Anonymous Multi-Agent Path Finding with Individual Deadlines

Gilad Fine
Bar Ilan University, Israel
giladfine1@gmail.com

Dor Atzmon
Royal Holloway, University of London, UK
dor.atzmon@rhul.ac.uk

Noa Agmon
Bar Ilan University, Israel
agmon@cs.biu.ac.il

ABSTRACT

Anonymous Multi-Agent Path Finding (AMAPF) is the problem of planning conflict-free paths to a set of target locations for a group of agents, where each agent is not associated with a specific target. This paper studies AMAPF with Individual Deadlines (AMAPFwID), where each target must be reached before a specific deadline, and the agent stays there to perform some long-term mission, for instance securing an asset, responding to an emergency, or performing a maintenance job. We examine three types of behavior of agents when reaching a target: (a) disappear on target, (b) stay on target, and (c) move after the deadline. The latter is only possible if the agent is replaced by another agent. We refer to this replacement as *hot swapping*. We propose a solution to AMAPFwID with each type of such behavior, based on a reduction to Network Flow. We test all solutions experimentally and show cases where hot swapping is beneficial. Finally, we also provide a solution to the case where agents disappear on targets that maximizes the number of targets reached and discuss other aspects of the problem.

KEYWORDS

MAPF, Network Flow, Multiagent Systems, Path Planning

ACM Reference Format:
Gilad Fine, Dor Atzmon, and Noa Agmon. 2023. Anonymous Multi-Agent Path Finding with Individual Deadlines. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

Recent years have brought considerable attention to the use of robots in missions that vary from target protection to package delivery or warehouse management. While the use of robots has unique advantages—among those human-safety and cost efficiency—it also raises special challenges, first and foremost the need to prevent robots from colliding with their peers when traveling in the environment. This constraint is encapsulated in the problem of Multi-Agent Path-Finding (MAPF) [24], which focuses on finding collision-free paths for multiple agents (robots) that travel from given initial positions to target locations.

Motivated by the use of robots in *ongoing missions*, such as target protection or event handling, in which an agent or a robot must reach a target location and maintain its position there for performing some task, we introduce a new version of the MAPF problem, the **Anonymous Multi-Agent Path-Finding with Individual Deadlines** (AMAPFwID). In this problem, a group of agents needs to reach (acquire) permutation-invariant target locations in the environment. Each target has an individual deadline, which determines the maximal timestep where this target must be reached by some agent in order for the target to be considered *acquired*. After this deadline, *at all times*, there must be some agent in the target's location.

The AMAPFwID is a special case of the Anonymous MAPF (AMAPF), in which agents are not assigned a specific target, and finding collision-free paths minimizing the sum of traveled distances (fuel), the maximal path cost (makespan), or the sum of all paths' costs (SOC) can be solved in polynomial time [30]. When targets are correlated with individual deadlines, the existing solutions become irrelevant, and thus in this paper we examine in depth the AMAPFwID problem and suggest algorithms for solving it.

We examine three different cases of the AMAPFwID problem, that differ in the behavior of an agent once reaching the target location: either (a) the agent disappears with the target (similar to entering a building), thus not disturbing the movement of other agents, (b) the agent remains present and blocks other agents (as an obstacle), or (c) the agent stays at the target, though allowing for other agents to take over the target while it continues to travel to another target (similar to a relay race). We refer to the novel action of relaying the acquisition on a target to another agent as *hot swapping*. Real-life scenarios for hot swapping include keeping a target secured while switching security guards, and providing a continuous service (e.g., medical attention or technical maintenance) while replacing the service provider staff (e.g., physicians or technicians). Most work on MAPF assumes the second behavior, that is, the agents stay at target, though some refer to the possibility that agents disappear at target, as seen recently in [16] for non-anonymous MAPF. We therefore introduce the hot-swapping behavior which is applicable only in anonymous MAPF, and—as shown herein—is extremely useful when handling such ongoing missions.

Our main focus in this paper is a variant of the problem that aims to find a *feasible* solution, in which we are interested to know whether all targets can be acquired by the agents given their initial positions. We show that this can be solved in polynomial time in all three cases (a-c) and describe reductions to Network Flow for each of the cases. We prove that our solutions also have the minimal traveled distances (fuel) among all possible solutions. Additionally, we provide a solution for case (a)—where agents disappear at their target locations—that maximizes the number of targets acquired by the agents and discuss other aspects of the problem.

We have fully implemented all algorithms[1], and examined their performance in benchmark environments. We show that the ability to relay acquisition on targets by performing hot swapping significantly extends the range of problem-solving in various settings.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

[1] Our implementation is publicly available at https://github.com/GiladFine/MAPF-with-Adversarial-Constraints.

2 RELATED WORK

AMAPFwID is rooted in the domain of **Multi-Agent Path-Finding (MAPF)** problems. In these problems, the main goal is to navigate a team of physical agents to a set of target locations while avoiding collisions [20, 23, 24]. The problem has two main variations relevant to our work: **Non-Anonymous** and **Anonymous**, where the difference is whether the agents are initially assigned with a specific goal location, or not (respectively). The classical non-anonymous MAPF is *NP*-hard for different known objective functions [8, 25, 31]. Nevertheless, many efficient optimal MAPF algorithms were proposed [2, 4, 7, 14, 15, 21, 22, 26, 28]. In the anonymous case, reducing the problem to a Network Flow problem and using polynomial time algorithms such as the Ford-Fulkerson algorithm [5, 6] for finding max-flow, was shown to solve the anonymous MAPF problem in polynomial time [30].

Ma et al. [18] examined the problem of MAPF with deadlines, in which the goal is to maximize the number of agents arriving at their target locations before a given deadline T in the non-anonymous case. They prove the problem is *NP*-hard, introduce two optimal solvers, and show how they scale in different scenarios. Wang and Chen [29] considered the MAPF problem with deadlines as a soft constraint (referred to as *due time*), where the goal is to minimize the maximal tardiness, sum of tardiness or total penalty for the tardiness with respect to a set of individual deadlines D. They show that in the anonymous case the problem is solvable in polynomial time, a setting that is similar to the AMAPFwID discussed in this work when agents stay on targets once those are reached. However, in our work, we examine in depth the problem of satisfying the deadlines (while also minimizing traveled distances), and we specifically focus on different agent behaviors once reaching the targets, and how this influences the system's performance.

The problem of Adversarial Cooperative Path-Finding (*ACPF*) was presented by Ivanova et al. [10, 11], extending the MAPF problem to consider a game between two groups of agents, one is controlled by the player and the other by an adversary, where the goal is to reach first a set of target locations. The game is sequential, all teams take turns in moving every agent in their team, and the team that reaches more target locations wins the game. Solving this as a non-anonymous MAPF problem is clearly computationally hard. Thus, solutions to the problem are examined empirically. A further version of this problem is the Area Protection Problem (*APP*) [12], attempting to find a defending strategy for a team of agents tasked with protecting an area populated with high-profile assets points (targets), known to be targeted by a rival team of agents. In *APP*, the adversarial agents' strategy is unknown, yielding a PSPACE-hard complexity for solving it. The AMAPFwID can be considered as an instance of *APP* where the adversarial strategy is known, thus simplifying the time complexity for solving it.

Another research area closely related to the AMAPFwID problem is the *Task Assignment* problems. In this type of problems, we need to assign limited resources to tasks, in order to optimize certain criteria, often in a setting of robots navigation, e.g., assigning delivery jobs to agents [9], or even more specifically, assigning targets to agents [17], while attempting to minimize the makespan, i.e., the overall time that takes these agents to complete their task. A common solution to the task assignment problem is the *Hungarian Method* [13], which operates in polynomial time. However, the Hungarian method neglects to consider the possible collisions resulting from the robots operating in the same physical space, making it impractical for AMAPFwID. Other studies explored the combination of task allocation and path planning in different environments [1, 3, 27]. However, none of them support individual deadlines for the targets. Future work may adjust other algorithms for this purpose.

3 ANONYMOUS MULTI-AGENT PATH FINDING WITH INDIVIDUAL DEADLINES

The Anonymous MAPF with Individual Deadlines (AMAPFwID) is defined as a 5-tuple (Gr, A, G, D, L) where Gr is a finite undirected graph $Gr = \{V, E\}$ describing the environment, $A = \{a_0, \ldots, a_{n-1}\}$ is a team of n agents, $G = \{g_0, g_1, \ldots, g_{n-1}\}$ is a set of n target goal locations (vertices) on the graph, and $D = \{d_{g_0}, d_{g_1}, \ldots, d_{g_{n-1}}\}$ is a set of deadlines (in timesteps) for each goal in G. The initial location of an agent $a_i \in A$ is denoted by l_i. We are given $L = \{l_0, \ldots, l_{n-1}\}$, the set of initial locations of all agents in A. We denote by $N(v)$ the set of neighboring locations of location $v \in V$ ($\forall v' \in N(v) : (v', v) \in E$). Two locations $v_1, v_2 \in V$ are adjacent locations if $v_1 \in N(v_2)$ (or visa versa).

An **Action** is the operation an agent performs between two consecutive timesteps t and $t+1$. There are two types of actions: (1) a *move* action, where an agent positioned at location v at timestep t moves to an adjacent location and is then positioned at some location $v' \in N(v)$ at timestep $t+1$ and (2) a *wait* action, where an agent is positioned at location v at timesteps t and $t+1$.

A **Path** for an agent $a_i \in A$ is defined as a sequence of locations such that the agent either performs *wait* or *move* actions between any pair of consecutive timesteps -
$P_{a_i} = \{[p_0, p_1, p_2, \ldots]\}$, where $\forall p_j, p_{j+1} \in P_{a_i} : \exists p_{j+1} \in N(p_j)$ or $p_j = p_{j+1} \in V$. Each $p_j = P_{a_i}[j]$ denotes the location of the agent at timestep j.

A **Conflict/Collision** between two paths P_{a_x} and P_{a_y} occurs if the agents a_x and a_y occupy the same location at the same timestep t ($\exists t : P_{a_x}[t] = P_{a_y}[t]$) or if the agents swap locations between two consecutive timesteps t and $t+1$ ($\exists t : P_{a_x}[t] = P_{a_y}[t+1] \wedge P_{a_x}[t+1] = P_{a_y}[t]$).

A **Plan** for the agents A is defined as a dictionary -
$S_A = \{(a_i, P_{a_i}) | \forall a_i \in A \text{ and } P_{a_i}[0] = l_i\}$ - each agent is mapped to a path that starts in its initial location. A plan S_A is *conflict-free* if any two paths in S_A do not conflict.

A target $g_i \in G$ is said to be **acquired** by plan S_A if $\forall t' \geq d_{g_i} \exists (a_j, P_{a_j}) \in S_A$ where $P_{a_j}[t'] = g_i$. This means that for each timestep larger than or equal to the deadline, we have an agent in our plan that is located at the target location at this timestep.

A **Solution** for AMAPFwID is a conflict-free plan S_A for agents A such that all targets are acquired. We define three different cases for agent behavior once reaching a target:

(a) **Agents disappear on targets (DOT).** In this case, the agent disappears immediately when the target is acquired at its deadline.
(b) **Agents stay on targets (SOT).** In this case, the agent must stay at the target location when the target is acquired at its deadline and after the deadline.

(c) **Agents can move after deadline (HOT).** In this case, agents are free to move away from a target location at any time (before and after the deadline). This creates a unique situation, where an agent acquiring a target can move even after the deadline, as long as another agent is moving to the target location, keeping the target acquired at all times. We refer to this case of relaying the acquisition on a target to another agent as **hot swapping**. Assume that an agent a_x currently acquires some target $g \in G$, and agent a_y is located on some location $u \in N(g)$. Then, if a_x performs hot swapping with a_y, both agents move simultaneously, a_x to some location $v \in N(g)$ $(v \neq u)$ and a_y moves into g, leaving g acquired at all timesteps.

Note that hot swapping is only possible when the problem is anonymous, as it requires an agent to switch targets.

There are different ways in the literature to define the cost function that evaluates a given solution [24]. One possible cost function, also known as *fuel* or *traveled distances*, counts the number of *move* actions performed in the given solution. In this cost function, there is no cost for *wait* actions. Later in this paper, we show that our solutions for all cases defined above also minimize the traveled distances cost function of the system.

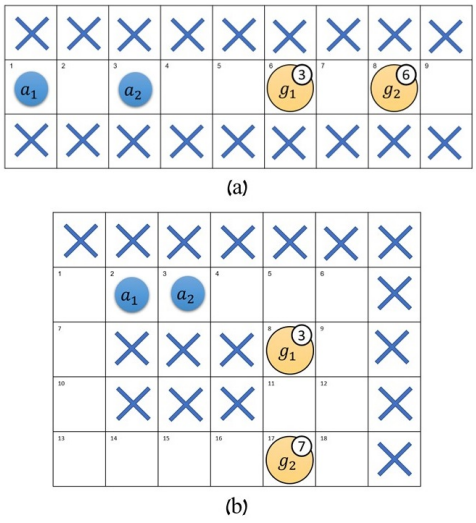


Figure 1: Examples for hot-swapping. Targets g_1 and g_2 are represented as yellow circles with their associated deadlines, agents a_1 and a_2 appear in their initial locations

We show herein that the behaviors described above may have an important effect on the satisfiability of a problem instance, as well as the distances traveled by the agents. Consider the example in Figure 1(a). In SOT and DOT, the agents cannot reach both targets on time, as agent a_1, initially located on square 1 is unable to reach any of the targets on time. However, if using HOT, the problem is satisfied. Specifically, agent a_2 (initially located on square 3) reaches g_1 before the deadline, and agents a_1 and a_2 perform hot swapping, allowing a_2 to continue acquiring g_2 by its deadline while a_1 stays in square 6 to acquire g_1. Therefore, their paths are $P_{a_1} = [1, 2, 3, 4, 5, 6]$ for a_1, and $P_{a_2} = [3, 4, 5, 6, 6, 7, 8]$ for a_2.

Algorithm 1: AMAPFwID-DOT to Network Flow Reduction

Input: AMAPFwID problem instance $(Gr(V, E), A, G, D, L)$
Output: Network $N = (\overrightarrow{Gr}(V', E'), u, c, source, sink)$

Reduction$((Gr(V, E), A, G, D, L))$
  $T \leftarrow max_{d \in D}(d)$
  $V', E' \leftarrow \{source, sink\}, \emptyset$
  for $v \in V$ **do**
    for $t \in \{0, \ldots, T\}$ **do**
      $V' \leftarrow V' \cup \{v(t), v'(t)\}$
      $e \leftarrow (v(t), v'(t)), c(e) = 0$
      $E' \leftarrow E' \cup \{e\}$ // *Location-occupancy edges*
    for $t \in \{0, \ldots, T-1\}$ **do**
      $e_w \leftarrow (v(t)', v(t+1)), c(e) = 0$
      $E' \leftarrow E' \cup \{e_w\}$ // *Wait actions*
    for $v_n \in N(v)$ **do**
      $V_G, E_G \leftarrow$ Gadget$(v'(t), v'_n(t), v(t+1), v'_n(t+1))$
      $V' \leftarrow V' \cup V_G$
      $E' \leftarrow E' \cup E_G$ // *Move actions*
  for $l_i \in L$ **do**
    $e \leftarrow (source, l'_i(0)), c(e) = 0$
    $E' \leftarrow E' \cup \{e\}$
  for $g_i \in G$ **do**
    $e \leftarrow (g'_i(d_{g_i}), sink), c(e) = 0$
    $E' \leftarrow E' \cup \{e\}$
  for $e \in E'$ **do**
    $u(e) = 1$
  $N \leftarrow (\overrightarrow{Gr}(V', E'), u, c, source, sink)$
  return N

Gadget$(v'_x(t), v'_y(t), v_x(t+1), v_y(t+1))$
  $V_G = \{v_1, v_2\}$
  $e_1 \leftarrow (v'_x(t), v_1), e_2 \leftarrow (v'_y(t), v_1), c(e_1) = 0, c(e_2) = 0$
  $e_3 \leftarrow (v_2, v_x(t+1)), e_4 \leftarrow (v_2, v_y(t+1)), c(e_3) = 0, c(e_4) = 0$
  $e_n \leftarrow (v_1, v_2), c(e_n) = 1$
  $E_G = \{e_1, e_2, e_3, e_4, e_5\}$
  return V_G, E_G

Figure 1(b) describes a case in which SOT is not satisfied, though DOT and HOT are. In DOT, $P_{a_1} = [2, 3, 4, 5, 8, 11, 17]$ and $P_{a_2} = [3, 4, 5, 8]$, where a_2 disappears in g_1, allowing for a_1 to travel to acquire g_2. In HOT, $P_{a_1} = [2, 3, 4, 5, 8]$ and $P_{a_2} = [3, 4, 5, 8, 11, 17]$, meeting both deadlines by performing hot swapping in g_1. Note that if setting d_{g_2} to 6, the problem is satisfied according to HOT but not according to DOT. If d_{g_2} is set to 8, then the problem is satisfied according to all three options, though SOT yields paths that are less efficient in terms of traveled distances.

4 AMAPFWID TO NETWORK FLOW REDUCTION

Here, we describe the reduction from AMAPFwID to Network Flow, based on the reduction from AMAPF suggested by Yu and LaValle [30]. The idea behind the reduction is to create a network that consists of a time-expanded version of the original MAPF graph, representing the movements of agents through time. To create a network such that it represents a reduction from AMAPFwID, instead of a reduction from AMAPF, two important definitions must be considered: (1) the deadline of each target, and (2) the behavior of the agent when arriving at a target location. We fully describe AMAPFwID-DOT and then present the modifications required for AMAPFwID-SOT and AMAPFwID-HOT.

4.1 AMAPFwID-DOT (Disappear on Targets)

The reduction from AMAPFwID-DOT to Network Flow is presented in Algorithm 1. Figure 2 shows an AMAPFwID problem instance with 4 locations and two agents, such that their initial locations are $L = \{u, v\}$, the targets are $G = \{x, z\}$, and their deadlines are $d_x = 2$ and $d_z = 1$. Figure 3 shows the network $N = (\overrightarrow{Gr}, u, c, source, sink)$ constructed for the reduction, as explained next. A network N consists of a directed graph $\overrightarrow{Gr} = (V', E')$, a capacity function $u : E' \to \mathbb{Z}^+$, a cost function $c : E' \to \mathbb{Z}^+$, and $source \in V'$ and $sink \in V'$ vertices.

First, we set $T \leftarrow max_{d \in D}(d)$ (line 2), where T represents the number of timesteps considered by the network, which in our case is the latest deadline (the agents cannot move after this deadline). We then initialize the sets of vertices V' and edges E' for our network, where we add the *source* and *sink* vertices to V' (line 3).

Now, we perform a cycle for every location $v \in V$ to create the time-expanded graph for the network (lines 4-15). For each vertex v, at each timestep $t \in \{0, \ldots, T\}$ we create two vertices $v(t)$ and $v'(t)$ and a directed edge $(v(t), v'(t))$ (lines 5-8). This edge, also illustrated in Figure 3, is created to prevent the case where more than one agent occupies a vertex at the same timestep. We call this edge a *location-occupancy* edge.

Then, to model the movement of the agents, we create two types of edges for *wait* and *move* actions. To allow *wait* actions, we create a directed edge $(v'(t), v(t+1))$ for each location $v \in V$ and for each timestep $0 \leq t < T$ (lines 9-11). To allow *move* actions between two adjacent locations v and u, we create a five-edge gadget (lines 12-15), as illustrated in Figure 3. This gadget is created to prevent agents from conflicting while swapping adjacent locations. As proved by Yu and LaValle [30], any solution to AMAPF that only contains such swapping conflicts can be reconstructed such that no such conflicts exist. Here, we describe the version of the network that results in a conflict-free solution without the need to rebuild the returned plan.

Finally, we create a directed edge $(source, l_i(0))$ for each start location $l_i \in L$ at timestep 0, and a directed edge $(g'_i(d_{g_i}), sink)$ for each target location g_i at its deadline d_{g_i} (lines 16-21).

To prevent conflicts, all edges receive a capacity of 1 (lines 22-23). To guarantee optimality (fuel), only *move* actions have a cost of 1 (line 30) while all other edges have a cost of 0.

This reduction results in a minimum-cost flow problem (MCFP), solvable in polynomial time, for instance, using the *Ford–Folkerson* algorithm, where a feasible flow exists if and only if the original graph has a valid AMAPFwID-DOT solution. Moreover, the returned flow can be easily converted to a set of paths for the agents that represent a valid AMAPFwID-DOT solution. This can be done by following the vertices in which the flow passes through while omitting the *source* and *sink* vertices, as well as any $v'(t)$ vertex (with an apostrophe). In the proofs below, when referring to a flow, we mean the set of paths the flow represents, which omits the irrelevant vertices.

Theorem 4.1. *Given a* AMAPFwID-DOT *problem instance, a solution exists iff a feasible flow exists in the network created by Algorithm 1. Moreover, if a solution exists, the minimum-cost flow represents a solution minimizing the traveled distances (fuel).*

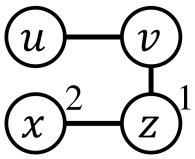


Figure 2: AMAPFwID problem instance

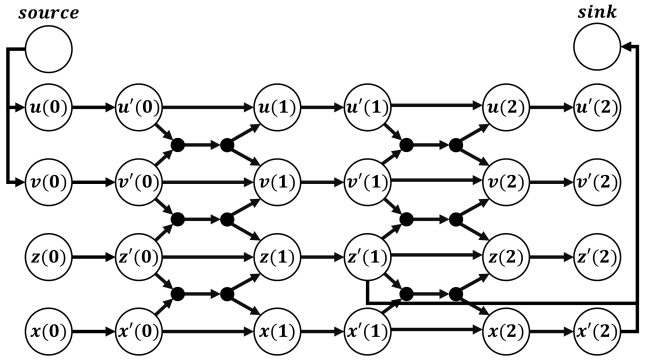


Figure 3: AMAPFwID-DOT to Network-Flow reduction example

Proof outline. The network represents an exact time-expanded graph of the received graph; all *wait* and *move* actions for the agents are represented in the network. Therefore, both a plan for the agents and a feasible flow in the network can have the exact same transitions. Both (a plan and a flow) start from the initial locations of the agents and finish at the target locations at their deadlines. Due to the capacity of the edges (= 1) and their construction, the flow also guarantees that no two agents will be able to occupy the same location at the same timestep or traverse an edge in opposite directions between consecutive timesteps. After deadlines, the flow ends and other agents may pass through the corresponding target location. Lastly, in a plan and in a flow, agents cannot perform any other actions after the latest deadline.

The optimality of the returned solution follows as a direct result of setting a cost of 1 for any edge that represents a *move* action and a cost of 0 for any other edge. By defining the problem as a minimum-cost flow problem, the returned flow must have the lowest cost among all feasible flows. □

4.2 AMAPFwID-SOT (Stay on Targets)

In AMAPFwID-SOT, the agents must stay at the target locations after their deadline and not move. Therefore, here, other agents cannot pass through any target location after its deadline. To model this behavior, we make the following modification in the network created in Algorithm 1.

- Remove all five-edge gadgets that represent *move* actions from/to target locations after their deadline.

Figure 4 shows the network presented in Figure 3 after this modification. We can see that after the deadline $d_z = 1$ of target location z, agents can no longer enter this location.

AMAPFwID-SOT is a more restrictive definition than AMAPFwID-DOT. Therefore, any solution to SOT is also a valid solution to DOT.

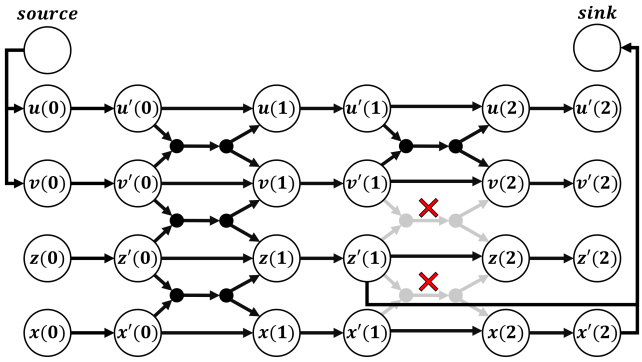


Figure 4: AMAPFwID-SOT to Network-Flow reduction example

The problem instance presented in Figure 2 is not solvable for both DOT and SOT. However, if the deadline $d_x = 2$ of target location x was $d_x = 3$ instead, it would have been solvable in DOT but not in SOT. This demonstrates again that some instances can be solvable in AMAPFwID-DOT but not in AMAPFwID-SOT. Proving that the network created in the reduction for SOT solves AMAPFwID-SOT is identical to Theorem 4.1, except the fact that here agents cannot move from/to target locations after their deadline, which is modeled by the above modification.

4.3 AMAPFwID-HOT (Hot Swapping)

Hot swapping poses complex challenges to the network flow reduction. In this case, we must allow agents to leave the target locations after the deadlines, but only if another agent replaces them there (or, in other words, hot swapping can be performed only if the target remains occupied at all timesteps after the deadline).

Assume an optimal (fuel) solution S_A to AMAPFwID-HOT. We can order the agents according to the time each agent arrived at a target location in S_A and stayed there. The agents may perform hot swapping only with agents that have not yet reached a target location and stayed there, e.g., there cannot be hot swapping at the latest acquired target. This gives us the following observation.

Observation 1. *Given an optimal solution S_A to* AMAPFwID-HOT*, the maximum number of hot swapping performed in S_A is $N_{HS} = \sum_{i=1}^{n-1}(i)$ (n is the number of targets).*

For example, if we have two targets, there may be only up to $N_{HS} = 1$ hot-swapping operation; if we have three targets, there may only be up to $N_{HS} = 3$ hot-swapping operations: two times on one target, one time on another target, and no hot swapping on the latest acquired target.

Let $SD = \sum_{g_i \in G} d_{g_i}$, i.e., the sum of all deadlines, and let $X = SD + N_{HS} + 1$. The modifications needed for AMAPFwID-HOT, in the network defined in Algorithm 1, are as follows.

- Remove every $(g'_i(d_{g_i}), sink)$ edge of every target g_i.
- Add an edge $e = (g'_i(T), sink)$ for every target location g_i at timestep T, with $c(e) = 0$ and $u(e) = 1$.
- Consider each location-occupancy edge $e = (v(t), v'(t))$, defined in line 7. If $v \in G$ (v is a target location) and $t \geq d_v$, set $c(e) = 0$. Otherwise, set $c(e) = X$.

In the problem instance from Figure 2, the sum of all deadlines is $SD = 3$ and the maximum number of hot swapping is $N_{HS} = 1$. Thus, $X = 5$. Figure 5 presents the network from Algorithm 1 after performing the above modifications. The costs of edges with a non-zero cost are presented in the figure.

The modified network is now sent to be solved by a minimum-cost flow algorithm. Let f be the returned flow (if exists).

- If there is no feasible flow, return *no solution*.
- If the cost of f is $\geq X \cdot (SD + 1)$, return *no solution*.
- Otherwise, return f.

By sending the network from Figure 5 to a minimum-cost flow algorithm, the returned flow consists of ($source$, $v(0)$, $v'(0)$, $z(1)$, $z'(1)$, $x(2)$, $x'(2)$, $sink$) and ($source$, $u(0)$, $u'(0)$, $v(1)$, $v'(1)$, $z(2)$, $z'(2)$, $sink$). The cost of this flow is 19, which is lower than $X \cdot (SD + 1) = 20$. Therefore, we return this flow as a solution to our problem instance. This flow represents the solution of (v, z, x) and (u, v, z), where the agents perform hot swapping at location z.

Theorem 4.2. *Given a* AMAPFwID-HOT *problem instance, a solution exists iff the reduction (after the above modifications) returns a feasible flow. Moreover, if a solution exists, the minimum-cost flow represents a solution minimizing the traveled distances (fuel).*

Proof outline. Assume a solution S_A exists for the given problem instance. Now, let us observe the flow f that represents the same movement of the agents defined by S_A. In this flow, all location-occupancy edges of targets after their deadlines are traversed. Otherwise, it means that one of the targets was not acquired at all timesteps. The maximum possible cost for such a flow is $(X + 1) \cdot SD + N_{HS}$. This cost is received when the agents only performed *move* actions (each costs $X + 1$, considering also location-occupancy edges) at any possible timestep before the deadlines (SD timesteps) and performed the maximum number of hot swapping N_{HS} (each increases the cost by 1). An example of such a maximum-cost flow is presented above in our example. We get that $(X+1) \cdot SD + N_{HS} = X \cdot SD + SD + N_{HS} < X \cdot SD + X = X \cdot (SD + 1)$. That it, this cost is lower than $X \cdot (SD + 1)$, and thus this flow will be returned by the reduction.

Assume that there is no solution to the given problem instance, then either no feasible flow will be found in our network, or a flow with a cost of $\geq X \cdot (SD + 1)$, as at least one location-occupancy edge with a cost of X must be traversed, instead of a cost of 0 at a target location after the deadline. In either case, the reduction returns *no solution*.

Optimality of the returned solution follows again as a direct result of setting a cost of 1 for any edge that represents a *move* action and a cost of 0 for any edge that represents a *wait* action. By defining the problem as a minimum-cost flow problem, the returned flow must have the lowest cost among all feasible flows.

□

5 HOT-SWAPPING VARIATIONS

In general, we have two main motivations for performing hot swapping: clearing roadblocks (for example, an agent securing a target on a bottleneck and thus blocking the passage to another target) and saving time (as the swapping saves a single timestep for the proceeding agent). In Figure 1(a), we can see an example of both

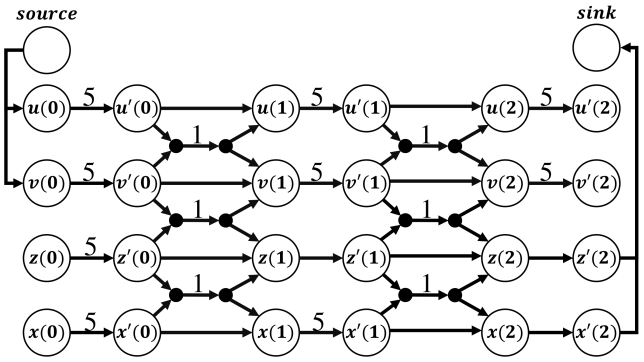


Figure 5: AMAPFwID-HOT to Network-Flow reduction example

motivations. In some real-life cases, performing hot swapping may consume time. For instance, when the agents are required to transfer information in order to perform such a switch. This time, denoted by T_{HS}, can be considered as the time it takes the agent currently acquiring the target to pass the responsibility to the incoming agent.

Let us redefine hot swapping. Assume that an agent a_x currently acquires some target $g \in G$, agent a_y is positioned on some location u adjacent to g. To perform hot swapping, let us assume that agent a_x moves to location v, which is also adjacent to g ($v \neq u$) and agent a_y moves to the target location g. When a_x performs hot swapping with a_y, the agents are positioned at each timestep according to the following definition (starting at timestep t).

(1) Timestep t: a_x is at target g and a_y is at location u.
(2) Timesteps $t+1$ to $t+T_{HS}$: both a_x and a_y are at target g.
(3) Timestep $t+T_{HS}+1$: a_y is at target g and a_x is at location v.

As we can see, this movement is guaranteed to leave g acquired at all times, as needed. In this movement, we allow agents to conflict, *but only on the target location and only after the deadline*, as the purpose of hot swapping is to keep a target acquired. If an agent is blocking the way on a non-target location (or on a target before the deadline), we have no target to keep acquired. It is easy to see that for $T_{HS} = 0$ (denoted as HS_0), we get the standard **AMAPFwID with hot swapping**. Also, for $T_{HS} = 1$ (HS_1), we get the following correlation:

Theorem 5.1. *In* AMAPFwID *with hot swapping, if* $T_{HS} = 1$ *(*HS_1*), then the problem is equivalent to* AMAPFwID *when Agents Disappear on Targets.*

Proof outline. Let us consider a generic hot swapping case: Agent $a_x \in A$ is located at $g \in G$ at timestep t ($t \geq d_g$). Recall that $N(v')$ is the set of neighboring locations of location v'. Agent $a_y \in A$ is located at $u \in V$ at timestep t ($u \in N(g)$). We also have a non-occupied location $v \in N(g)$ ($v \neq u$). In the following paths, we consider only the locations of the agents between timesteps t and $t+2$. The paths for the agents for HS_1 are $P_{a_x} = [\ldots, g, g, v, \ldots]$, $P_{a_y} = [\ldots, u, g, g, \ldots]$. The paths for the Disappearing Targets variation (DOT) are $P_{a_x} = [\ldots, g]$, $P_{a_y} = [\ldots, u, g, v, \ldots]$. We need to show that in every location u, g, or v at each time from t to $t+2$ we have the same status in both problems, in terms of the occupancy of the locations. For u, at timestep t, we have a_y at location u for both problems. At timesteps $t+1$ and $t+2$, we have no agent at location u in both problems. For v, at timesteps t and $t+1$, we have no agent at location v in both problems, and at timestep $t+2$, we have a_x at location v for HS_1, and a_y at location v for DOT, which is the same as the agents are anonymous. For g, in both problems, we have target g acquired from t to $t+2$. At timestep t, both problems have a_x at target g. At timesteps $t+1$ and $t+2$, DOT does not have any agent in target g (HS_1 has a_y there), but it is equivalent, as a_y being at target g in HS_1 will not interfere with future agents moving through target g as these agents can perform hot swapping with a_y again, as we described. □

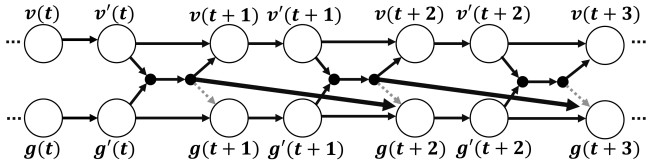


Figure 6: Modified Network for HS_1

For $T_{HS} > 1$ ($HS_{>1}$), we get a new variant of the problem, for which we describe a new algorithm that solves it optimally. For this variant, we consider the algorithm for solving AMAPFwID-HOT (that is, with HS_0), and change it to solve $HS_{>1}$ by applying the following modification.

- For each target $g \in G$, modify each gadget that represents a *move* action from any location $v \in N(g)$ after the deadline d_g, so that the incoming edge to target g is moved T_{HS} timesteps ahead, to model the time delay of hot swapping. This moved edge is given a cost of $T_{HS} \cdot X$, where X is the edge cost defined in Section 4. This cost guarantees that if not all targets are acquired at all timesteps after the deadlines, no solution will be returned.

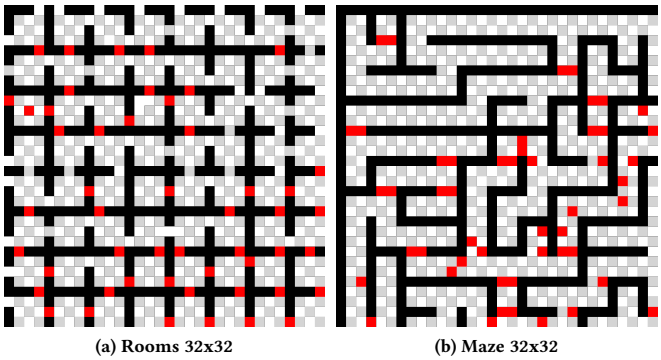
Figure 6 presents the modification for HS_1 ($t \geq d_g$) - dotted grey arrows represent the removed edges, and the bold solid arrows are the new edges.

Theorem 5.2. *The modification to the network described in* AMAPFwID-HOT *correctly models the behavior of* HS_x*, for any* $x > 1$.

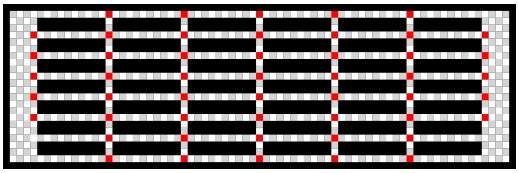
Proof outline. Consider the generic hot swapping (HS_0), where we start at timestep t ($t \geq d_g$), and have 1 unit of flow at location u, and 1 unit of flow at g. Location v is the hot swapping destination location ($g \in G$ and $u, v \in N(g)$, $u \neq v$). At timestep t, we start with 1 unit of flow at u and 1 unit of flow in g, representing agents being in u and g locations. For each timestep from $t+1$ to $t+T_{HS}$, according to the definition of our new construction, we still have 1 unit of flow in g (this is enforced by a cost of 0), and **no flow** in u or in v, which is modeling the desired behavior, as g is still acquired and u and v have no agents in them. At timestep $t+T_{HS}+1$, the flow from u finally reaches g, so the swap can finally take place, causing the flow that was in g to move to v, thus representing agents being in g and v, respectively. □

6 EMPIRICAL EVALUATION

We have fully implemented the algorithms described for the 4 settings: DOT, SOT, HOT-0, HOT-2 for the purpose of evaluating the effect of the agents' behavior when reaching the target on the satisfiability

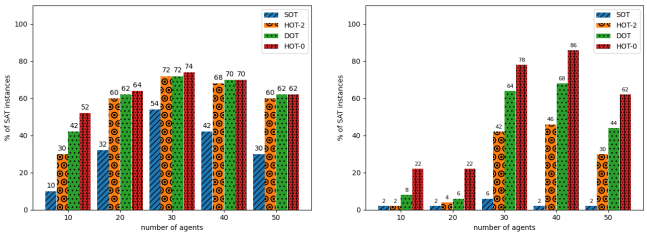


(a) Rooms 32x32 (b) Maze 32x32

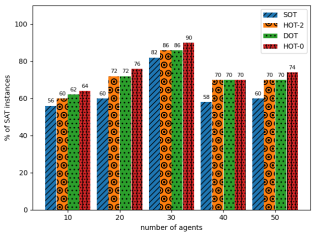


(c) Warehouse 75x24

Figure 7: Environments with goal locations (red squares), used for empirical evaluations

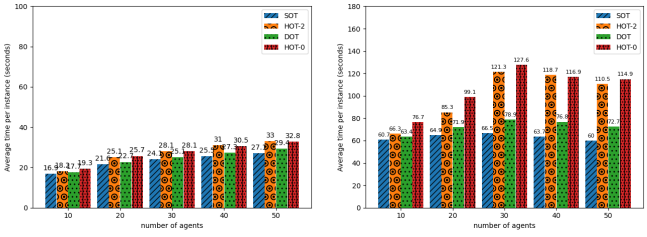


(a) Room-32 (b) Maze-32

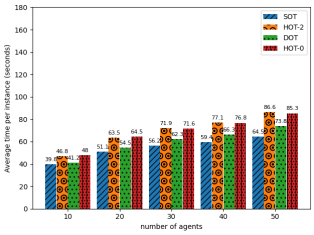


(c) Warehouse

Figure 8: Percentage of satisfied instances for each problem setting results



(a) Room-32 (b) Maze-32



(c) Warehouse

Figure 9: Average instance runtime for each problem setting

of the AMAPFwID problem. In order to do so, we have examined three standard benchmark[2] maps - *rooms*, *maze* and *warehouse* (Figures 7(a) , 7(b), and 7(c) respectively), while placing the targets at interesting locations on the map (mainly bottlenecks, noted by a red square at Figure 7), and placing an equal number of agents at random locations on the map. We have assigned a deadline to each target at random, choosing a value between the minimal distance from any agent to the target, to the maximal distance (other deadlines are either easily or impossibly acquired by any agent, making this instance not interesting). Deadlines are chosen uniformly, though we do factor in relevant information on the problem (mainly the number of goals and the map difficulty). In order to further trim non-interesting instances of the problem, we use the Hungarian Method to check that our problem has a satisfying solution for all targets when neglecting the path-finding element of non-colliding paths. If not, we have repeated the procedure again until resulting in a valid instance.

We have examined instances of the problem for $n = 10, 20, 30, 40, 50$ number of agents and targets. For each number of agents, we have created 50 valid instances, differing in the initial locations of the agents and target deadlines. For each instance, we executed the 4 reductions: one for agents disappearing on targets (DOT), one in which the agents stay at target (SOT), one for hot swapping with 2 timesteps delay (HOT-2), and one for standard hot swapping (HOT-0).

In Figure 8, we present the percentage of satisfied instances (y-axis), for each behavior, for the rooms, maze, and warehouse maps, respectively. We can see that for all maps, for every number of agents (x-axis), we see a growth in the percentage of satisfiability when using a better problem setting, as expected. The gaps between the different settings suggest the existence and likelihood of a problem instance to 'benefit' from hot swapping. These results demonstrate both the major impact that the setting of the problem has on the solutions and the overall usefulness of the hot swapping, which was able to return optimal solutions for many instances that were not solvable before, even for the case where agents disappeared on targets. We have found that the usefulness of hot swapping is better in relation to other problem settings when

[2]https://movingai.com/benchmarks/mapf/index.html

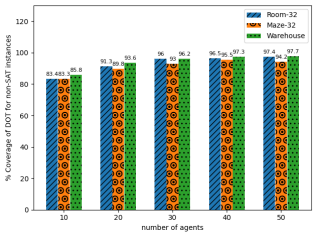


Figure 10: Percentage of goals covered in non-satisfiable instances of DOT per number of agents

the number of agents is relatively small (<10 on *rooms* and <20 on *maze*, for example).

In Figure 9, we show the average running time (seconds) of each method. These results show that the average runtimes grow linearly with the number of agents, for both maps, and for all methods. This is of course expected, due to the Polynomial nature of Network-Flow algorithms. We can also see that solving HOT results in a slightly greater runtime on average. We conjecture that this is due to the graph built for HOT containing more edges before reaching the sink.

7 OPTIMIZING NUMBER OF TARGETS COVERED - DISCUSSION

Until now we exclusively discussed the satisfiability problem, which is, finding a solution that covers all targets (while minimizing the traveled distances). In this section, we show that even if no such cover exists, the flow returned in AMAPFwID-DOT from a max-flow algorithm also solves the *optimization* variant of AMAPFwID, which maximizes the number of targets acquired by the agents.

Theorem 7.1. *Given an* AMAPFwID-DOT *problem instance, the maximum flow in the network created by Algorithm 1 maximizes the number of acquired targets.*

Proof outline. Assume a flow f with value $|f| = x$ for the reduction defined for AMAPFwID-DOT. This flow represents a partial plan S_A acquiring x targets. In S_A, each agent starts in its initial location and ends in a target at its deadline, while not conflicting with any other path. As agents disappear on target locations, other agents may pass through either acquired or not acquired targets. Thus, the partial plan is valid for AMAPFwID-DOT. Similarly, assume a partial plan S_A acquiring x targets. In S_A, agents start at initial locations, disappear on targets, and do no conflict. In our reduction for AMAPFwID-DOT, agents may pass any target after its deadline, and a similar flow that represents S_A exists in the network for DOT. □

In Figure 10, we have the percentage of goals covered for all non-satisfiable instances of DOT in all 3 maps. We can see an increase in coverage as the number of agents becomes larger, which is a result of the map getting filled with agents, and so the average distance between an agent and a goal is getting smaller. Also, the high values of the DOT coverage indicate that using this problem setting is also beneficial in cases when a solution was not found.

It is important to note that this optimization of the number of the acquired targets does not work for SOT or HOT:

- In AMAPFwID-SOT, the targets that are not acquired by any agent should be available for the agents to pass through after their deadlines. However, the modification defined for AMAPFwID-SOT does not allow other agents to visit a target location after its deadline, even if no agent has acquired this target location. This kind of conditioning does not exist in our reduction to AMAPFwID-DOT.
- In AMAPFwID-HOT, we should be able to differentiate between acquired and not acquired target locations, as well. However, according to our modification, traversing a location-occupancy edge of a target location after the deadline costs 0. Therefore, the maximum flow may not represent the case of maximum acquired targets as the targets may not be acquired at all timesteps from their deadlines.

As seen above, optimizing the number of acquired targets may require solving a Max-Flow problem in networks with conditional properties on the flow, which unfortunately is a generalization of the NP-hard Integer Equal Flow problem [19]. We leave the research on methods for the optimization of the number acquired targets in SOT and HOT for future work.

8 CONCLUSIONS AND FUTURE WORK

This paper focuses on the Anonymous Multi-Agent Path Finding problem, in which n agents are required to acquire n targets, and those targets are associated with individual deadlines. Thus, the problem is satisfied only if each target is acquired by some agent by its respective deadline. We examine three possible agent behavior when reaching the target: either it disappears once the deadline is reached (DOT), stays in place (SOT), or has the ability to move after the deadline, but only if relaying the target acquisition to another agent replacing it by performing *hot swapping* (HOT). We describe algorithms for all settings, using a modified network-flow reduction, and prove that not only do these algorithms solve the problem, they provide a solution that is optimal with respect to the traveled distances. Further, we prove that when agents disappear at targets, optimization of the number of acquired targets can be solved in polynomial time, as well. We have shown empirically that HOT significantly extends the satisfiability of the problem in various settings.

The possibility of relaying the target acquisition to another agent raises many possibilities for future work. First, we would like to examine the optimality of the number of acquired targets in SOT and HOT for AMAPFwID, as mentioned above. Also, hot swapping can contribute to other versions of AMAPF, for example, we can generalize the case where teams of agents exist, and each team needs to accomplish a AMAPF problem [17], to the case where each team is required to solve a AMAPFwID problem. Another example is generalizing the *evacuation* case, where there are fewer targets than agents and each agent needs to be evacuated from one of the targets [30], to the case of AMAPFwID where each evacuation target has a deadline. Moreover, one may try to define a special MAPF variant of hot swapping where each agent must end at a specific target while its target can be acquired by another agent before the specified agent reaches its target.

ACKNOWLEDGEMENTS

This research was supported in part by ISF grant #2306/18

REFERENCES

[1] Luka Antonyshyn, Jefferson Silveira, Sidney Givigi, and Joshua Marshall. 2022. Multiple Mobile Robot Task and Motion Planning: A Survey. *ACM Comput. Surv.* (2022).

[2] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *the International Joint Conference on Artificial Intelligence (IJCAI)*. 740–746.

[3] Fatma Faruq, Bruno Lacerda, Nick Hawes, and David J. Parker. 2018. Simultaneous Task Allocation and Planning Under Uncertainty. In *the International Conference on Intelligent Robots and Systems (IROS)*. 3559–3564.

[4] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. 2018. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In *the International Conference on Automated Planning and Scheduling (ICAPS)*. 83–87.

[5] Lester Randolph Ford and Delbert R Fulkerson. 2009. Maximal flow through a network. In *Classic papers in combinatorics*. Springer, 243–248.

[6] Lester R Ford Jr. 1956. *Network flow theory*. Technical Report. Rand Corp Santa Monica Ca.

[7] Graeme Gange, Daniel Harabor, and Peter J. Stuckey. 2019. Lazy CBS: implicit Conflict-based Search using Lazy Clause Generation. In *the International Conference on Automated Planning and Scheduling (ICAPS)*. 155–162.

[8] Tzvika Geft and Dan Halperin. 2022. Refined Hardness of Distance-Optimal Multi-Agent Path Finding. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 481–488.

[9] Christian Henkel, Jannik Abbenseth, and Marc Toussaint. 2019. An optimal algorithm to solve the combined task allocation and path finding problem. *arXiv preprint arXiv:1907.10360* (2019).

[10] Marika Ivanová and Pavel Surynek. 2013. Adversarial cooperative path-finding: a first view. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*.

[11] Marika Ivanová and Pavel Surynek. 2014. Adversarial cooperative path-finding: Complexity and algorithms. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*. IEEE, 75–82.

[12] Marika Ivanová and Pavel Surynek. 2017. Area Protection in Adversarial Path-Finding Scenarios with Multiple Mobile Agents on Graphs: a theoretical and experimental study of target-allocation strategies for defense coordination. *arXiv preprint arXiv:1708.07285* (2017).

[13] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[14] Edward Lam, Pierre Le Bodic, Daniel Harabor, and Peter J. Stuckey. 2019. Branch-and-Cut-and-Price for Multi-Agent Pathfinding. In *the International Joint Conference on Artificial Intelligence (IJCAI)*. 1289–1296.

[15] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. 2019. Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding. In *the AAAI Conference on Artificial Intelligence (AAAI)*. 6087–6095.

[16] Hang Ma, Daniel Harabor, Peter J Stuckey, Jiaoyang Li, and Sven Koenig. 2019. Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7643–7650.

[17] Hang Ma and Sven Koenig. 2016. Optimal target assignment and path finding for teams of agents. In *the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 1144–1152.

[18] Hang Ma, Glenn Wagner, Ariel Felner, Jiaoyang Li, TK Kumar, and Sven Koenig. 2018. Multi-agent path finding with deadlines. In *the International Joint Conference on Artificial Intelligence (IJCAI)*. 417–423.

[19] Carol A Meyers and Andreas S Schulz. 2009. Integer equal flows. *Operations Research Letters* 37, 4 (2009), 245–249.

[20] Malcolm Ross Kinsella Ryan. 2008. Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research* 31 (2008), 497–542.

[21] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

[22] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence* 195 (2013), 470–495.

[23] David Silver. 2005. Cooperative Pathfinding. *AIIDE* 1 (2005), 117–122.

[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 *the International Symposium on Combinatorial Search (SoCS)*. 151–159.

[25] Pavel Surynek. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *the AAAI Conference on Artificial Intelligence (AAAI)*. 1261–1263.

[26] Pavel Surynek. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *the Pacific Rim International Conference on Artificial Intelligence (PRICAI)*. 564–576.

[27] Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijaya Kondepogu. 2014. Goal assignment and trajectory planning for large teams of interchangeable robots. *Autonomous Robots* 37 (2014), 401–415.

[28] Glenn Wagner and Howie Choset. 2015. Subdimensional expansion for multirobot path planning. *Artificial Intelligence* 219 (2015), 1–24.

[29] Hanfu Wang and Weidong Chen. 2022. Multi-Robot Path Planning With Due Times. *IEEE Robotics and Automation Letters* 7, 2 (2022), 4829–4836.

[30] Jingjin Yu and Steven M LaValle. 2013. Multi-agent path planning and network flow. In *Algorithmic foundations of robotics X*. Springer, 157–173.

[31] Jingjin Yu and Steven M. LaValle. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *the AAAI Conference on Artificial Intelligence (AAAI)*. 1443–1449.