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

Optimal Path Planning in Hostile Environments

Andrzej Kaczmarczyk[1], Šimon Schierreich[1,2], Nicholas Axel Tanujaya[3], Haifeng Xu[4]

[1]Czech Technical University in Prague
[2]AGH University of Krakow
[3]Bina Nusantara University
[4]University of Chicago
andrzej.kaczmarczyk@cvut.cz, schiesim@fit.cvut.cz, nicholas.axel.135@gmail.com, haifengxu@uchicago.edu

Abstract

Coordinating agents through hazardous environments, such as aid-delivering drones navigating conflict zones or field robots traversing deployment areas filled with obstacles, poses fundamental planning challenges. We introduce and analyze the computational complexity of a new multi-agent path planning problem that captures this setting. A group of identical agents begins at a common start location and must navigate a graph-based environment to reach a common target. The graph contains hazards that eliminate agents upon contact but then enter a known cooldown period before reactivating. In this discrete-time, fully-observable, deterministic setting, the planning task is to compute a movement schedule that maximizes the number of agents reaching the target. We first prove that, despite the exponentially large space of feasible plans, optimal plans require only polynomially-many steps, establishing membership in NP. We then show that the problem is NP-hard even when the environment graph is a tree. On the positive side, we present a polynomial-time algorithm for graphs consisting of vertex-disjoint paths from start to target. Our results establish a rich computational landscape for this problem, identifying both intractable and tractable fragments.

1 Introduction

A humanitarian aid organization is planning to transport life-critical supplies through a conflict zone. The hostile actions of parties involved may severely constrain the planning task: these parties may attempt to divert or seize aid shipments. In particular, hostile forces monitor the locations under their control. When a convoy crosses such a location, it is detected and captured. However, capturing a convoy occupies the hostile patrol for a predictable period during which subsequent convoys may pass safely. How should the organization schedule dispatches to maximize the number of convoys reaching their destination?

This scenario motivates a more general planning problem. A central planner manages a set of identical agents at a common starting location. They seek to navigate as many of these agents as possible through a hostile environment to a designated target, potentially sacrificing some agents to enable others to pass. Beyond humanitarian logistics (van Wassenhove 2006; Rottkemper and Fischer 2013), this model captures applications such as cargo transport through pirate-prone waters (Jakob et al. 2012), UAV-based delivery of resources (Wu, Ramchurn, and Chen 2016), and analysis of cybersecurity intrusion detection where probe packets cause a temporary detector failure (Ptacek and Newsham 1998).

We formalize the environment as a graph whose vertices represent locations and whose edges represent feasible transitions. Two vertices are designated as the start and target. Hostility is modeled via traps placed at certain vertices. Each trap eliminates any agent that enters its vertex but then enters a reload period of known duration before reactivating. The planning task is to compute a movement schedule that maximizes the number of agents reaching the target. This formulation yields a multi-agent path planning problem with a new temporal-strategic dimension.

A rich literature spans related topics including constrained path finding (Phillips and Likhachev 2011; Ahmadi et al. 2021), multi-agent path finding (Stern et al. 2019), token reconfiguration (Călinescu, Dumitrescu, and Pach 2008), pursuit-evasion games (Borie, Tovey, and Koenig 2011), and humanitarian logistics in conflict zones (Boehmer et al. 2024) (see Section 2 for further discussion). However, to the best of our knowledge, the existing models do not address our specific problem. The key distinction lies in the nature of our traps, which are statically placed (limiting the applicability of game-theoretic frameworks), deterministic (making stochastic models unnecessarily general), and, crucially, which require a reload time after each capture. This reload mechanic is new and demands novel analytical techniques.

We briefly justify our modeling choices. First, reloading of traps captures deceptive strategies in which early agents absorb hostile attention allowing those who follow to pass. Second, we model a static rather than adversarial environment. Instead of analyzing repeated games, we focus on a single, short-horizon assignment within a known environment; an essential building block for longer-horizon decision-making. Third, while stochastic models may appear more realistic, they are notoriously difficult to solve in settings like ours. Furthermore, our deterministic approach isolates the fundamental computational structure of the problem and admits a natural interpretation as worst-case reasoning by a risk-averse planner. Finally, we assume that the reload durations are known. In practice, these can be estimated from historical data or knowledge of standard operational procedures.

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Our Contribution. We introduce a novel multi-agent path-planning model for transportation in hostile environments and analyze it from the perspective of computational complexity. First, we identify tractable fragments of the problem. We show that if the underlying topology is a simple path, then an optimal strategy can be computed in polynomial time. Despite the simplicity of this topology, our *run-wait-sacrifice* strategy requires surprisingly non-trivial arguments about the structure of optimal plans. We then extend these results to more general topologies whose condensation (see Definition 4) has a linear structure; these include, e.g., vertex-disjoint unions of paths. In contrast to these positive results, we establish computational hardness for modest generalizations. Specifically, we prove that the problem is NP-hard even when the underlying topology is a tree with maximum degree 3, or when each agent crosses at most 6 traps. *Check the full version for all the details (Kaczmarczyk et al. 2026).*

2 Related Work

Our problem is connected to various research streams in computer science, mathematics, operations research, artificial intelligence, and planning literature.

Classical Path Finding. The foundational problem underlying our model is Reachability: given a graph G and vertices s and t, determine whether t can be reached from s. This can be decided in linear time via a depth-first search, and the shortest paths can be computed in polynomial time using Dijkstra's algorithm (Dijkstra 1959) or, with heuristic guidance, the A* algorithm (Hart, Nilsson, and Raphael 1968). However, our setting differs fundamentally: the presence of traps with reload periods introduces temporal dynamics, and we must coordinate paths for multiple agents rather than a single one. Substantial work has also addressed reachability under *uncertainty* (Wagner and Choset 2017; Shofer, Shani, and Stern 2023), in *distributed* settings (Čáp, Vokřínek, and Kleiner 2015), and in *faulty* environments (Papadimitriou and Yannakakis 1991; Fried et al. 2013; Fioravantes et al. 2025b); in contrast, we assume a fully-informed central planner and deterministic, fault-free environment.

Dynamic and Temporal Environment. Recent research has explored path finding in graphs whose structure changes over time. In *temporal graphs*, edge availability is restricted to specific discrete time steps. While single-agent reachability remains polynomial-time solvable in temporal graphs (Wu et al. 2016), multi-agent variants become computationally hard even on simple paths (Klobas et al. 2023). In a different direction, Carmesin et al. (2023) and Dvořák et al. (2025) studied *self-deleting graphs*, where visiting a vertex removes certain incident edges. Our model differs from both: all edges remain available at all times, but agents can be eliminated by traps that subsequently enter a reload period.

Multi-Agent Path Finding. Our work is closely related to the extensively studied Multi-Agent Path Finding (MAPF) problem (Felner et al. 2017; Stern et al. 2019; Wang et al. 2025). In MAPF, each of k agents has a designated start and goal location, and the objective is to find collision-free paths minimizing makespan or total travel time. MAPF has been thoroughly analyzed from a computational complexity perspective (Surynek 2010; Yu and LaValle 2013; Nebel 2024; Fioravantes et al. 2024, 2025a; Deligkas et al. 2025), and numerous algorithmic approaches have been developed (Sharon et al. 2015; Li et al. 2019; Surynek 2022). Our problem differs in several key respects: (i) all agents share a common start and target, (ii) certain vertices contain traps that eliminate agents, and (iii) the objective is to maximize the number of surviving agents rather than minimizing travel time. These differences fundamentally change the problem structure.

Pursuit-Evasion Games. *Pursuit-evasion games* (Parsons 1976; Borie, Tovey, and Koenig 2011) study scenarios where pursuers attempt to capture evaders in a graph. The complexity of these problems depends heavily on the graph class and movement rules; for instance, determining the minimum number of pursuers needed to clear a graph is related to treewidth and pathwidth (Seymour and Thomas 1993). While superficially similar, our model differs substantially. Our traps are statically placed rather than actively pursuing agents, they operate deterministically rather than strategically, and crucially, they need to reload after each capture—a feature not considered in classical pursuit-evasion problems.

Token Reconfiguration. In *token reconfiguration* problems (Kornhauser, Miller, and Spirakis 1984; Hearn and Demaine 2005; Călinescu, Dumitrescu, and Pach 2008; van den Heuvel 2013), tokens placed on graph vertices must be moved from an initial configuration to a target configuration under various movement rules (sliding along edges or jumping to non-adjacent vertices). These problems are typically NP-complete and APX-hard on general graphs, though polynomial-time algorithms exist, e.g., for trees (Călinescu, Dumitrescu, and Pach 2008; Demaine et al. 2015). While our agents sometimes resemble moving tokens (and we use this analogy in some proofs), the key distinction is that our agents may be *eliminated* during transit, and the objective is to maximize the number of agents reaching the target.

Video Games and Motion Planning. A surprising connection exists between our work and certain video games. For example, in *Lemmings* (McCarthy 1998), agents spawn from a door and must be guided to a target, navigating hazards that become temporarily passable after eliminating an agent. The key difference is that Lemmings agents move autonomously and are controlled only indirectly via skill assignments, whereas we directly control agent movements. Moreover, Lemmings solutions can require exponentially many steps (Forišek 2010), and the decision problem is PSPACE-complete (Viglietta 2015); in contrast, we show that optimal plans in our model have polynomial length.

Security Games. Finally, there are several game-theoretical models of games related to security applications, like Stackelberg (Tambe 2012) or Blotto (Behnezhad et al. 2018; Kaźmierowski 2025) games. The closest to ours are the recently introduced *escape sensing games* (Boehmer et al. 2024), where one player tries to move as many of her assets from the start to target location, while an adversary uses sensors to detect some of the moving assets. In contrast, in our work the behavior of traps is implicit from their reload times.

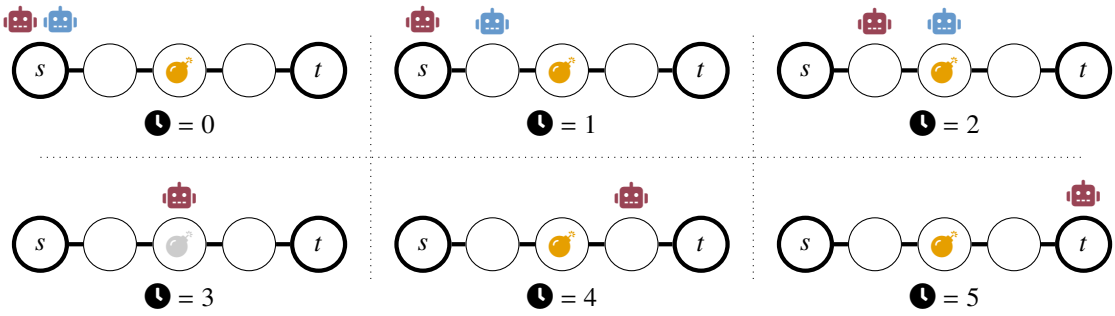


Figure 1: An illustration of our problem showing timesteps 0 to 5, as indicated by 🕒. There are two assets, **red** and **blue**. The reload time of the single trap r (💣) is $c(r) = 1$. If the trap is depicted in light gray (💣), then it is inactive in this round.

3 The Model

We model the space of possible moves as an undirected graph $G = (V, E)$ that we refer to as *topology*. Herein, the vertices represent certain important locations[1] and the edges are shortest connections between them. We assume that traversing an edge takes one unit of time. There are two distinguished locations $\{s, t\} \subseteq V$ — the *initial* and *target location*, respectively. We use n to denote the number of vertices of G. For some $V' \subseteq V$, we use $G[V']$ to denote the subgraph of G induced by V'. For some positive integer x, $[x]$ represents the set $\{1, 2, \ldots, x\}$.

We assume a set A of m *assets*[2] $a_1, \ldots, a_m$, which are placed at the initial location s at the beginning. Our goal is to move as many assets as possible to the target location t.[3] The assets move in discrete rounds using edges of the graph, and no pair of assets can occupy the same vertex (except for s and t) on the same round. Moreover, a subset of locations $R \subseteq V \setminus \{s, t\}$ are protected by *traps* $r_1, \ldots, r_\tau$. These traps are *known* in advance, and each trap r_i, $i \in [\tau]$, is associated with its *reload time* $c(r_i) \in \mathbb{N}$. Initially, each trap is *active*. If an asset enters a location protected by some active trap r_i, then the asset is eliminated, and r_i becomes *inactive*. If r_i is inactive, then assets can pass it without being eliminated. Trap r_i remains inactive for $c(r_i)$ rounds after it was triggered, and then becomes active again.

Example 1. *For an illustration of the problem, see the instance in Figure 1. There are two assets, red and blue, and a single trap with a reload time of one. Initially, both assets are placed on the initial vertex s. In the first round, the blue asset moves to the single neighbor of s and the red asset has to stay on s, as otherwise they would occupy the same vertex, which is forbidden. In the next round, both assets move one step toward t. The blue asset activates the trap, is eliminated, and the trap becomes inactive. Since the trap is inactive, the red asset can move to its location in the next round without being eliminated. However, in round* 4*, the red asset must leave this location, as the trap gets active again and would destroy the red asset. Finally, the red asset moves to the target location.*

The movement of assets is represented by a *movement plan*, or simply '*plan*.' Formally, introducing a special location $\dagger \notin V$ for the eliminated assets, a plan $\rho \colon A \times \mathbb{N}_0 \to V \cup \{\dagger\}$ assigns every asset $a \in A$ and timestep $j \in \mathbb{N}_0$ the location $v \in V \cup \dagger$ of a at time j. The plan ρ is *valid* if all the following conditions are satisfied:

1. $\rho(a, 0) = s$ for every $a \in A$ and, for every $j > 0$, if $\rho(a, j) = s$ then $\rho(a, j-1) = s$; that is, every asset starts at the initial location s and never returns there after leaving once,
2. if $\rho(a, j) \in \{t, \dagger\}$ for some $a \in A$ and $j \in \mathbb{N}_0$, then $\rho(a, j+1) = \rho(a, j)$; that is, assets cannot leave the target and special location after they reach them,
3. for every $a \in A$ and every $j \in \mathbb{N}$ it holds that if $\rho(a, j) \neq \dagger$, then $\rho(a, j)$ is adjacent to or equal to $\rho(a, j-1)$; that is, at each round assets move along the edges of the topology or stay put,
4. for two distinct assets $a, a' \in A$ and $j \in \mathbb{N}_0$, $\rho(a, j) = \rho(a', j)$ if and only if $\rho(a, j) \in \{s, t, \dagger\}$; that is, no two assets can stand on the same vertex v, unless v is the initial, target, or special location $\dagger$,
5. if $\rho(a, j) = r \in R$ for some $a \in A$ and $j \in \mathbb{N}$ and there is no $a' \in A$ and $\ell \in [c(r)]$ such that $\rho(a', \max\{0, j-\ell\}) = r$, then $\rho(a, j+1) = \dagger$; that is, if an asset steps on an active trap, it is eliminated,
6. for every $j \in \mathbb{N}$, if there is $a \in A$ such that $\rho(a, j) \notin \{t, \dagger\}$, then for at least one $a' \in A$ we have $\rho(a', j) \neq \rho(a', j+1)$; that is, as long as some asset remains, at least one asset moves each round, and
7. there is some $\mathrm{len}(\rho) \in \mathbb{N}$ such that for every $a \in A$ it holds that $\rho(a, \mathrm{len}(\rho)) \in \{t, \dagger\}$, and there is an asset $a \in A$ with $\rho(a, \mathrm{len}(\rho) - 1) \notin \{t, \dagger\}$; that is, all assets either reach the target or are eliminated in finite time.

Unless stated explicitly, we assume only valid plans. For a valid plan ρ of *length* $\mathrm{len}(\rho)$, we say that the plan *ends* at $\mathrm{len}(\rho)$, and we say that $a \in A$ *survived* if $\rho(a, \mathrm{len}(\rho)) = t$; otherwise, *a failed*.

Example 2. *A plan capturing the movement of the assets as described in Example 1 is as follows (we assume that the internal vertices of G are, from left to right,* v_2*,* v_3*, and* v_4*).*

[1] We use the term location and vertex interchangeably.

[2] We prefer "assets" over "agents," as the former is more general as it encompasses both autonomous and non-autonomous entities.

[3] We do not consider other goals, such as the total time of all assets reaching the target location.

ρ	0	1	2	3	4	5
🤖	s	v_2	v_3	†	†	†
🤖	s	s	v_2	v_3	v_4	t

It is easy to verify that this plan is valid and has length $\text{len}(\rho) = 5$.

We study the above-defined model through the lens of computational complexity of the following decision problem.

Routing Plan	
Input:	A set of assets A, a topology $G = (V, E)$, distinct initial and target vertices $s, t \in V$, a set of traps $R \subseteq V \setminus \{s, t\}$, a reload time function $c\colon R \to \mathbb{N}$, and a goal $k \in [\lvert A\rvert]$.
Question:	Is there a valid plan ρ yielding at least k surviving assets?

Remark 1. *Unlike in (some variants of) Multi-Agent Path Finding, we do not explicitly forbid two assets from swapping positions across an edge. However, since our assets are indistinguishable, any swap of two assets a_i and a_j in a plan ρ can be eliminated by the following transformation: (i) At the time of the swap, let both assets wait at their respective pre-swap positions (i.e., before they would have crossed the edge). (ii) After that time, assign to a_i the remainder of a_j's plan, and to a_j the remainder of a_i's plan. By exhaustively applying this transformation to all swaps in ρ, one obtains a modified plan without any swaps, achieving the same number of surviving assets, with identical temporal performance. Thus, when the application requires a plan without swaps, we may enforce this restriction without loss of optimality.*

4 The Existence of Short Plans

The properties required in the definition of a valid plan directly forbid some unreasonable solutions, such as infinite plans or keeping all assets at the initial location without moving any of them. However, it is not strong enough to forbid very lengthy plans, e.g., plans whose length is exponential in the input size. The following theorem shows that a plan ρ ensuring that k assets survive implies a *short* plan ρ' with at least the same number of surviving assets.

Theorem 1. *Let $\mathcal{I} = (G, s, t, R, c, A, k)$ be an instance of Routing Plan and let ρ be a plan of length ℓ with at least k surviving assets witnessing that $\mathcal{I}$ is a* Yes*-instance. Then, there is a plan ρ' of length $\ell' \in \text{poly}(n, m)$ with at least k surviving assets.*

Proof. Our first step is to transform the given network G into a vertex-colored graph that depends on ρ. Given the new colored graph, we interpret movement plans as graph-coloring reconfigurations and combine existing results with new observations to obtain our theorem.

Definition 1. *A* simple transform *of a network $G = (V, E)$ and a number k of surviving assets for some plan, is a graph $\hat{G} = (V \cup S \cup T, E \cup E')$ such that vertex set S consists of $m - 1$ vertices arranged in a path attached at one of its ends to $s \in V$, vertex set T consists of $k - 1$ vertices arranged in a (different) path attached with one of its ends to t, and E' contains the respective edges.*

To fully exploit the simple transform and capture the dynamics of moving assets, we define a specific graph coloring.

Definition 2. *For some timestep $i \in \mathbb{N}$, an* i-snapshot *of a network $G = (V, E)$ and a plan ρ with length at least i is tuple $\text{snap}_i^{G,\rho} = (\hat{G}, h)$, where $\hat{G} = (V \cup S \cup T, E \cup E')$ is the simple transform of G and ρ and h is a binary coloring of the vertices of $\hat{G}$ such that:*

1. *for each $v \in V$, $h(v) = 1$ if there is an asset $a \in A$ that, according to ρ, occupies v at time $i \in \mathbb{N}$, $h(v) = 0$ otherwise;*
2. *given that the number x of assets occupy the initial vertex s at time i, for $(x - 1)$ vertices $v \in S$ whose distance is at most $x-1$ from s, $h(v) = 1$, and $h(v) = 0$ for the remaining vertices in S; and*
3. *given that the number y of assets occupy the terminal vertex t at time i, for $(y-1)$ vertices $v \in T$ whose distance is at most $(y - 1)$ from t, $h(v) = 1$, and $h(v) = 0$ for the remaining vertices in T.*

For readability, we use snap_i whenever it is unambiguous.

Conveniently, the number of timesteps needed to reach one snapshot from another is upper-bounded by a polynomial.

Lemma 1. *Given a network $G = (V, E)$, an empty set $R = \emptyset$ of traps, a plan ρ, and two snapshots $\text{snap}_i^{G,\rho}$ and $\text{snap}_j^{G,\rho}$ for $i \leq j \leq \text{len}(\rho)$, there is a plan ρ' of length $\text{len}(\rho') < (2m+n)^2$ such that $\text{snap}_0^{G,\rho'} = \text{snap}_i$ and $\text{snap}_{\text{len}(\rho')}^{G,\rho'} = \text{snap}_j$.*

Proof. By definition, snapshots $\text{snap}_i^{G,\rho}$ and $\text{snap}_j^{G,\rho}$ are in fact two different colorings h_i and h_j, both with two colors, of the same graph $\hat{G} = (V \cup S \cup T, E \cup E')$, which has $\hat{n} := m - 1 + n + k - 1 < 2m + n$ vertices, as $k \leq m$.

Given this interpretation, to find the claimed moving plan means to obtain a 2-coloring h_j from that of h_i via a series of certain steps. Specifically, in each step, we can swap colors of a pair of adjacent vertices of mutually distinct color. Indeed, such a step represents moving an asset from the vertex it occupies (colored 1 by definition of a snapshot) to some vertex that no asset occupies (colored 0 by the same definition).

It is sufficient to only upper-bound the number of steps needed for transforming the colorings. To do so, we directly apply the result of Yamanaka et al. (2018, Lemma 1) stating that the upper bound is $\binom{\hat{n}}{2} < (2m + n)^2$, as we claim. ◀

We introduce a restricted snapshot that allows us to analyze snapshots of a certain part of the topology.

Definition 3. *Consider some timestep $i \in \mathbb{N}$, a network $G = (V, E)$, a plan ρ, an i-*snapshot *$\text{snap}_i^{G,\rho} = (\hat{G}, h)$, where $\hat{G} = (V \cup S \cup T, E \cup E')$, and a subset $V' \subseteq V$. Let $\widehat{V}$ be a copy of V' augmented with S if $s \in V'$ and with T if $t \in V'$. Then, a V'-*restricted *i-*snapshot *$\text{snap}_i^{G,\rho}{\restriction_{V'}}$ is a tuple $(\hat{G}[\widehat{V}], h{\restriction_{\widehat{V}}})$, where $h{\restriction_{\widehat{V}}}$ is the restriction of h to vertices in $\widehat{V}$.*

It is immediate that Lemma 1 can be applied to a pair of snapshots restricted to a set of vertices that do not contain traps and form a connected component.

Corollary 1. *Given a network $G = (V, E)$, a set $R \subset V$ of traps, a plan ρ, a subset $V' \subseteq V$ such that $V' \cap R = \emptyset$ and $G[V']$ is a connected component, and two snapshots $\mathrm{snap}_i^{G,\rho} \restriction_{V'}$ and $\mathrm{snap}_j^{G,\rho} \restriction_{V'}$ for $i \leq j \leq \mathrm{len}(\rho)$ whose number of vertices colored 1 are the same, there is a plan ρ' of length $\mathrm{len}(\rho') < (2m+n)^2$ such that $\mathrm{snap}_0^{G,\rho'} \restriction_{V'} = \mathrm{snap}_i^{G,\rho} \restriction_{V'}$ and $\mathrm{snap}_{\mathrm{len}(\rho')}^{G,\rho'} \restriction_{V'} = \mathrm{snap}_j^{G,\rho} \restriction_{V'}$.*

We now show that contracting a part of a plan between two snapshots is sufficient to contract the whole plan.

Lemma 2. *Given an instance (G, s, t, R, c, A, k) of Routing Plan, let ρ be a valid plan of length $\mathrm{len}(\rho) = x + \delta + y$ in which at least k assets survive such that ρ does not activate any trap in timesteps $x + 1$ to $x + \delta - 1$, inclusive. If there is a (non-valid) plan ρ'' of length $\mathrm{len}(\rho'') = x + \delta''$ such that*

1. *for each asset $a \in A$ and all $i \in [x]$ it holds that $\rho(a, i) = \rho''(a, i)$;*
2. *$\mathrm{snap}_{x+\delta''}^{\rho''} = \mathrm{snap}_{x+\delta}^{\rho}$;*
3. *plan ρ'' does not activate any trap in timesteps $x + 1$ to $x + \delta'' - 1$ inclusive; and*
4. *$\delta'' < \delta$;*

then there exists a valid plan ρ' of length $\mathrm{len}(\rho') = x + \delta'' + y$ with at least k surviving assets.

Proof. We build plan ρ' by copying ρ'' and then extending it with the last y steps of ρ, adapting them if needed.

Consider running ρ' according to ρ'' until timestep $x + \delta''$. Since $\mathrm{snap}_{x+\delta''}^{\rho'} = \mathrm{snap}_{x+\delta}^{\rho}$, we now attempt to replicate in ρ' every timestep j, in the natural order, from $x + \delta + 1$ to $x + \delta + y$ according to plan ρ. Note that we certainly succeed for the timestep $x + \delta + 1$ because $\mathrm{snap}_{x+\delta}^{\rho} = \mathrm{snap}_{x+\delta''}^{\rho'}$.

For some j, the replication might become impossible. That is, there is an asset a such that at timestep j of ρ this asset should occupy a vertex v occupied by another asset a'. Our intention is to leave a intact at vertex v at timestep j but this, however, might invalidate the move of another asset at the same time, the one that potentially was supposed to take position at v instead of a. Hence, we collect those assets that we cannot move due to leaving a intact. On the graph, these assets can form either a path or a cycle. In both cases, we leave *all* of these assets intact at timestep j. Since we are considering the first such j for which mimicking ρ failed, the described situation can happen only because of some asset a' that was destroyed at time j. This means, in particular, that a' moves at time j to the destroyed state, †, in plan ρ. However, we also let a' stay intact, as it is now not destroyed; for, in the opposite case a would have been able to take its place. It is not hard to verify that because of these alignments, we obtain a snapshot $\mathrm{snap}_j^{\rho'}$ whose set O of vertices colored 1 is a superset (potentially a strict superset) of that of snap_j^{ρ}. Put in words, the former contains all assets at the same vertices as that of the latter, and, second, the former has maybe even more assets. For the next step, we relabel the assets in our plan ρ such that they meet exactly the positions of vertices in our just-extended plan ρ'. There will be assets that cannot be matched, and these will be deemed to stay put in ρ', unless another conflict like the described one appears.

As a result of running the whole "copying" procedure, we obtain ρ' that loses at most as many assets as the original plan ρ. Indeed, our procedure of copying never decreased the number of non-destroyed assets. Potentially, the remaining vertices might be directed to the final vertex, if that turns out to be possible. ◀

We combine the previous observations into the following crucial lemma, which upper-bounds the number of timesteps between two subsequent activations of traps.

Lemma 3. *Let $k_1, \ldots, k_q$, such that $k_i \leq k_{i+1}$ for each $i \in [q]$ be the timesteps at which plan ρ activates traps. Then, if there is $i \in [q-1]$ such that $k_{i+1} - k_i > (2m+n)^2$, then there is a plan ρ' such that $\mathrm{snap}_0^{G,\rho'} = \mathrm{snap}_0^{G,\rho}$ and $\mathrm{snap}_{\mathrm{len}(\rho')}^{G,\rho'} = \mathrm{snap}_{\mathrm{len}(\rho)}^{G,\rho}$ and such that in the trap activation series $k'_1, \ldots, k'_q$ of ρ' it holds that $k'_{i+1} - k'_i < (2m+n)^2$.*

Proof. Let us fix i such that $k_{i+1} - k_i > (2m+n)^2$, and let $K \subseteq V$ be the set of vertices passed by at least one asset between timesteps k_i and k_{i+1}, inclusive. Now, let us consider the graph $G' = G[V \setminus R \cup K]$. Note that the vertices of G' are either non-trap vertices or vertices whose trap is deactivated between timesteps k_i and k_{i+1}, so they can be considered non-trap vertices during that period.

Let $C_1, C_2, \ldots C_x$ be the connected components of G' (clearly, there must be at least one). According to Corollary 1, for each component C_j, $j \in [x]$, there is a plan ρ_j such that $\mathrm{snap}_{k_i+1}^{\rho} \restriction_{C_j} = \mathrm{snap}_{k_i+1}^{\rho_j} \restriction_{C_j}$ and $\mathrm{snap}_{k_{i+1}-1}^{\rho} \restriction_{C_j} = \mathrm{snap}_{k_{i+1}-1}^{\rho_j} \restriction_{C_j}$. Let L be the maximum length among these new plans $\rho_1, \rho_2, \ldots, \rho_x$. We build a new plan ρ' as follows. We start with copying the plan ρ for the first k_i timesteps for all assets. Then, we let each group A_j of assets that remained in the respective connected component C_j at timesteps $k_i + 1$ to $k_{i+1} - 1$ follow their plan ρ_j for a $\mathrm{len}(\rho_j)$ timesteps. For each such j that $\mathrm{len}(\rho_j) < L$, we let the respective assets from group A_j stay put for the next $L - \mathrm{len}(\rho_j)$ timesteps. Finally, we again let all assets follow the original plan ρ, which completes the construction of ρ'.

Naturally, $\mathrm{snap}_y^{G,\rho'} = \mathrm{snap}_y^{G,\rho}$ for each $y \in [0, k_i]$ as the respective plans are identical at these timesteps. By applying Corollary 1 to plans $\rho_1, \ldots, \rho_x$, copying these plans into ρ', and enforcing the same length of L by forced idling, it holds that $\mathrm{snap}_{k_i+L}^{G,\rho'} = \mathrm{snap}_{k_{i+1}-1}^{G,\rho}$. Note that by the fact that the vertices of G' are either non-trap vertices or vertices for which their trap is deactivated between timesteps k_i and k_{i+1}, applying the forced idling does not activate any trap. Finally, for the remaining part of plans ρ and ρ', we note that they meet the conditions of Lemma 2. So, we apply this lemma and get that in plan ρ' there are at least as many surviving assets as in plan ρ. ◀

To get the result, we exhaustively apply Lemma 3. □

Due to the previous theorem, we can focus only on deciding whether there is a short solution to the Routing Plan problem. Hence, for the rest of the paper, we assume only short solutions without explicitly specifying it. Therefore, we obtain the following.

Theorem 2. *Routing Plan is in NP.*

Proof. By Theorem 1, if a given instance $\mathcal{I}$ is a Yes-instance, then there exists a certificate (plan ρ) of polynomial length. Therefore, to verify the certificate, we simulate it, check its validity at every step, and observe whether at least k assets have been successful. □

5 Efficient Algorithms

The containment in NP established in the above section is just the first step on the way of understanding the hardness of Routing Plan. In particular, it does not imply efficient solvability, a trait potentially of practical relevance. So, can we do better and solve Routing Plan in polynomial time?

Chasing the answer to the posed question, let us first make several helpful assumptions on the input of Routing Plan. By this, we will avoid distracting trivial cases and simplify the formal analysis of the computational complexity of our problem. To this end, let us fix some instance $\mathcal{I} = (G, s, t, R, c, A, k)$ of Routing Plan. Then, we assume without loss of generality that:

1. G contains an s-t path, as otherwise $\mathcal{I}$ is trivially a No-instance;
2. G is connected; if not, using the breadth first search, we identify in polynomial time all components of G and narrow the topology down to the one containing s and t;
3. $G[V \setminus R]$ does not contain an s-t path; otherwise, $\mathcal{I}$ is a Yes-instance because all assets survive by following the trap-free path in question (note that $k \leq |A|$).

Having identified and dismissed the obvious cases, our first result delivers good news. It turns out that Routing Plan is efficiently solvable for path topologies by following an intuitive *run-wait-sacrifice* plan. Here, each asset always moves towards the target—runs—if only the following location is unoccupied and is not an active trap, waits before an active trap r until it is followed by at least $c(r)$ assets, and then sacrifices itself by stepping on r and thus letting the $c(r)$ followers pass the now-inactive r. Contrary to the general simplicity of the run-wait-sacrifice approach, the proof of its optimality involves a complex and careful analysis.

Theorem 3. *If G is a path, Routing Plan is solvable in polynomial time.*

Proof. Our algorithm is built on top of auxiliary Claims 1 to 3 below that show what an optimal solution for path graphs looks like. Prior to focusing on the structure of a solution, we make several observations to get rid of trivial cases. Without loss of generality, we assume in the rest of the proof that (i) $\deg(s) = 1$, as by property 1 of a valid plan, no asset can return to s once it leaves it, (ii) $\deg(t) = 1$, as by property 2 of valid plan, no asset can leave t once it reaches it. Obviously, both properties heavily rely on the fact that the topology under consideration is a simple path. As a consequence, we have that every solution plan follows the shortest s, t-path. For the rest of the proof, we assume that $V(G) = \{v_1, \ldots, v_n\}$, $E(G) = \{\{v_i, v_{i+1}\} \mid i \in [n-1]\}$, and $s = v_1$ and $t = v_n$.

Previous properties show that the assets use solely vertices of the shortest s, t-path. However, they do not necessarily ensure that assets are not "returning" to previously left locations. In our first auxiliary claim, we prove that we can indeed assume only plans where, in each step, assets are either waiting on their current locations, or they are moving closer to the target location t.

Claim 1. *If an instance $\mathcal{I}$ of Routing Plan, where G is a path, admits a solution plan ρ, then it also admits a solution plan ρ' such that for every $a \in A$ and every timestep j we have $\rho'(a, j+1) = \rho'(a, j)$; $\rho'(a, j) = v_i$ and $\rho'(a, j+1) = v_{i+1}$ for some $i \in [n-1]$; or $\rho'(a, j+1) = \dagger$.*

Next, we show that if there is an asset that can move to an empty vertex closer to the target, that is, a vertex that is not a trap and is not occupied by another asset, this asset can always move to this vertex.

Claim 2. *Let ρ be a solution plan, $a \in A$ be an asset such that $\rho(a, j) = v_i$, $i \in [n-1]$, $\rho(a, j+1) \neq \dagger$, v_{i+1} is not an active trap in round $j+1$, and there is no $a' \in A \setminus \{a\}$ such that $\rho(a', j+1) = v_{i+1}$. Then, there is a solution plan ρ' such that $\rho'(a, j+1) = v_{i+1}$.*

That is, according to Claim 2, assets do not wait at the same vertex if they can move closer to the target location t. The same claim also implies that whenever an asset is at location v_{n-1}, it immediately moves to $v_n = t$ in the next round (unless it is destroyed by a trap). However, the necessary condition for the movement of such asset is that the neighboring vertex is not a trap. In such situation, on the other hand, we show that waiting is highly desirable. However, for this, we need more notation. Let $r_1, \ldots, r_\tau$ be an ordering of traps such that $\mathrm{dist}(r_i, t) > \mathrm{dist}(r_{i+1}, t)$ for every $i \in [\tau - 1]$. We call the sub-path between traps r_i and r_{i+1} a *segment*, and we denote it as S_i.

Claim 3. *Let ρ be a solution plan, $a \in A$ be an asset such that $\rho(a, j) = v_\ell$, $\ell \in [n]$, $v_{\ell+1} \in R$, let the trap r_{i+1} on $v_{\ell+1}$ be active in round $j+1$. Then, there is a solution plan ρ' such that a waits on v_ℓ and moves to $v_{\ell+1}$ only after all $v_{\ell-1}, \ldots, v_{\ell-\ell'}$, where $\ell' = \min\{|S_i|, c(r_{i+1}), |\{b \in A \setminus \{a\} \mid \rho(b, j) = v_{\ell''} \wedge \ell'' \leq \ell\}|\}$, are occupied by some assets.*

In other words, Claim 3 says that no trap is activated if there are not enough assets to follow the destroyed asset. This rule forces us to pass traps efficiently.

Our algorithm exploits the structure of solutions described by Claims 1 to 3. We arbitrarily fix an order of the assets, and let them leave s in consecutive rounds. After the first asset passes the first trap, we apply the following strategy for all assets in order. Each asset moves to the next vertex v if (i) v is unoccupied or (ii) v is an active trap and there are enough assets to follow (according to Claim 3). In all other cases, the asset in question stays put. Optimality and correctness of this approach follows from the previous claims. □

The run-wait-sacrifice strategy proves to be a crucial ingredient for extending the family of polynomial-time solvable instances of Routing Plan with the case of a collection of disjoint paths connecting the initial and target locations. Intuitively, this scenario models a collection of independent routes to reach a goal. After figuring out how many assets survive taking each of the available routes, the decision mostly depends on how many assets to deploy to each of the available methods. The latter task resembles making a change given a fixed set of coin denominations, a canonical example of dynamic programming (DP). Hence, we devise a DP-based algorithm similar to that solving the coin change problem, using run-wait-sacrifice as a subroutine.

Proposition 1. *If $G \setminus \{s,t\}$ is a disjoint union of paths, Routing Plan can be solved in polynomial time.*

Aiming at further exploiting the run-wait-sacrifice technique, we show that it performs well for topology structures reaching far beyond those having an "obviously" linear structure (or a collection thereof). For intuition, consider some single *s*-*t* path. Run-wait-sacrifice achieves optimality by waiting with passing a forthcoming trap until it "stores" enough assets (or as many of them as possible) on non-trap locations between the previous and the forthcoming trap. However, because assets are indistinguishable, the structure of the non-trap locations used for storage is irrelevant (given that it does contain any edge connecting to any other "storages") for optimality. In particular, run-wait-sacrifice optimality retains after attaching an arbitrary connected component of non-trap vertices to some non-trap vertex of the path.

Before formalizing the intuition above, let us introduce the notion of a *condensed topology*. Intuitively, we obtain a condensed topology from some topology by contracting every "connected component" of solely non-trap (and non-terminal) vertices of the topology into a "supernode" representing the contracted non-trap vertices; formally:

Definition 4. *Let $G = (V,E)$ be a topology, $R \subseteq V$ be a collection of traps, $C = \{C_1, C_2, \ldots, C_k\}$ be a collection of connected components of graph $G[V \setminus R \setminus \{s,t\}]$, where each C_i, $i \in [k]$, has vertices V_i^c. Then, a* condensed topology *is an undirected graph $G' = (V', E')$, where $V' = \{v_i' \colon C_i \in C\} \cup R \cup \{s,t\}$ and E' contains an edge $e = \{u,w\}$, $\{u,w\} \subset V'$, if $e \in E$, and contains an edge $\{v_i, w\}$, $i \in [k]$, $w \in R \cup \{s,t\}$, if there is a vertex $u' \in C_i$ such that $\{u', w\} \in E$.*

Now, a careful analysis of the run-wait-sacrifice strategy allows us to extend our catalog of polynomial-time solvable instances of Routing Plan further to topologies whose condensations have a linear structure.

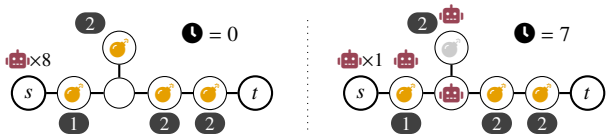
Proposition 2. *Let $\mathcal{I} = (G, s, t, R, c, A, k)$ be an instance of Routing Plan such that each $r \in R$ has degree two in G and all s-t paths in the respective condensed topology are mutually disjoint. Then $\mathcal{I}$ is solvable in polynomial time.*

As a natural next step beyond the topologies of a linear structure (or a collection thereof), we explore a broader class of trees. We thus start with stars, structurally very simple trees. By our initial assumptions on the topology G, the only non-trivial case to consider is that the start and target vertex are separated with one vertex being a trap. For this case, however, run-wait-sacrifice yields an optimal solution.[4] In passing, we mention that precisely the considered case is the reason why we cannot apply Proposition 2, which disallows non-trap vertices connected solely to trap vertices.

Proposition 3. *If G is a star, then Routing Plan can be solved in $O(1)$ time.*

The crux of the positive results above lies mostly in exploiting the topology structure that limits the number of sensible ways to get from the start location to the target one. In particular, we never took a step at a trap vertex that did not bring us closer to the target. As we demonstrate in the following example, this may not be the case even for relatively small topologies. Sometimes, achieving optimality requires taking a strategic non-obvious detour from an *s*-*t* path, on which we sacrifice assets to use a certain trap vertex as "storage."

Example 3. *Consider the following instance with 8 assets.*



Ignoring the upper trap yields the case of a path, for which we know that run-wait-sacrifice is optimal. Simulating run-wait-sacrifice shows that no asset survives, as to pass the two traps before t we would need three consecutive assets; this is impossible due to the trap with reload time one. Yet, there is a plan yielding 1 successful asset achievable by sending all assets one by one and using the upper trap. First, one asset activates the upper trap at time 5. This unlocks the situation depicted in the right picture at time 7 that can evolve in the next round to having the required three consecutive assets (at the untrapped location and its sideways neighbors).

6 Limits of Tractability

We now turn to exploring the limits of efficient computability by showing the computational intractability of Routing Plan by proving NP-hardness. In general, our goal is to show the hardness for instances that are as constrained as possible, thereby precisely identifying the borders of tractability.

Theorem 4. *Routing Plan is NP-complete, even if each asset crosses at most six traps.*

Proof. We reduce from the NP-complete (Gonzalez 1985) Restricted Exact Cover by 3-Sets problem. The input of this problem consists of a universe $\mathcal{U} = \{u_1, \ldots, u_{3N}\}$ and a family of subsets $\mathcal{S} = (S_1, \ldots, S_{3N})$ such that $\forall j \in [3N] \colon S_j \in \binom{\mathcal{U}}{3}$ and each element $u \in \mathcal{U}$ appears in exactly 3 subsets $S \in \mathcal{S}$. The goal is to decide whether there is a collection $C \subseteq \mathcal{S}$ of size N such that $\bigcup_{S \in C} S = \mathcal{U}$.

Let $\mathcal{I}$ be an instance of the RX3C problem. We construct an equivalent instance $\mathcal{J}$ of Routing Plan using several gadgets. We first focus on them one by one, and then describe how they work together; Figure 2 illustrates the whole construction.

[4] More precisely, "run-sacrifice" works better in this case, as the trap is a neighbor of the initial location, so no asset needs to wait.

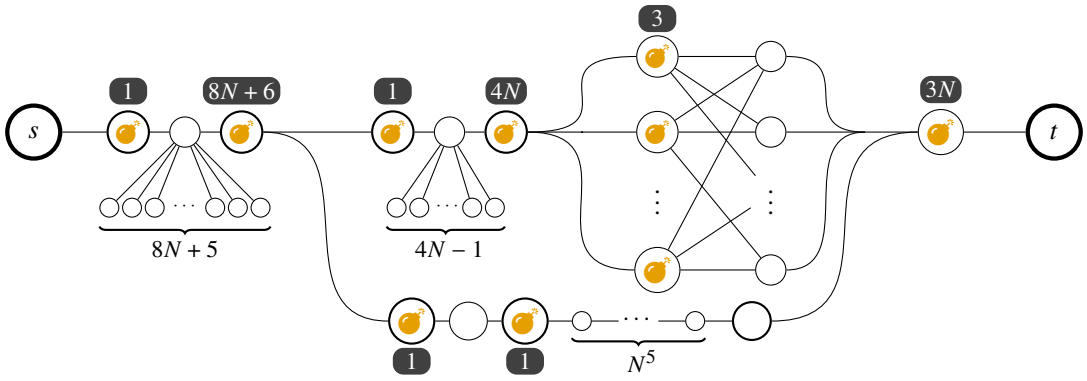


Figure 2: Illustration of the construction from Theorem 4. The dark labels by the traps (🍊) depict the respective reload times.

The core ingredient of our construction is a *batch gadget* $B(x)$, which ensures that, assuming an optimal solution in terms of the number of lost assets and enough assets on the input, x assets leave the gadget in consecutive rounds; i.e., in one batch. Formally, the batch gadget is a star with $x + 1$ leaves $v_0, \ldots, v_x$ and the center c. The leaf v_0 is called an *input port* and the leaf v_x is an *output port*. The gadget contains two traps. The first is placed on the input port with reload time one. The other is placed on the output port with reload time equal to the parameter x. The crucial idea of the gadget is that we need to store $x - 1$ assets on the leaves $v_1, \ldots, v_{x-1}$ and only after all the leaves are occupied, the trap on the output port is deactivated. Observe that due to the trap at the input port, we lose at least one asset for every asset that ever visits the center c, and we lose at least one asset due to the trap on the output port. Consequently, to pass x assets through the gadget, we lose at least $x + 2$ assets.

Now, we formally describe the construction (see Figure 2 for an illustration). We have three batch gadgets besides the initial vertex s and terminal vertex t. The first batch gadget $B(8N + 6)$ is connected through its input port to the initial vertex. The second batch gadget $B(4N)$ is connected to the first batch gadget and is followed by $3N$ *set vertices* $s_1, \ldots, s_{3N}$, each protected by a trap with reload time 3. Next, we have $3N$ *element vertices* $u_1, \ldots, u_{3N}$, which are unprotected. There is an edge $\{s_j, u_i\}$ if and only if $u_i \in S_j$ and all element vertices are connected with a *guard vertex* g. Moreover, there is a trap protecting the guard vertex with reload time $3N$. The third batch gadget $B(1)$ is also connected to the first gadget and is followed by a *slowdown path* of length N^5, which again ends in the guard vertex. The guard vertex is connected to the terminal vertex. Finally, we set $A = \{a_1, \ldots, a_{16N+14}\}$ and $k = 3N$.

The whole construction guarantees that the assets split between the set vertices, and the slowdown path. In an optimal solution, at some point, all element vertices are occupied by assets waiting for a couple of rounds for an asset using the slowdown path. This asset eventually deactivates the trap on the guard vertex and allows all assets waiting on the element vertices to pass through g to the terminal vertex t. □

Furthermore, by reducing from the NP-hard 3-Partition problem (Garey and Johnson 1975), we rule out polynomial-time computability for trees with maximum degree 3.

Theorem 5. *Routing Plan is NP-complete, even if the topology G is a tree with maximum degree 3.*

In light of the established hardness, it is natural to consider approximations. Let Routing Plan* be a natural optimization variant of our problem, in which we seek the maximum number of assets that survive. Importantly, the proof of Theorem 5 excludes a constant-factor approximation algorithm for Routing Plan* (under standard assumptions).[5]

Theorem 6. *There is no polynomial-time algorithm yielding a multiplicative constant-factor approximation to Routing Plan*, even if topology G is a tree with maximum degree 3.*

7 Conclusions

We introduced a new planning model for adversarial environments in which agents may be eliminated while traversing hostile terrain. We established that the problem is in NP despite the exponentially large plan space, proved NP-hardness even on trees of constant degree, and identified several tractable cases. Our framework captures applications ranging from humanitarian aid transportation in conflict zones to coordinated drone swarm navigation.

Our results open multiple promising avenues for future research. In this initial study, we assumed complete knowledge of the environment. Since such full observability is rare in real-world scenarios, a natural extension is to incorporate *uncertainty* (Nikolova, Brand, and Karger 2006; Wagner and Choset 2017; Shofer, Shani, and Stern 2023) in trap locations, reload times, or graph topology.

Additionally, our hardness results motivate the design and analysis of efficient *heuristics and approximations*. While constant-factor approximations may be limited to special cases due to our inapproximability result, heuristics, albeit without formal optimality guarantees, could yield high-quality solutions in many practical instances.

Finally, it is interesting to consider *multiple start or target locations* to broaden the model's applicability to scenarios such as distributed supply networks or multi-objective delivery missions.

[5] Routing Plan* is not only outside of APX but also APX-hard.

Acknowledgments

Andrzej Kaczmarczyk and Haifeng Xu were partially supported by the ONR Award N00014-23-1-2802. Šimon Schierreich was partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 101002854). Additionally, Andrzej Kaczmarczyk and Šimon Schierreich were partially funded by the European Union under the project Robotics and advanced industrial production (reg. no. CZ.02.01.01/00/22_008/0004590). Most of this work was done while Šimon Schierreich was a Fulbright-Masaryk Fellow at Pennsylvania State University.

References

Ahmadi, S.; Tack, G.; Harabor, D. D.; and Kilby, P. 2021. A Fast Exact Algorithm for the Resource Constrained Shortest Path Problem. In *Proceedings of AAAI-21*, 12217–12224.

Behnezhad, S.; Blum, A.; Derakhshan, M.; Hajiaghayi, M. T.; Mahdian, M.; Papadimitriou, C. H.; Rivest, R. L.; Seddighin, S.; and Stark, P. B. 2018. From Battlefields to Elections: Winning Strategies of Blotto and Auditing Games. In *Proceedings of SODA-18*, 2291–2310.

Boehmer, N.; Han, M.; Xu, H.; and Tambe, M. 2024. Escape Sensing Games: Detection-vs-Evasion in Security Applications. In *Proceedings of ECAI-24*, 3260–3267.

Borie, R. B.; Tovey, C. A.; and Koenig, S. 2011. Algorithms and Complexity Results for Graph-Based Pursuit Evasion. *Autonomous Robots*, 31(4): 317–332.

Călinescu, G.; Dumitrescu, A.; and Pach, J. 2008. Reconfigurations in Graphs and Grids. *SIAM Journal on Discrete Mathematics*, 22(1): 124–138.

Čáp, M.; Vokřínek, J.; and Kleiner, A. 2015. Complete Decentralized Method for on-Line Multi-Robot Trajectory Planning in Well-Formed Infrastructures. In *Proceedings of ICAPS-15*, 324–332.

Carmesin, S.; Woller, D.; Parker, D.; Kulich, M.; and Mansouri, M. 2023. The Hamiltonian Cycle and Travelling Salesperson Problems With Traversal-Dependent Edge Deletion. *Journal of Computational Science*, 74: 102156.

Deligkas, A.; Eiben, E.; Ganian, R.; Kanj, I.; and Ramanujan, M. S. 2025. Parameterized Algorithms for Multiagent Pathfinding on Trees. In *Proceedings of AAMAS-25*, 584–592.

Demaine, E. D.; Demaine, M. L.; Fox-Epstein, E.; Hoang, D. A.; Ito, T.; Ono, H.; Otachi, Y.; Uehara, R.; and Yamada, T. 2015. Linear-Time Algorithm for Sliding Tokens on Trees. *Theoretical Computer Science*, 600: 132–142.

Dijkstra, E. W. 1959. A Note on Two Problems in Connexion With Graphs. *Numerische Mathematik*, 1: 269–271.

Dvořák, M.; Knop, D.; Opler, M.; Pokorný, J.; Suchý, O.; and Szilágyi, K. 2025. Pathfinding in Self-Deleting Graphs. In *Proceedings of ISAAC-25*, 28:1–28:15.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N. R.; Wagner, G.; and Surynek, P. 2017. Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *Proceedings of SOCS-17*, 29–37.

Fioravantes, F.; Knop, D.; Křisťan, J. M.; Melissinos, N.; Opler, M.; and Vu, T. A. 2025a. Solving Multiagent Path Finding on Highly Centralized Networks. In *Proceedings of AAAI-25*, 23186–23193.

Fioravantes, F.; Knop, D.; Křišťan, J. M.; Melissinos, N.; and Opler, M. 2024. Exact Algorithms and Lowerbounds for Multiagent Path Finding: Power of Treelike Topology. In *Proceedings of AAAI-24*, 17380–17388.

Fioravantes, F.; Knop, D.; Melissinos, N.; and Opler, M. 2025b. When Agents Break Down in Multiagent Path Finding. *CoRR*, abs/2508.03777.

Forišek, M. 2010. Computational Complexity of Two-Dimensional Platform Games. In *Proceedings of FUN-10*, 214–227.

Fried, D.; Shimony, S. E.; Benbassat, A.; and Wenner, C. 2013. Complexity of Canadian Traveler Problem Variants. *Theoretical Computer Science*, 487: 1–16.

Garey, M. R.; and Johnson, D. S. 1975. Complexity Results for Multiprocessor Scheduling under Resource Constraints. *SIAM Journal on Computing*, 4(4): 397–411.

Gonzalez, T. F. 1985. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38: 293–306.

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

Hearn, R. A.; and Demaine, E. D. 2005. PSPACE-completeness of Sliding-Block Puzzles and Other Problems Through the Nondeterministic Constraint Logic Model of Computation. *Theoretical Computer Science*, 343(1-2): 72–96.

van den Heuvel, J. 2013. The Complexity of Change. In *Surveys in Combinatorics*, volume 409 of *London Mathematical Society Lecture Note Series*, 127–160. Cambridge University Press.

Jakob, M.; Vaněk, O.; Hrstka, O.; and Pěchouček, M. 2012. Agents vs. Pirates: Multi-Agent Simulation and Optimization to Fight Maritime Piracy. In *Proceedings of AAMAS-12*, 37–44.

Kaczmarczyk, A.; Schierreich, Š.; Tanujaya, N. A.; and Xu, H. 2026. Optimal Path Planning in Hostile Environments. *CoRR*, abs/2603.18958.

Kaźmierowski, S. 2025. Equilibria of the Colonel Blotto Games With Costs. In *Proceedings of AAAI-25*, 13969–13976.

Klobas, N.; Mertzios, G. B.; Molter, H.; Niedermeier, R.; and Zschoche, P. 2023. Interference-Free Walks in Time: Temporally Disjoint Paths. *Autonomous Agents and Multi-Agent Systems*, 37(1): 1.

Kornhauser, D.; Miller, G. L.; and Spirakis, P. G. 1984. Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications. In *Proceedings of FOCS-84*, 241–250.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019. Disjoint Splitting for Multi-Agent Path Finding With Conflict-Based Search. In *Proceedings of ICAPS-19*, 279–283.

McCarthy, J. 1998. Partial Formalizations and the Lemmings Game. Technical report, Stanford University, Formal Reasoning Group.

Nebel, B. 2024. The Computational Complexity of Multi-Agent Pathfinding on Directed Graphs. *Artificial Intelligence*, 328: 104063.

Nikolova, E.; Brand, M.; and Karger, D. R. 2006. Optimal Route Planning Under Uncertainty. In *Proceedings of ICAPS-06*, 131–141.

Papadimitriou, C. H.; and Yannakakis, M. 1991. Shortest Paths Without a Map. *Theoretical Computer Science*, 84(1): 127–150.

Parsons, T. D. 1976. Pursuit-Evasion in a Graph. In *Theory and Applications of Graphs*, volume 642 of *Lecture Notes in Mathematics*, 426–441. Berlin, Heidelberg: Springer.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe Interval Path Planning for Dynamic Environments. In *Proceedings of ICRA-11*, 5628–5635.

Ptacek, T. H.; and Newsham, T. N. 1998. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc.

Rottkemper, B.; and Fischer, K. 2013. Decision Making in Humanitarian Logistics - A multi-objective optimization model for relocating relief goods during disaster recovery operations. In *Proceedings of ISCRAM-13*.

Seymour, P. D.; and Thomas, R. 1993. Graph Searching and a Min-Max Theorem for Tree-Width. *Journal of Combinatorial Theory, Series B*, 58(1): 22–33.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 219: 40–66.

Shofer, B.; Shani, G.; and Stern, R. 2023. Multi Agent Path Finding Under Obstacle Uncertainty. In *Proceedings of ICAPS-23*, 402–410.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of SOCS-19*, 151–158.

Surynek, P. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *Proceedings of AAAI-10*, 1261–1263.

Surynek, P. 2022. Problem Compilation for Multi-Agent Path Finding: A Survey. In *Proceedings of IJCAI-22*, 5615–5622.

Tambe, M. 2012. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

Viglietta, G. 2015. Lemmings Is PSPACE-complete. *Theoretical Computer Science*, 586: 120–134.

Wagner, G.; and Choset, H. 2017. Path Planning for Multiple Agents Under Uncertainty. In *Proceedings of ICAPS-17*, 577–585.

Wang, S.; Xu, H.; Zhang, Y.; Lin, J.; Lu, C.; Wang, X.; and Li, W. 2025. Where Paths Collide: A Comprehensive Survey of Classic and Learning-Based Multi-Agent Pathfinding. *CoRR*, abs/2505.19219.

van Wassenhove, L. N. 2006. Humanitarian Aid Logistics: Supply Chain Management in High Gear. *Journal of the Operational Research Society*, 57(5): 475–489.

Wu, F.; Ramchurn, S. D.; and Chen, X. 2016. Coordinating Human-UAV Teams in Disaster Response. In *Proceedings of IJCAI-16*, 524–530.

Wu, H.; Cheng, J.; Ke, Y.; Huang, S.; Huang, Y.; and Wu, H. 2016. Efficient Algorithms for Temporal Path Computation. *IEEE Transactions on Knowledge and Data Engineering*, 28(11): 2927–2942.

Yamanaka, K.; Horiyama, T.; Keil, J. M.; Kirkpatrick, D.; Otachi, Y.; Saitoh, T.; Uehara, R.; and Uno, Y. 2018. Swapping Colored Tokens on Graphs. *Theoretical Computer Science*, 729: 1–10.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of AAAI-13*, 1443–1449.