, since P_i is a path and since each of $First_i$ and $Last_i$ has size at most $\beta(k)$, the number of vertices in E_i with no pendant vertices attached to them is at most $|First_i| + |Last_i| \leq 2\beta(k) + 2$ (the plus 2 is to account for the endpoints of P_i). Furthermore, for every agent in S, the number of pendant vertices that it visits is at most $\sigma(k)/2 + 2$ (each time such an agent visits a pendant vertex it incurs a slack of at least 2, except possibly for the first and last such vertices). Therefore, after noting that no two pendant vertices are adjacent to the same vertex in E_i, we conclude that any subtree of E_i of size s contains a large number of pendant vertices that are safe, to be precise, at least $(s - 2(\beta(k) + 1) - k(\sigma(k) + 4))/2$.

Consider an agent $R_i \in L$ and consider its extension E_i. We divide the routing of R_i in the modified schedule into three phases. The first phase starts at time step 0, and ends when R_i reaches a designated safe vertex. The second phase starts after that, routes R_i from a safe vertex to another (safe vertex), and ends when R_i reaches the last safe vertex at distance at most $k \cdot \sigma(k)$ from its destination t_i. The third and last phase starts after the second phase is complete and ends at time step λ, when R_i reaches its destination t_i. The fact that we have sufficiently many safe vertices will be used to show that all three phases can be completed.

Phase 1. We route R_i during the first phase, considering two cases based on the size of $First_i$.

Case 1: If $|First_i| > 2k(\sigma(k) + 1)$, then either E_i contains at least $k \cdot \sigma(k) + k$ vertices that are either leaves in $First_i$ or pendant vertices attached to $First_i$, or at least $k\sigma(k) + k$ vertices of degree at least 3 in $First_i$. In the former case, and since the number of pendant and leaf-vertices in E_i traversed by all the agents in S is at most $k \cdot \sigma(k)$, it follows that the number of safe vertices in $First_i$ or attached to $First_i$ is at least k, and hence, R_i, and all the agents in L that are present in $First_i$ during the first phase, can be assigned safe vertices in $First_i$ (or attached to it) to stay at during the first

phase. That is, we can either find a safe vertex that is not visited by any other agent in L or we can achieve such an assignment. In the latter case, the same conclusion follows from Lemma 6.

Case 2: Suppose now that $|First_i| \leq 2k(\sigma(k)+1) \leq 4k\sigma(k)$. If there exists a vertex $v_i \in First_i$ at which R_i waits for $\Omega(k \cdot \sigma(k))^3$ time steps during the first $\beta(k)$ steps, let T be a time interval during which R_i waits at v_i this many steps, and consider the path v_i-t_i in $\mathcal{T}$; let P be the subpath of the v_i-t_i path that starts at v_i and has length $|First_i| + 2k\sigma(k)$. By our assumption on $|First_i|$, $|P| = O(k\sigma(k))$. Moreover, more than k vertices on P have pendant vertices that are safe (since the agents in S can visit at most $k\sigma(k)$ pendant vertices). Applying Lemma 7, we conclude that there is a subinterval $T' \subseteq T$ of length more than $|P|$ during which P is devoid of agents in S. Therefore, R_i can be routed to a safe pendant vertex on P during T' (while avoiding the agents in L if any).

Suppose now that R_i never waits more than $O((k \cdot \sigma(k))^3)$ time steps at any vertex in $First_i$ during the first $\beta(k)$ many time steps. By the choice of $\beta(k)$, it can be easily verified that there exists a vertex v_i that R_i revisits at least $\Omega((k \cdot \sigma(k))^2)$ many times. Let P be the path defined above, and observe that there must exist two time instances, t_1, t_2 at which R_i visits v_i and such that $T' = [t_1, t_2]$ is devoid of agents in S and satisfying $|T'| > |P|$. (This is because the same agent can be at $|P|$ during at most $|P| + \sigma(k)$ many revisits of v_i by R_i.) Again, we can route R_i to a safe pendant vertex on P during T' (while avoiding agents in L if any).

Phase 2. We route each agent $R_i \in L$ from a safe vertex on E_i to another safe vertex on E_i as follows. Let $R_i \in L$ be an agent and suppose that R_i is located at a safe vertex v at the beginning of Phase 2. Define P_v to be the path from v to t_i plus a pendant vertex, chosen from E_i (which must exist), attached to every internal vertex in P_v. We route R_i along P_v to the farthest pendant safe vertex w in the direction of t_i that R_i can go to, moving nonstop (and avoiding agents in L), until it becomes within a distance of $k \cdot \sigma(k)$ from an agent in S or from its destination. Note that such a pendant safe vertex must exist, since the agents in S can visit fewer than $k \cdot \sigma(k)$ pendant vertices during Γ. If no such vertex w exists, then v must be already within distance $k \cdot \sigma(k)$ from an agent in S, in which case R_i will wait at v until it can achieve the above; note that the waiting time of R_i at any safe vertex is $O(k^2 \cdot \sigma(k))$. R_i then repeats. Note that such a waiting scenario cannot happen more than $k \cdot \sigma(k)$ many times, as after that R_i would never encounter any agent in S. When R_i is routed to w, we possibly avoid other agents in L along the way (using pendant vertices), and incurring a slack of at most $O(k)$ in the process. Phase 2 continues until each agent R_i in L is within distance at most $k \cdot \sigma(k)$ from t_i. Since the slack of each agent in L, $\rho(k)$, is much larger than $\beta(k)$, each agent in L still has large slack (larger than $\beta(k)$) when Phase 2 is complete.

Phase 3. We assume that R_i is located at a safe pendant vertex t_i' on $P_v \subseteq E_i$ that is within a distance of $k \cdot \sigma(k)$ from t_i; let $P_i' = t_i' t_i$. The arguments for routing R_i in Phase 3 are essentially symmetrical to those in Phase 1. The only difference is showing that we can route R_i from t_i' so that it arrives to a vertex w_i at time step x such that R_i is at w_i at time step x in Γ. R_i will then follow the schedule Γ after time step x. We again distinguish two cases, whereas the formal details of these are analogous to the corresponding ones in Phase 1:

- If $|Last_i|$ is "large", then we establish the existence of a safe vertex w_i in $Last_i$ that R_i' will be located at time step f_i during the last $\beta(k)$ many steps in Γ. Since the remaining slack of R_i is large (much larger than $\beta(k) > |Last_i|$), we can find a time interval during which R_i can be routed to w_i without colliding with any agents in S, and while avoiding the agents in L. R_i stays at w_i until time step f_i, where from that point on it follows Γ.
- Suppose now that $|Last_i|$ is "small". Similarly to the arguments made in Phase 1, either there exists a vertex $w_i \in Last_i$ that R_i waits at for a long time during the last $\beta(k)$ many steps, or that it revisits many times during the last $\beta(k)$ time steps. Again, we can find a time interval, where R_i can be routed from t_i' to w_i without collision, so that it arrives to it at the same time step as when it is there in Γ; R_i then follows Γ afterwards.

From the above, it follows that each agent reaches its destination. Moreover, the route of each agent in S remains unchanged. It suffices to show that each agent in L is confined to $B(P_i(x), \rho(k))$. This is clearly the case in Phase 1. By Lemma 7, each vertex can be occupied by an agent in S for at most $\sigma(k)$ many times steps. Moreover, if a vertex on a path of R_i is occupied, then this can happen at most $k \cdot \sigma(k)$ many times. It follows from the above that each agent in L is delayed at $3\beta(k)$ many steps, and hence resides in the desired ball. As for Phase 3, by construction each agent in L during that time interval resides in a ball of radius at most a small function of $\beta(k)$ from its destination.

We note that, in some of the above arguments, we did not explicitly include the other agents in L in our consideration since the agents in L have sufficient slack to wait at separate safe pendant vertices and to reconfigure among themselves. □

CONCLUDING REMARKS

The two presented algorithmic results substantially advance our understanding of the fundamental Multiagent Pathfinding problem on trees— a setting that is both foundational and routinely encountered in heuristic approaches [10, 17, 20]. The main question left open by our work is whether one could have the "best of both worlds": a fixed-parameter algorithm that solves Tree-MAPF *to optimality* on *all trees*. We conjecture that such an algorithm does indeed exist, and in fact Theorem 3 can be generalized to also solve all trees without 2-paths of length exceeding an arbitrary fixed function of k. Still, obtaining an exact fixed-parameter algorithm on the class of all trees remains beyond our current understanding and is far from a simple corollary of our techniques: the notion of *havens* that forms one of the two central pillars for Theorem 1 does not translate to the exact setting, while the way we decouple the schedule of large-slack agents from that of small-slack agents (in particular via the use of Lemma 6) will not work in the same way on general trees.

ACKNOWLEDGMENTS

A. Deligkas was supported by UKRI EPSRC (EP/X039862/1). R. Ganian was supported by the Vienna Science and Technology Fund (WWTF, project 10.47379/ICT22029) and the Austrian Science Fund (FWF, 10.55776/Y1329 and 10.55776/COE12). I. Kanj was supported by from DePaul URC Grants 606601 and 350130.

REFERENCES

[1] Oswin Aichholzer, Erik D. Demaine, Matias Korman, Anna Lubiw, Jayson Lynch, Zuzana Masárová, Mikhail Rudoy, Virginia Vassilevska Williams, and Nicole Wein. 2022. Hardness of Token Swapping on Trees. In *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany (LIPIcs, Vol. 244)*, Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 3:1–3:15. https://doi.org/10.4230/LIPICS.ESA.2022.3

[2] Roman Barták, Jiri Svancara, and Marek Vlk. 2018. A Scheduling-Based Approach to Multi-Agent Path Finding with Weighted and Capacitated Arcs. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar (Eds.). International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 748–756. http://dl.acm.org/citation.cfm?id=3237494

[3] Eli Boyarski, Ariel Felner, Pierre Le Bodic, Daniel Damir Harabor, Peter J. Stuckey, and Sven Koenig. 2021. f-Aware Conflict Prioritization & Improved Heuristics For Conflict-Based Search. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 12241–12248. https://doi.org/10.1609/AAAI.V35I14.17453

[4] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. 2015. *Parameterized Algorithms*. Springer. https://doi.org/10.1007/978-3-319-21275-3

[5] Argyrios Deligkas, Eduard Eiben, Robert Ganian, Iyad Kanj, and M. S. Ramanujan. 2024. Parameterized Algorithms for Coordinated Motion Planning: Minimizing Energy. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia (LIPIcs, Vol. 297)*, Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 53:1–53:18. https://doi.org/10.4230/LIPICS.ICALP.2024.53

[6] Erik D. Demaine and Mikhail Rudoy. 2018. A simple proof that the (n2-1)-puzzle is hard. *Theor. Comput. Sci.* 732 (2018), 80–84. https://doi.org/10.1016/J.TCS.2018.04.031

[7] Reinhard Diestel. 2012. *Graph Theory, 4th Edition*. Graduate texts in mathematics, Vol. 173. Springer.

[8] Rodney G. Downey and Michael R. Fellows. 2013. *Fundamentals of Parameterized Complexity*. Springer. https://doi.org/10.1007/978-1-4471-5559-1

[9] Eduard Eiben, Robert Ganian, and Iyad Kanj. 2023. The Parameterized Complexity of Coordinated Motion Planning. In *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA (LIPIcs, Vol. 258)*, Erin W. Chambers and Joachim Gudmundsson (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:16. https://doi.org/10.4230/LIPICS.SOCG.2023.28

[10] Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. 2023. Connected coordinated motion planning with bounded stretch. *Auton. Agents Multi Agent Syst.* 37, 2 (2023), 43. https://doi.org/10.1007/S10458-023-09626-5

[11] Sándor P Fekete, Phillip Keldenich, Dominik Krupke, and Joseph SB Mitchell. 2022. Computing coordinated motion plans for robot swarms: The cg: shop challenge 2021. *ACM Journal of Experimental Algorithmics (JEA)* 27 (2022), 1–12.

[12] Foivos Fioravantes, Dusan Knop, Jan Matyás Kristan, Nikolaos Melissinos, and Michal Opler. 2024. Exact Algorithms and Lowerbounds for Multiagent Path Finding: Power of Treelike Topology. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (Eds.). AAAI Press, 17380–17388. https://doi.org/10.1609/AAAI.V38I16.29686

[13] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* 4, 2 (1968), 100–107. https://doi.org/10.1109/TSSC.1968.300136

[14] Daniel Kornhauser, Gary L. Miller, and Paul G. Spirakis. 1984. Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications. In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*. IEEE Computer Society, 241–250. https://doi.org/10.1109/SFCS.1984.715921

[15] Athanasios Krontiris, Ryan Luna, and Kostas E. Bekris. 2013. From Feasibility Tests to Path Planners for Multi-Agent Pathfinding. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013, Leavenworth, Washington, USA, July 11-13, 2013*, Malte Helmert and Gabriele Röger (Eds.). AAAI Press, 114–122. https://doi.org/10.1609/SOCS.V4I1.18289

[16] Pascal Kunz, Hendrik Molter, and Meirav Zehavi. 2023. In Which Graph Structures Can We Efficiently Find Temporally Disjoint Paths and Walks?. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*. ijcai.org, 180–188. https://doi.org/10.24963/IJCAI.2023/21

[17] Duong Le and Erion Plaku. 2019. Multi-Robot Motion Planning With Dynamics via Coordinated Sampling-Based Expansion Guided by Multi-Agent Search. *IEEE Robotics Autom. Lett.* 4, 2 (2019), 1868–1875. https://doi.org/10.1109/LRA.2019.2898087

[18] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. 2021. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 11272–11281. https://doi.org/10.1609/AAAI.V35I13.17344

[19] Robert Morris, Corina S. Pasareanu, Kasper Søe Luckow, Waqar Malik, Hang Ma, T. K. Satish Kumar, and Sven Koenig. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *Planning for Hybrid Systems, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016 (AAAI Technical Report, Vol. WS-16-12)*, Daniele Magazzeni, Scott Sanner, and Sylvie Thiébaux (Eds.). AAAI Press. http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12611

[20] Erion Plaku, Kostas E. Bekris, Brian Y. Chen, Andrew M. Ladd, and Lydia E. Kavraki. 2005. Sampling-Based Roadmap of Trees for Parallel Motion Planning. *IEEE Trans. Robotics* 21, 4 (2005), 597–608. https://doi.org/10.1109/TRO.2005.847599

[21] Daniel Ratner and Manfred K. Warmuth. 1990. NxN Puzzle and Related Relocation Problem. *J. Symb. Comput.* 10, 2 (1990), 111–138. https://doi.org/10.1016/S0747-7171(08)80001-6

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

[23] Jamie Snape, Stephen J. Guy, Jur van den Berg, Ming C. Lin, and Dinesh Manocha. 2012. Reciprocal Collision Avoidance and Multi-Agent Navigation for Video Games. In *Multiagent Pathfinding, Papers from the 2012 AAAI Workshop, MAPF@AAAI 2012, Toronto, Ontario, Canada, July 22, 2012 (AAAI Technical Report, Vol. WS-12-10)*, Ariel Felner, Nathan R. Sturtevant, Kostas E. Bekris, and Roni Stern (Eds.). AAAI Press. http://www.aaai.org/ocs/index.php/WS/AAAIW12/paper/view/5247

[24] 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. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, Pavel Surynek and William Yeoh (Eds.). AAAI Press, 151–158. https://doi.org/10.1609/SOCS.V10I1.18510

[25] Pavel Surynek. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, Maria Fox and David Poole (Eds.). AAAI Press, 1261–1263. https://doi.org/10.1609/AAAI.V24I1.7767

[26] Manuela M. Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. 2015. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, Qiang Yang and Michael J. Wooldridge (Eds.). AAAI Press, 4423. http://ijcai.org/Abstract/15/656

[27] Peter R. Wurman, Raffaello D'Andrea, and Mick Mountz. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Mag.* 29, 1 (2008), 9–20. https://doi.org/10.1609/AIMAG.V29I1.2082

[28] Jingjin Yu and Steven M. LaValle. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*, Marie desJardins and Michael L. Littman (Eds.). AAAI Press, 1443–1449. https://doi.org/10.1609/AAAI.V27I1.8541