Coordination of Multiple Robots along Given Paths with Bounded Junction Complexity

Mikkel Abrahamsen
University of Copenhagen, Denmark
miab@di.ku.dk

Tzvika Geft
Tel Aviv University, Israel
zvigreg@mail.tau.ac.il

Dan Halperin
Tel Aviv University, Israel
danha@tauex.tau.ac.il

Barak Ugav
Tel Aviv University, Israel
barakugav@gmail.com

ABSTRACT

We study a fundamental NP-hard motion coordination problem for multi-robot/multi-agent systems: We are given a graph G and set of agents, where each agent has a given directed path in G. Each agent is initially located on the first vertex of its path. At each time step an agent can move to the next vertex on its path, provided that the vertex is not occupied by another agent. The goal is to find a sequence of such moves along the given paths so that each agent reaches its target or to report that no such sequence exists. The problem models guidepath-based transport systems, which is a pertinent abstraction for traffic in a variety of contemporary applications, ranging from train networks or Automated Guided Vehicles (AGVs) in factories, through computer game animations, to qubit transport in quantum computing. It also arises as a subproblem in the more general multi-robot motion-planning problem.

We provide a fine-grained tractability analysis of the problem by considering new assumptions and identifying minimal values of key parameters for which the problem remains NP-hard. Our analysis identifies a critical parameter called *vertex multiplicity* (VM), defined as the maximum number of paths passing through the same vertex. We show that a prevalent variant of the problem, which is equivalent to Sequential Resource Allocation (concerning deadlock prevention for concurrent processes), is NP-hard even when VM is 3. On the positive side, for VM ≤ 2 we give an efficient algorithm that iteratively resolves cycles of blocking relations among agents. We also present a variant that is NP-hard when the VM is 2 even when G is a 2D grid and each path lies in a single grid row or column. By studying highly distilled yet NP-hard variants, we deepen the understanding of what makes the problem intractable and thereby guide the search for efficient solutions.

KEYWORDS

multi-robot motion planning; multi-agent path finding; predefined paths; sequential resource allocation; scheduling; complexity

ACM Reference Format:
Mikkel Abrahamsen, Tzvika Geft, Dan Halperin, and Barak Ugav. 2023. Coordination of Multiple Robots along Given Paths with Bounded Junction Complexity. 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

We study the problem of coordinating the motion of a fleet of robots/agents[1] with assigned paths. The problem arises in the context of guidepath-based vehicles, such as Automated Guided Vehicles and overhead monorail systems that are used in industrial environments [28]. Such environments are typically highly structured and constrain the vehicles to move along predefined paths in a centrally controlled manner. A crucial component of such systems is ensuring *liveness*, which is the ability of the vehicles to complete their assigned tasks and perform similar future tasks. Liveness can be lost due to *deadlocks*, which can arise due to the fact that certain vehicle motions are irreversible, for example, when a vehicle cannot move backward on a railway. Before entering a new state, such as giving a vehicle a new assigned path, it is desirable to check if liveness can be preserved. In general, this check amounts to determining the existence of a sequence of motions that allows all agents to complete their current trips, which in turn boils down to solving our problem. This problem of checking state liveness is known as liveness-enforcing supervision and has received interest from the Discrete Event Systems (DES) community [27].

Another motivation comes from the world of smart transportation. Recent years have demonstrated that operating autonomous vehicles in mixed traffic/urban areas remains highly challenging and is therefore unlikely to prevail soon. A more viable setting for operating them safely is using dedicated infrastructure (e.g., guideways, rails, or dedicated lanes), which is simpler due to limited interaction with human drivers, pedestrians, and obstructions. Unlike regular road networks, such infrastructures constrain vehicles to move on a limited set of paths, which ultimately lends itself to our setting. Such constrained autonomous systems are expected to evolve beyond simple topologies and fixed schedules (e.g., in airport shuttles) [18] and hence demand more complex motion coordination. For example, to cater for increased demand, which will also be flexible (e.g., as a result of on-demand service), a system's efficiency might be increased by allowing vehicles heading in opposite directions to use the same path segments. Indeed, developing algorithms for the special structure of dedicated infrastructure transport systems has been recently highlighted as a research direction in urban mobility and logistics [18].

[1]We interchangeably use the terms *agents* and *robots* in this work. In the terminology of the motion-planning literature, robots are typically used when moving in continuous domains, and agents (or pebbles, among others) are used for motion on graphs. The distinction is sometimes blurred since often continuous motion-planning problems are reduced to motion planning of agents on graphs.

Our problem belongs in the wider context of *Multi-Robot Motion Planning* (MRMP). In MRMP, instead of having the whole path specified, we are given only the start and target for each robot/agent, and the goal is to find a collision-free motion that brings all the robots to their targets. The general problem has been shown to be PSPACE-hard in various planar settings [5, 14, 16, 33]. The problem is remains NP-hard even when all the robots move one by one [12]. Relaxations involving assumptions on the spacing between robots in their start and target placements have been introduced in order to make the problem tractable [2, 32, 34]. The discrete counterpart of MRMP, where agents move on a graph, is solvable in polynomial time [19, 38]. However, when optimal solutions are sought, e.g., with respect to a time or distance objective, the problem becomes NP-hard, even on 2D grid graphs [3, 9, 11]. This intractability has been recently tackled by approximation algorithms [9, 36, 37]. We remark that there is a tremendous body of work on MRMP variants, for which it is impossible to do justice here.

Although MRMP has more freedom than our problem due to not constraining paths a priori, MRMP with given paths (MRMP-GP for short) is useful as a subroutine for solving MRMP. A common paradigm for solving MRMP, known as *decoupled planning*, is to first plan the path of each robot without taking other robots into account. Then, some form of coordination between robots given their individual paths follows, such as adjusting the robots' speeds along paths. Early works focused on variants of the problem for a constant number of robots [17]. In [22], the *coordination diagram*, which represents placements along each robot's path at which mutual collisions might occur, was used for two robots. The diagram has later been generalized for multiple robots [20] and has since been used to coordinate their motion along assigned paths [23, 31]. However, the size of the diagram has a worst-case exponential dependency on the number of robots. A different flavor of works focuses on the *execution* of paths (more precisely, trajectories) that have already been planned. In practice, there is no guarantee that a robot will follow a planned trajectory. An external interference might invalidate an initially valid plan. Therefore, there is a perpetual need to verify and coordinate the motion of robots while they move along planned paths (possibly replanning them). Some works of this flavor include [4, 6, 7, 15].

Despite sustained interest in the problem, the first NP-hardness results explaining the lack of efficient algorithms appeared, to our knowledge, only relatively recently [27, 28]. These results have been presented as part of a line of work on deadlock prevention in resource allocation problems [24], affiliated with the DES community, which have wide applicability in various automation scenarios [25]. Such problems involve allocating a finite set of reusable resources to a set of concurrently executing processes, where each process can only be executed by acquiring resources in a certain order. Our problem can be seen as being equivalent to the simplest class of resource allocation problems, known as Linear Single-Unit Resource Allocation Systems (L-SU-RAS), which is NP-hard [24, 28]. In the terms of L-SU-RAS, each robot emulates a process that needs to "acquire" one location at a time from a sequence of locations, each of which can host only a single robot at a time.

We observe the following undesirable properties of previous hardness constructions: (i) Unbounded vertex multiplicity, i.e., there is a "congested" location that has to be visited by an unbounded number of robots. (ii) There are path segments that have to be traversed in opposite directions, i.e., an inherent potential for a head-on collision exists. (iii) The robot's paths are not the shortest between their endpoints. In particular, robots have to visit the same vertex multiple times along their path (in proofs for the planar case). We summarize the previous hardness results in $Table$ 1. Arguably, these properties do not necessarily represent real systems, which prompts new analysis for cases where they do not hold.

Contribution. In this work, we perform a fine-grained complexity study of the problem of coordinating the motion of robots along fixed paths. We consider two parameters: vertex multiplicity and the path shape complexity, which is the maximum number of turns made on the grid (exact definitions are provided in Section 2). We consider two main problem variants: (1) a prominent variant that corresponds to the L-SU-RAS resource allocation problem, and (2) a lesser-studied variant where the aforementioned property (ii) of opposite direction paths does not hold. For each variant, we identify the critical values of the parameters for which the problem remains NP-hard, such that below this value the problem is efficiently solvable. By that, we establish a sharper boundary between negative and positive results.

Our main positive result is for variant (1) where we present an efficient algorithm solving the problem for any graph such that the vertex multiplicity is 2. By that, we expand the class of efficiently solvable instances. At the heart of our solution is an iterative procedure for resolving cycles of blocking relations among agents. For each such cycle, we construct a special type of a directed graph, which we call a *graph composed of paths*. We repeatedly contract this graph until it reaches a problem-equivalent simplified and irreducible state in which it is easy to determine whether the blocking can be untangled or otherwise a deadlock is detected and the instance has no solution. The algorithm runs in time linear in the total lengths of the given paths.

On the negative side, we show that for a vertex multiplicity of 3, the problem is NP-hard. Furthermore, for variant (2) we show that the problem is NP-hard for agents moving along straight paths on a 2D grid graph where the vertex multiplicity is 2, which is considerably more restricted than previous hardness results.

Table 1: Comparison to previous hardness results. For exact definitions of problem variants and parameters see Section 2. A check mark in the last column ("Shortest paths?") indicates that each given path in the construction is the shortest between its start and target vertices. We use "min" to indicate that a parameter is the minimum value below which the problem variant is efficiently solvable.

Problem variant	Paper	Graph type	Vertex multiplicity	Max. # turns in path	Shortest paths?
NBT	[21]	general	unbounded	n/a	✓
	[27]	planar	unbounded	n/a	
	[28]	2D grid	unbounded	>20	
	ours	**general**	**3 (min)**	**n/a**	✓
	ours	**2D grid**	**4**	**1 (min)**	✓
UNI	**ours**	**2D grid**	**2 (min)**	**0 (min)**	✓

Our fine-grained complexity analysis echoes recent calls for deepening the understanding of what makes multi-agent motion coordination problems hard [10, 13, 29]. Such an improved understanding can guide the search towards improved algorithms and new efficiently solvable problem variants. We discuss potential developments in this spirit in the conclusion.

Due to space constraints, we omit certain details and proofs, which can be found in the full version [1].

2 PROBLEM DEFINITION AND ASSUMPTIONS

In this section, we formally define the problem that we study, along with the assumptions and parameters we consider.

Multi-robot motion planning with given paths (MRMP-GP).. We are given a set R of n robots that operate in a workspace W, which is a finite undirected graph. The vertices of W, denoted by $V(W)$, may also be called *positions*. Each robot $r \in R$ has a start vertex $s(r) \in V(W)$ (also referred to as a *source*) and a target vertex $t(r) \in V(W)$, and a path $\pi(r) = v_1, \ldots, v_\ell$, where $v_1 = s(r), v_\ell = t(r)$ and $v_i v_{i+1}$ is an edge of W for all $1 \le i < \ell$. Note that since $\pi(r)$ is a path, each vertex only appears once in $\pi(r)$.

Each robot is initially located at its start vertex. MRMP-GP asks to find a *motion plan* or *solution*, which is a sequence of moves that bring each robot from its source to its target using its given path without inducing collisions with other robots. A legal move consists of a single robot moving to the next vertex v on its path, provided that no other robot is located at v. No backward moves are allowed, i.e., a robot may not move to the previous vertex on its path. In particular, once a robot reaches its target it stays there.

Figure 1 shows an example of an instance. Note that MRMP-GP is a feasibility problem and not an optimization problem; hence, we only move one robot at a time in a solution.

Variants. Now we define the two variants of the problem we consider. Let r and r' be two different robots. We say that $t(r')$ is a *blocking target* of r if $t(r')$ lies on $\pi(r)$. An instance is said to have the *non-blocking targets* property if it does not have blocking targets. This case corresponds to the L-SU-RAS resource allocation problem [26] because in this problem when a process completes it is essentially gone from the world. This is equivalent to a robot whose target is not a blocking target of any other robot since we can consider the robot as disappearing once it reaches its target.

For our second variant, an edge (u, v) in W is called *bi-directional* if it has to be traversed in both directions by different robots, i.e., there are two robots r and r' where (u, v) appears on $\pi(r)$ and (v, u) appears on $\pi(r')$. An instance is said to have *one-way* or *unidirectional* motion if it has no bi-directional edges.

Parameters. For a vertex $v \in V(W)$, we denote by $N(v)$ the number of paths in which it appears, i.e., $N(v) := |\{r \mid v \in \pi(r), r \in R\}|$. We define the *vertex multiplicity* of an MRMP-GP instance M, denoted by $\mathsf{VM}(M)$, to be $\max_{v \in V(W)} N(v)$. For the case where W is a 2D grid, we examine path shape complexity. We define the *turn number* to be the maximum number of turns, i.e., the minimum number of line segments needed to draw a path on the grid minus one, made by any input path.

Shorthand notation. We use the following shorthand notation throughout the paper. For the problem variant with non-blocking

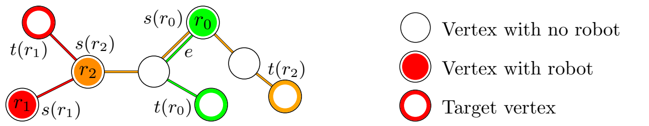

Figure 1: An MRMP-GP instance in which W has 8 vertices. There are 3 robots, each with its own source, target and path which are shown in the robot's color. The instance has the non-blocking targets property but does not have one-way motion (due to the edge e). Its vertex multiplicity is 2. A possible solution to this instance is to move each robot all the way to its target in the order r_0, r_2, r_1. (While it is not needed in this example, in general we allow a robot to stay put in an intermediate vertex along its path, while other robots move.)

targets, we use MRMP-GP-NBT(x), where x indicates the vertex multiplicity. We similarly use MRMP-GP-UNI(x) for the variant that has uni-directional motion. When W is a 2D grid, we also indicate the turn number as the second parameter, e.g., MRMP-GP-UNI(2,0) indicates a VM of 2 and straight paths (i.e., 0 turns).

3 ALGORITHM

We now consider MRMP-GP-NBT(2), i.e., the variant of MRMP-GP having the non-blocking targets property with a vertex multiplicity of at most 2. We present a polynomial-time algorithm that finds a solution or reports that none exists. The algorithm has two phases, which we now describe.

Phase 1. A robot r located at vertex $v \in \pi(r)$ is said to have a *clear path* if there are no other robots along the remainder of its path, i.e., the subpath of $\pi(r)$ from v to $t(r)$. Given the restriction that a robot's target is not included in any other robot's path, a robot with a clear path that is moved to its target will not block any other robot in the future. Hence, we move each robot with a clear path to its target until no such robot exists. Note that moving one robot may clear the path of another robot, therefore the robots are checked repeatedly.

Phase 2. Next, the algorithm iteratively identifies and solves *cycles*. In each iteration, a single cycle is solved, which amounts to moving only the cycle's robots. At the beginning of each iteration, we have the following invariant:

> Each robot either (i) reached its target vertex, or (ii) is still at its source vertex and it does not have a clear path.

At the end of Phase 1, the invariant is clearly maintained. A robot r is said to be *blocked* by another robot r' if the path from the current position of r to $t(r)$ contains the current position of r'. At the beginning of each iteration, any robot that has not reached its target yet is blocked by some other robot. Let r_0 be one such robot and let r_1 be the robot blocking it, which must therefore not be at its target $t(r_1)$. Then robot r_1 has not yet reached its target, therefore the path $\pi(r_1)$ contains some other robot r_2, which blocks r_1, and so

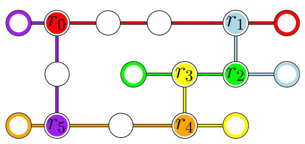

Figure 2: A simple cycle of 6 robots, $r_0, \ldots, r_5$, where each robot is blocking the previous one from reaching its target.

on. The blocking relationship yields a sequence $r_0, r_1, r_2, \ldots, r_{h-1}$, where r_{h-1} is the first robot that is blocked by a robot r_i already appearing in the sequence, i.e., $i < h-1$. See Figure 2 for an illustration. To simplify notation, from this point on we use indices of robots modulo h, i.e., we write r_i instead of $r_{(i \mod h)}$.

Lemma 3.1. *A blocking sequence $r_0, r_1, ..., r_{h-1}$ that ends when r_{h-1} is blocked by some robot r_i, for $0 \le i < h-1$, forms a cycle, i.e., each robot r_i is blocked by r_{i+1}.*

Proof. Any robot r_j, $j < h-1$ is blocked by r_{j+1} by definition. Hence, it remains to prove that r_{h-1} is blocked by r_0. Assume for a contradiction that r_{h-1} is blocked by a robot r_i with $i > 0$. This means that the paths of the robots r_{i-1}, r_i, r_{h-1} all contain the start vertex $s(r_i)$, contradicting the assumption that each vertex is contained in at most two robots' paths. □

Therefore, all the robots that are not at their target can be divided into disjoint cycles. The algorithm *solves* each cycle (in a sense that we define next) independently, and then moves the cycle robots to their targets. This is possible since, as stated in Lemma 3.2 below, their paths must be clear.

Let us fix a cycle $C = r_0, r_1, ..., r_{h-1}$. Each robot $r_i \in C$ is currently at its source vertex. *Solving* the cycle C is defined as moving each robot r_i to the start position of the next robot, i.e., $s(r_{i+1})$, and $s(r_{i+1})$ is called the *cycle target* of r_i.

Lemma 3.2. *After C is solved, all the robots in C have a clear path.*

Proof. Assume by contradiction that some robot r_i of C is blocked by another robot r after solving C. If r is not in C, then r belongs to another cycle, where it blocks another robot r'. This means that r blocks two robots at its current vertex, which contradicts the assumption that vertex multiplicity is 2. If r is also part of C, then r was blocked by a robot r' in C. After solving C, r is at the source vertex $s(r')$, which implies that $s(r')$ appears on 3 robot's paths, namely, r, r', and r_i, which again contradicts the assumption of a vertex multiplicity of 2. □

Solving a cycle. The subpath of robot r_i from $s(r_i)$ to $s(r_{i+1})$ is called the *cycle path* of robot r_i. If there is a robot r_j with a vertex p on its cycle path that does not appear in the cycle path of another robot in C, then we call r_j a *scout*. For example, r_0 in Figure 2 is a scout robot. If C contains a scout robot, then C can be solved by moving the scout robot to p and then moving each robot in the reverse order from the scout robot along its cycle path. We omit the full details of this simple case.

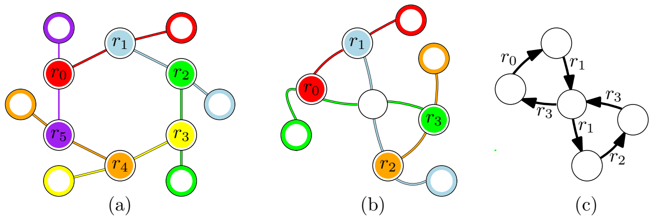

Figure 3: Unsolvable instances, each containing a single cycle: (a) A deadlock where no robot can move. (b) Either r_1 or r_3 can move, but either move will result in a deadlock of three robots similar to (a). (c) The graph G constructed by the algorithm for the cycle in (b).

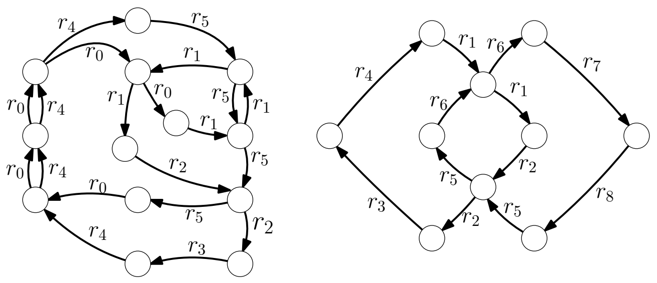

Figure 4: Left: A graph composed of paths, which is solvable according to Lemma 3.4. Right: A graph that is not composed of paths, since the edges do not form an Eulerian cycle of the required type.

If there is no scout robot in C, a solution might not exist; see Figure 3. To determine whether a cycle is solvable, we construct a directed graph G with vertices $V(G) \subset V(W)$ and edges $E(G)$. The vertices $V(G)$ are a subset of the vertices on the cycle paths of the robots in C. For each robot r in C and each directed edge e on the cycle path of r, we add the edge e to $E(G)$. We label this edge by r to remember which robot induced it. It is therefore possible that we have two edges from one vertex to another in G, but they will be labeled by different robots. We say that a directed graph with labeled edges of this form is *composed* of paths; see Figure 4. Note that the graph G has the following properties:

- For each distinct label, the edges with that label form a path.
- Each vertex appears on the paths of exactly two labels.
- The end vertex of a path with one label is the start vertex of a path with another label.
- The start vertex of a path with one label is the end vertex of a path with another label.
- There is an Eulerian cycle in G such that for each label, the edges with that label are traversed consecutively in the cycle.

Any directed graph with labeled edges that has these properties is said to be *composed* of paths. A *start* vertex of such a graph G is the start vertex of the path with any label, and we denote the set of start vertices as $V_s(G)$.

A graph G which is composed of paths represents an equivalent robot cycle obtained in the following simple way. For the path π in G consisting of all edges with label r, we place the robot r on the start vertex of π, and the robot r must traverse the path π. See Figure 3(c) for illustration. If this instance of MRMP-GP has a motion plan, we say that G is *solvable*.

Let us first prove an elementary lemma about the degrees of a graph G composed of paths. Let $\delta_{\text{in}}(v)$ and $\delta_{\text{out}}(v)$ denote the in- and out-degree of a vertex v of G.

Lemma 3.3. *Let G be a graph composed of paths and v be a vertex of G. If $v \in V_s(G)$ then we have $(\delta_{in}(v), \delta_{out}(v)) = (1, 1)$ and otherwise $(\delta_{in}(v), \delta_{out}(v)) = (2, 2)$.*

Proof. Consider $v \in V_s(G)$. By the properties of G, the path of one label starts at v, and the path of another label ends at v. Since v only appears on the paths of two labels, the statement follows.

Now consider a vertex $v \in V(G) \setminus V_s(G)$. By the properties of G, the vertex v is included in the paths of two labels and is not a target vertex of either of them. This means that on both paths, there are edges to v and edges out of v, so $(\delta_{\text{in}}(v), \delta_{\text{out}}(v)) = (2, 2)$. □

We now turn our attention to classifying the solvable graphs that are composed of paths. Lemmas 3.4 and 3.5 present the **base cases** of solvable vs. unsolvable graphs:

Lemma 3.4. *Consider a graph G that is composed of paths. If all simple cycles in G have at least two vertices that are not start vertices, then G is solvable.*

Lemma 3.5. *If a graph G composed of paths contains a simple cycle where all vertices are start vertices, then G is not solvable.*

We omit the proof of Lemma 3.5. As for Lemma 3.4, we postpone the proof to first describe the algorithm that solves the corresponding instance of MRMP-GP; see Figure 5 for a non-trivial example.

The algorithm begins by partitioning the robots into blocks as follows. Consider a maximal directed path π in G with at least 3 vertices such that each vertex of π except the first and last is a start vertex. Let $r_1, \ldots, r_k$, $k \geq 1$ be the corresponding robots starting on these start vertices in order, and let $B_1 = (r_1, \ldots, r_k)$. Running over all such paths π, we obtain blocks of robots $B_1, B_2, \ldots$.

For a block $B_i = (r_1, \ldots, r_k)$, we then define $head(B_i) = r_k$ and $tail(B_i) = r_1$. The robot r_k has its *cycle target* at the start vertex of a robot r'_1 in another block B_j. We then define $next(B_i) = B_j$ and $prev(B_j) = B_i$. Due to the properties of a graph composed of paths, these relations are defined for all blocks B_i.

We now describe our algorithm; See Algorithm 1 for pseudocode. We first choose an arbitrary block B_i. Then, we move all robots of B_i a single edge forward from $head(B_i)$ to $tail(B_i)$. Move $head(B_i)$ along its cycle path until it cannot be moved any further. Let $B_j = next(B_i)$ be the block containing the robot, $tail(B_j)$, that blocks $head(B_i)$. Move the robots B_j forward by one edge and then move $head(B_i)$ to its cycle target, which is the original position of $tail(B_j)$. Move $head(B_j)$ as much as possible. From here on, the pattern repeats until all the blocks have been traversed.

Proof of Lemma 3.4. We show that Algorithm 1 solves $I(C)$, the instance of MRMP-GP corresponding to G of cycle C. When solving the cycle, we move each block of robots B_i from head to tail. Clearly, only moving $head(B_i)$ can be an invalid motion, as the rest of the robots in B_i are always moved to a vertex that was just evacuated by the previously moved robot.

Denote by u, v the vertices that are occupied by the tail and head of B_1 respectively (u, v might be the same vertex if B_i consists of a single robot). Both u, v are in $V_s(G)$, so by Lemma 3.3, u has a single in-going edge, from another vertex w, and v has a single out-going edge, to another vertex z. Both w, z are not occupied by robots and are not in $V_s(G)$, otherwise these robots would have been included in B_i. If $w = z$, then w and all the vertices occupied by the robots of B_i form a simple cycle with a single non-start vertex, in contradiction to the lemma's assumption, therefore $w \neq z$.

Algorithm 1: Solve an instance of MRMP-GP corresponding to a graph G composed of paths satisfying the requirements of Lemma 3.4.

```
1  $B_{start} \leftarrow B_1$;
2  $B \leftarrow B_{start}$;
3  **Loop**
4      **for** $r \in B$ *in order from* $head(B)$ *to* $tail(B)$ **do**
5          Advance $r$ one edge;
6      **if** $B$ *is not* $B_{start}$ **then**
7          Advance $head(prev(B))$ one edge;
8      Advance $head(B)$ as much as possible;
9      $B \leftarrow next(B)$;
10     **if** $B$ *is* $B_{start}$ **then**
11         **return**;
```

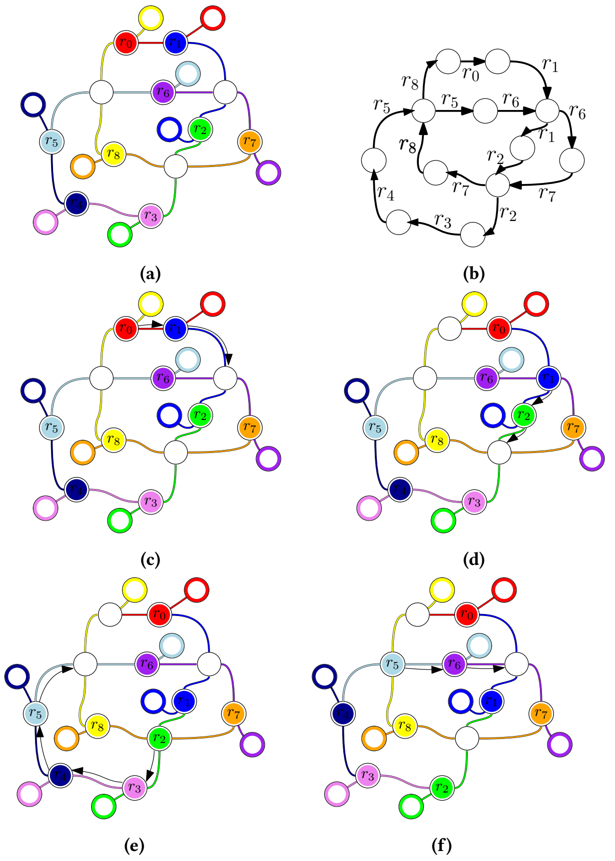

Figure 5: (a) An instance with a single blocking cycle. (b) The graph G created for the single cycle, in which all simple cycles contain two free (non-start) vertices. The blocks of maximal consecutive robots are $B_1 = (r_0, r_1), B_2 = (r_2), B_3 = (r_3, r_4, r_5), B_4 = (r_6), B_5 = (r_7), B_6 = (r_8)$. We illustrate a few steps of the solution obtained by Algorithm 1: (c) $B_1 = (r_0, r_1)$ is the first chosen block, for which robots are moved head to tail. (d) The robots of *next*(B_1), $B_2 = (r_2)$, are moved head to tail, and then *head*(B_1) is moved again. (e) The robots *next*(B_2), $B_3 = (r_3, r_4, r_5)$, are moved head to tail, and then *head*(B_2) is moved. (f) $B_4 = (r_6)$ and *head*(B_3) are moved.

At the beginning of an iteration of the main loop (line 3), in which B_i is handled, we have the following invariant: At most one vertex of $V(G) \setminus V_s(G)$ is occupied by a robot, and the occupied vertex is the source of the edge directed to the vertex $s(tail(B_i))$. In other words, the occupied vertex is w, as defined above. We now verify that the invariant is maintained.

Initially, all the robots are on $V_s(G)$ and the invariant is maintained. Suppose now that the invariant holds at the beginning of an iteration. Before moving any robot of $B = B_i$, w is the only non-start vertex that can be occupied, therefore z is never occupied, and moving $head(B_i)$ a single step to z is valid since $w \neq z$.

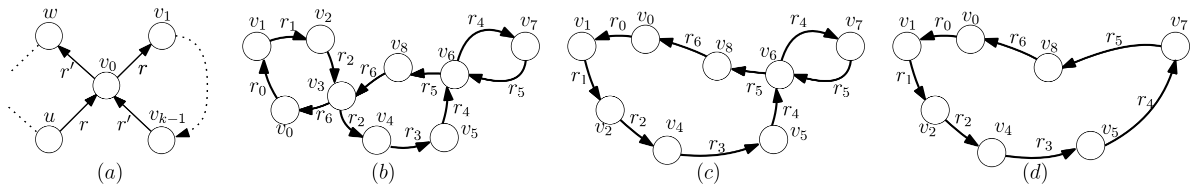

Figure 6: (a) A cycle $v_0, \ldots, v_{k-1}$ that requires untangling. (b) A graph composed of paths that requires two untangling operations, shown in (c) and (d), after which we conclude that the graph is unsolvable.

After moving the head of B_i, the rest of the robots of B_i are moved head to tail, and the robot that occupied w is moved to the original position of the tail of B_i. Lastly, the head of B_i is moved as much as possible, until it reaches w', the source vertex of the in-edge of $s(\mathit{tail}(\mathit{next}(B_i)))$, and the invariant is maintained. When the last block is handled, the head is moved as much as possible and reaches $s(\mathit{tail}(\mathit{next}(B_i)))$. Therefore the motion is valid.

We now verify that all robots get to their cycle targets. After the algorithm handles a single block B_i, all robots in B_i except $\mathit{head}(B_i)$ reach their cycle targets. As part of handling the next block $\mathit{next}(B_i)$, $\mathit{head}(B_i)$ is moved again and reaches its cycle target, so all the robots in B_i traverse their cycle path. If B_i is the last block handled, then all the robots of other blocks reached their cycle targets, including $\mathit{tail}(B_{start})$, therefore when $\mathit{head}(B_i)$ is advanced as much as possible it will reach its cycle target, the start position of $\mathit{tail}(B_{start})$. All the blocks $B_1, B_2, \ldots$ are examined, therefore all the robots reached their cycle targets and the cycle is solved. □

If G does not fall under one of the base cases of Lemmas 3.4 and 3.5, we gradually reduce G to a graph that does fall under one of these cases, as we describe next. So suppose that we are in a situation not included in the base cases, namely that G has a simple cycle $v_0, \ldots, v_{k-1}$ where v_0 is not a start vertex, but all other vertices are. Note that we must have $(\delta_{\text{in}}(v_0), \delta_{\text{out}}(v_0)) = (2, 2)$, according to Lemma 3.3. It follows that (v_0, v_1), (v_{k-1}, v_0), (u, v_0), (v_0, w) are edges in $E(G)$ incident to v_0, where u, w are vertices not appearing in the simple cycle $v_0, \ldots, v_{k-1}$. Moreover, there are two robots r and r' such that (u, v_0) and (v_0, v_1) are labeled with r, and (v_{k-1}, v_0) and (v_0, w) are labeled with r', as illustrated in Figure 6(a). If all four edges had the same label r then r would be a scout robot. The edges (v_{k-1}, v_0) and (v_0, v_1) cannot have the same label, as then the robots on $v_0, \ldots, v_{k-1}$ would form a blocking cycle independent of the rest of the blocking cycle C currently being solved.

We now define an *untangling* operation which leads to a new graph G' with vertices $V(G') = V(G) \setminus \{v_0\}$; see Figure 6. In G' we replace the two edges labeled by r with a single edge (u, v_1) and keep the label r. We likewise replace the two edges labeled r' with a single edge (v_{k-1}, w) labeled r'. Let us prove that the untangling operation maintains solvability.

Lemma 3.6. *Let G be a graph composed of paths and let G' be the graph obtained by performing the untangling process in G. Then (i) G' is also a graph composed of paths and (ii) G' is solvable if and only if G is solvable.*

Proof. Consider the untangling operation described above and the involved vertices v_{k-1}, v_0, v_1, u, w. First, it is easy to verify that the operation maintains the properties of a graph composed of paths. Now suppose that G is solvable. Two robots should pass through v_0: denote by r_u the robot that includes u, v_0, v_1 in its cycle path and r_w the robot that includes v_{k-1}, v_0, w in its cycle path. There is no robot at v_0, and the algorithm should decide which of r_u, r_w will go through v_0 first. If r_u is moved to v_0 before r_w, the robots enter a deadlock, as the vertices $v_0, \ldots, v_{k-1}$ will all contain robots (i.e., the same situation as in Lemma 3.5). Therefore r_w must move to v_0 before r_u. So r_w moves first to v_0 and then eventually to w, and then r_u moves to v_0 and then to v_1. It follows that in G', there is a motion plan where r_w first moves on the merged edge (v_{k-1}, w), then all robots on the path $v_1, \ldots, v_{k-2}$ move, and then r_u moves on the merged edge (u, v_1).

Suppose now that G' is solvable. We can then simulate the same solution in G: When r_w traverses the edge (r_{k-1}, w) in G', we let it traverse both edges (r_{k-1}, v_0) and (v_0, v_w) in G. Likewise, when r_u traverses (u, v_1) in G', we let it traverse both edges (u, v_0) and (v_0, v_1) in G. All other edges in G and G' are the same, and for these we copy the moves directly. We then have a solution for G. □

Our algorithm proceeds by untangling cycles until it is no longer possible. This results in a sequence of graphs $G_0, G_1, \ldots, G_m$, where $G_0 = G$ is the original graph and each G_i results from performing the untangling operation in G_{i-1}. By Lemma 3.6, all these graphs are composed of paths, and the resulting graph G_m is solvable if and only if the original graph G is solvable. Furthermore, there is nothing more to untangle in G_m, so we either have a simple cycle with robots on all vertices or there are at least two vertices with no robots on all simple cycles. It then follows from Lemmas 3.4 and 3.5 that there is a solution if and only if we are in the latter case.

After solving a cycle, all the cycle robots have a clear path to their target (see Lemma 3.2), and the algorithm moves all of them to their targets one by one. A single iteration of Phase 2 has ended, and the invariant that each robot is either at its start position or at its target position is maintained, as the only robots the algorithm moved were the cycle robots, and all of them reached their targets.

Any move we perform in the algorithm is such that there is a solution after the move if and only if there is a solution before it. Therefore, if the algorithm does not find a valid solution, namely one of the cycles was untangled to a graph with a simple cycle with robots on all vertices, there is no solution. Otherwise, the algorithm finds a valid motion plan for all robots.

Both phases of our algorithm run in time linear in the total lengths of the given paths. To keep Phase 1 efficient, the robots are checked for a clear path, and when a robot is moved, the blocked robots are checked only for clearness of their path suffix. To implement Phase 2 efficiently, each cycle is identified and solved independently. The size of the graph G is linear in the total length of the paths of the robots that are induced in the cycle and no more than $|V(G)|$ untangle operations need to be performed. See the full version [1] for a more detailed analysis.

Putting everything together, we obtain the following:

Theorem 3.7. *MRMP-GP-NBT(2) has an algorithm that finds a solution or reports that no solution exists, in time linear in the total lengths of the given paths.*

4 HARDNESS RESULTS

In this section, we present our hardness results, which are as follows:

Theorem 4.1. *The following MRMP-GP variants are NP-complete:*

(i) MRMP-GP-UNI(2, 0), i.e., the workspace is a 2D grid graph with VM = 2. *Furthermore, each agent's path is contained in a single grid row or column.*
(ii) MRMP-GP-NBT(4, 1), i.e., the workspace is a 2D grid graph with VM = 4. *Furthermore, each agent's path contains at most one turn.*
(iii) MRMP-GP-NBT(3), i.e., general graphs with VM = 3.

Note that in MRMP-GP-UNI we have unidirectional motion while blocking targets are allowed (see Section 2 for exact definitions). In contrast, in MRMP-GP-NBT we do not allow blocking targets, but do allow bi-directional motion, i.e., robots that traverse the same path but in opposite directions. Therefore, we can conclude that the presence of each of the elements of bi-directional motion or blocking targets by itself suffices to make MRMP-GP intractable.

Since our algorithm from Section 3 solves MRMP-GP-NBT(2), hardness result (iii) is tight. That is, we establish a tractability frontier based on vertex multiplicity for MRMP-GP-NBT. We identify additional tractability frontiers through the minimality of parameters as indicated in Table 1, which is easy to verify. To see that the turn number of MRMP-GP-NBT(4, 1) is minimal, we can reduce MRMP-GP-NBT(4, 0) to MRMP-GP-NBT(2).

We prove Theorem 4.1 in stages, establishing result (i), then (ii), and finally (iii). Here we sketch the proof, focusing on (i); see the full version [1] for the rest of the details. As it is straightforward to verify that MRMP-GP is in NP, we only discuss NP-hardness.

First, we show the NP-hardness of a variant of MRMP-GP, called *MRMP-GP with Precedence Constraints (MRMP-GP-PC)*. MRMP-GP-PC has additional constraints on the order in which robots visit vertices, which we use to abstract away the details of our complete constructions while showing the functionality of their gadgets. As part of the sketch, we illustrate how to realize the instance constructed by the reduction to MRMP-GP-PC, denoted by M_s, on the 2D grid. In our full proof, we incrementally convert M_s, to instances that are equally hard to solve. That is, each subsequent instance M' we describe is solvable if and only M_s is solvable.

The conversion steps of M_s are as follows. We first convert M_s to an instance M of MRMP-GP-UNI(2,0). To realize the gadgets in M, we use blocking targets. In the next stage, we convert M to M', in which we eliminate all the blocking targets. Such targets are replaced by paths going in opposite directions using the following mostly local changes: Let r, r' be two robots, where $t(r)$ is a blocking target located at vertex v, where $v \in \pi(r')$. We extend $\pi(r)$ so that it runs along $\pi(r')$ in the opposite direction. The change preserves the constraint that r must visit v before r' in a valid solution, which we use in our gadgets. The changes increase the VM of M' to 4 and make it an instance of MRMP-GP-NBT(4,1). Lastly, by carefully removing vertices with VM=4 from M' (which makes it no longer a 2D grid) we get an instance M'' of MRMP-GP-NBT(3).

We now begin the sketch, focusing on the hardness of MRMP-GP-UNI(2, 0). To establish the hardness of MRMP-GP-PC, we introduce a problem called Pivot Scheduling and prove that it is NP-hard. Next, we reduce Pivot Scheduling to MRMP-GP-PC.

Pivot Scheduling. An instance has the form (V, C), where V is a set of jobs that come in pairs and C is a set of ordering constraints: Let $V = \{x_1, y_1, \ldots, x_n, y_n\}$ be a set of $2n$ distinct jobs. Let $C = \{C_1, \ldots, C_m\}$, where each $C_j \in V^3$ is a triplet. The problem is to determine whether V can be partitioned into a before-set B and an after-set A (i.e., $A \cap B = \emptyset$ and $A \cup B = V$) such that the following constraints are satisfied: (i) for each pair x_i, y_i, we have either $x_i \in B$ or $y_i \in B$ and (ii) for each $C_j \in C$, one of the jobs in C_j must be in A, i.e., $C_j \cap A \neq \emptyset$.

We call the former constraints *before-constraints* and the latter constraints *after-constraints*. Intuitively, the before/after constraints implicitly imply the existence of a distinguished *pivot job* with respect to which the input jobs must be ordered. To be precise, B and A respectively correspond to the jobs that come before and after the pivot job.

We prove that Pivot Scheduling is NP-hard using a straightforward reduction from 3SAT; see the full version for the proof.

Lemma 4.2. *Pivot Scheduling is NP-hard.*

Hardness of MRMP-GP-PC.. Let us now define MRMP-GP-PC. The input is the same as MRMP-GP except that we also have special vertices, which we will use as gadgets. Each *gadget vertex* must be traversed (i.e., visited) by the robots passing through it in a certain *traversal order* (the exact constraints used will be specified in the reduction). A solution to MRMP-GP-PC is the same as for MRMP-GP with the additional requirement that gadget vertices must be visited according to their respective traversal order.

We proceed to the reduction. Given an instance of Pivot Scheduling $I = (V, C)$, we construct a corresponding MRMP-GP-PC instance M_s. We represent each job in V by a corresponding *job robot* in M_s. We also represent the implicit pivot job by the *pivot robot* r^*. To simplify notation, we use the same name for a job and its corresponding robot. Another robot is β, which can be thought of as a robot continuing the journey of r^*. Lastly, for each constraint $C_j = \{z_1, z_2, z_3\} \in V^3$, we have three corresponding *checker* robots c_j^1, c_j^2, c_j^3, which we use to emulate after-constraints.

We now discuss the gadgets in M_s. For each $i \in [n]$ we have a *before-constraint gadget*, denoted by B_i, which appears on the paths of r^*, x_i and y_i. The traversal order for B_i is defined as having either x_i or y_i traverse B_i before r^* does. For each $j \in [m]$ we have an *after-constraint gadget*, denoted by A_j, which appears on the paths of β and the checker robots c_j^1, c_j^2, c_j^3. The traversal order for A_j is defined as having one of c_j^1, c_j^2, c_j^3 traverse A_j after β does. The last gadget type we use is the *precedence gadget*, denoted by $\mathcal{P}(r, r')$, which must be visited first by r and then by r', where r and r' are arbitrary robots. The particular instances of this gadget that we use are specified below.

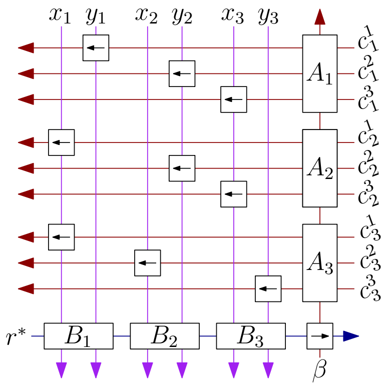

Figure 7: An MRMP-GP-PC instance corresponding to Pivot Scheduling with $V = \{x_1, y_1, \ldots, x_3, y_3\}$ and $C = \{\{y_1, y_2, x_3\}, \{x_1, y_2, x_3\}, \{x_1, x_2, y_3\}\}$. The robots' paths are shown as long arrows. Precedence gadgets are shown as squares containing a short arrow, which is oriented along the path of the robot that needs to traverse the gadget first.

We now describe the order of gadgets along the paths. The path $\pi(r^*)$ first passes through all before-constraint gadgets (in arbitrary order) and then passes through the precedence gadget $\mathcal{G} := \mathcal{P}(r^*, \beta)$. The path $\pi(\beta)$ first passes through $\mathcal{G}$, and then through all after-constraint gadgets (in arbitrary order). Since $\mathcal{G}$ is the last gadget along $\pi(r^*)$ and the first gadget along $\pi(\beta)$, β can essentially only start moving after r^* reaches its target. Now let c_j^ℓ be a checker robot, which corresponds to the job z in the constraint C_j. The robot c_j^ℓ first passes through A_j and then through the precedence gadget $\mathcal{P}(c_j^\ell, z)$. This means that before a job robot enters its respective B_i gadget, all the checker robots corresponding to the job must first traverse their respective A_j gadget.

Robots and path placements. An example of M_s is shown in Figure 7, which has a dual purpose of illustrating the conversion of M_s to the grid instance M. The figure should be interpreted as follows. Each (long and colored) arrow represents a path lying in a single grid row/column and the rectangles are gadgets. For M, each rectangle indicates the placement of a gadget, which contains a constant number of additional robots (not shown). For M_s, each gadget is a vertex, so a robot's path can be thought of as going off the grid for one vertex to visit the gadget. Note that vertex multiplicity outside of gadgets is indeed 2.

We realize M on the 2D grid as follows. The top row of M initially contains the job robots $x_1, y_1, \ldots, x_n, y_n$, ordered left to right, which have to move down to the bottom row of M. The rightmost column of M initially contains the checker robots $c_1^1, c_1^2, c_1^3, \ldots, c_m^1, c_m^2, c_m^3$, ordered top to bottom, which have to move left to the leftmost column of M. The pivot robot r^* is initially located near the bottom row and has to go from the leftmost column to the rightmost column of M. The robot β is initially located near the rightmost column and has to go from the bottom row to the top row of M.

We now turn to the correctness of the reduction. We have the following correspondence between a partition (B, A) for I and a solution to M_s: A job of an x_i, y_i pair is in the before-set B if and only if the corresponding job robot traverses B_i before r^* does. Now let (B', A') be a partition obtained from a solution to M_s, as just defined. Clearly, by the definition of the B_i's, B' satisfies the before-constraints of I. We now prove that after constraints are satisfied by A':

Lemma 4.3. *Let C_j be an after-constraint of I and let $R(C_j)$ denote the corresponding job robots in M_s. Then one of the robots of $R(C_j)$ traverses its respective before-constraint gadget after r^*.*

Proof. Assume for a contradiction that each robot in $R(C_j)$ traverses its respective before-constraint gadget before r^* does. Let r be the last robot of $R(C_j)$ to do so and let B_i be the gadget it traverses. We now examine the time step in which r is at B_i. At this point, r^* is located to the left of B_i while all the checker robots of C_j have already traversed the gadget A_j due to the precedence gadgets. Consequently, β must have already traversed A_j, as it cannot be the last robot to traverse the A_j, by its definition. In particular, β already traversed the precedence gadget $\mathcal{G} = \mathcal{P}(r^*, \beta)$ (bottom right in Figure 7). This is a contradiction since r^* must traverse $\mathcal{G}$ before β. □

Given a valid job partition, it is easily verified that the correspondence above directly lends itself to an ordering of the robots in M_s by which they can move to their targets one by one. We omit the rest of the proof.

5 CONCLUSION

We gave a refined complexity analysis of MRMP-GP that sheds new light on the problem's sources of difficulty. A key element of previous MRMP-GP hardness constructions is paths that traverse the same set of vertices in opposite directions (e.g., a given path would contain the sequence v_1, v_2, v_3 while another path would contain v_3, v_2, v_1). We show that hardness can arise even without such paths if instead we have a different element, which is blocking targets. This observation leads to an intriguing question, which is whether the MRMP-GP remains hard when neither elements are present. A positive answer could have implications for fixed-path robot/transport systems, which could be designed to avoid the aforementioned elements.

From the perspective of parameterized complexity [8], which is a research avenue for hard motion planning problems [29], our hardness results rule out candidate parameters. Namely, by showing that hardness remains even for a constant vertex multiplicity (VM) we prove that MRMP-GP is unlikely to be fixed-parameter tractable (FPT) when parameterized by VM. The same statement holds for path shape complexity. Therefore, we guide the search for parameterized algorithms toward other parameters. We believe that our hardness constructions more vividly expose potential parameters since our results hold for highly distilled MRMP-GP formulations.

A natural extension of MRMP-GP is optimizing the solution, e.g., its makespan. Such an optimization variant is closely related to Multi-Agent Path Finding [35], where its use as a subroutine may have potential. For example, in each high-level node of the popular Conflict-Based Search algorithm [30] a path is found for each agent, which may be viewed as fixed for that node. Hence, a fast algorithm for the optimization variant of MRMP-GP might improve lower bounds for the cost of a high-level node, thus better guiding the search. We believe that our new insights for deciding feasibility provide a better foundation for such future directions.

ACKNOWLEDGMENTS

M. Abrahamsen is supported by Starting Grant 1054-00032B from the Independent Research Fund Denmark under the Sapere Aude research career program and is part of Basic Algorithms Research Copenhagen (BARC), supported by the VILLUM Foundation grant 16582. Work on this paper by T. Geft and D. Halperin has been supported in part by the Israel Science Foundation (grant no. 1736/19), by NSF/US-Israel-BSF (grant no. 2019754), by the Israel Ministry of Science and Technology (grant no. 103129), by the Blavatnik Computer Science Research Fund, and by the Yandex Machine Learning Initiative for Machine Learning at Tel Aviv University. T. Geft has also been supported by scholarships from the Shlomo Shmeltzer Institute for Smart Transportation at Tel Aviv University and The Israeli Smart Transportation Research Center (ISTRC).

REFERENCES

[1] Mikkel Abrahamsen, Tzvika Geft, Dan Halperin, and Barak Ugav. 2023. Coordination of Multiple Robots along Given Paths with Bounded Junction Complexity. *arXiv preprint arXiv:2303.00745* (2023).

[2] Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. 2015. Efficient Multi-Robot Motion Planning for Unlabeled Discs in Simple Polygons. *IEEE Trans Autom. Sci. Eng.* 12, 4 (2015), 1309–1317.

[3] Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. 2017. Intractability of Time-Optimal Multirobot Path Planning on 2D Grid Graphs with Holes. *IEEE Robotics Autom. Lett.* 2, 4 (2017), 1941–1947.

[4] Alexander Berndt, Niels van Duijkeren, Luigi Palmieri, and Tamás Keviczky. 2020. A Feedback Scheme to Reorder a Multi-Agent Execution Schedule by Persistently Optimizing a Switchable Action Dependency Graph. *CoRR* abs/2010.05254 (2020).

[5] Thomas Brocken, G. Wessel van der Heijden, Irina Kostitsyna, Lloyd E. Lo-Wong, and Remco J. A. Surtel. 2021. Multi-Robot Motion Planning of k-Colored Discs Is PSPACE-Hard. In *FUN (LIPIcs, Vol. 157)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 15:1–15:16.

[6] Michal Cáp, Jean Gregoire, and Emilio Frazzoli. 2016. Provably safe and deadlock-free execution of multi-robot plans under delaying disturbances. In *IROS*. IEEE, 5113–5118.

[7] Adem Coskun, Jason M. O'Kane, and Marco Valtorta. 2021. Deadlock-Free Online Plan Repair in Multi-robot Coordination with Disturbances. In *FLAIRS Conference*.

[8] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. 2015. *Parameterized algorithms*. Vol. 5. Springer.

[9] Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. 2019. Coordinated Motion Planning: Reconfiguring a Swarm of Labeled Robots with Bounded Stretch. *SIAM J. Comput.* 48, 6 (2019), 1727–1762.

[10] Eric Ewing, Jingyao Ren, Dhvani Kansara, Vikraman Sathiyanarayanan, and Nora Ayanian. 2022. Betweenness Centrality in Multi-Agent Path Finding. In *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 400–408.

[11] Tzvika Geft and Dan Halperin. 2022. Refined Hardness of Distance-Optimal Multi-Agent Path Finding. In *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 481–488.

[12] Tzvika Geft, Dan Halperin, and Yonatan Nakar. 2021. Tractability Frontiers in Multi-Robot Coordination and Geometric Reconfiguration. *arXiv preprint arXiv:2104.07011* (2021).

[13] Ofir Gordon, Yuval Filmus, and Oren Salzman. 2021. Revisiting the Complexity Analysis of Conflict-Based Search: New Computational Techniques and Improved Bounds. In *SOCS*. AAAI Press, 64–72.

[14] Robert A. Hearn and Erik D. Demaine. 2005. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.* 343, 1-2 (2005), 72–96.

[15] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W. Durham, and Nora Ayanian. 2019. Persistent and Robust Execution of MAPF Schedules in Warehouses. *IEEE Robotics Autom. Lett.* 4, 2 (2019), 1125–1131.

[16] John E. Hopcroft, Jacob Theodore Schwartz, and Micha Sharir. 1984. On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE-Hardness of the "Warehouseman's Problem". *The International Journal of Robotics Research* 3, 4 (1984), 76–88.

[17] Kamal Kant and Steven W Zucker. 1986. Toward efficient trajectory planning: The path-velocity decomposition. *The international journal of robotics research* 5, 3 (1986), 72–89.

[18] Mor Kaspi, Tal Raviv, and Marlin W. Ulmer. 2022. Directions for future research on urban mobility and city logistics. *Networks* 79, 3 (2022), 253–263.

[19] Daniel Kornhauser, Gary L. Miller, and Paul G. Spirakis. 1984. Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications. In *FOCS*. IEEE Computer Society, 241–250.

[20] Steven M. LaValle and Seth Hutchinson. 1996. Optimal motion planning for multiple robots having independent goals. In *ICRA*. IEEE, 2847–2852.

[21] Mark Lawley and Spyros Reveliotis. 2001. Deadlock avoidance for sequential resource allocation systems: Hard and easy cases. *International Journal of Flexible Manufacturing Systems* 13, 4 (2001), 385–404.

[22] Patrick A. O'Donnell and Tomás Lozano-Pérez. 1989. Deadlock-free and collision-free coordination of two robot manipulators. In *ICRA*. IEEE Computer Society, 484–489.

[23] Roberto Olmi, Cristian Secchi, and Cesare Fantuzzi. 2010. Coordination of multiple robots with assigned paths. *IFAC Proceedings Volumes* 43, 16 (2010), 312–317.

[24] Spyros A. Reveliotis. 2006. *Real-time management of resource allocation systems: A discrete event systems approach*. Vol. 79. Springer Science & Business Media.

[25] Spyros A. Reveliotis. 2015. Coordinating Autonomy: Sequential Resource Allocation Systems for Automation. *IEEE Robotics Autom. Mag.* 22, 2 (2015), 77–94.

[26] Spyros A. Reveliotis. 2020. On the state liveness of some classes of guidepath-based transport systems and its computational complexity. *Autom.* 113 (2020), 108777.

[27] Spyros A. Reveliotis and Tomás Masopust. 2019. Some new results on the state liveness of open guidepath-based traffic systems. In *MED*. IEEE, 398–404.

[28] Spyros A. Reveliotis and Elzbieta Roszkowska. 2010. On the Complexity of Maximally Permissive Deadlock Avoidance in Multi-Vehicle Traffic Systems. *IEEE Trans. Autom. Control.* 55, 7 (2010), 1646–1651.

[29] Oren Salzman and Roni Stern. 2020. Research Challenges and Opportunities in Multi-Agent Path Finding and Multi-Agent Pickup and Delivery Problems. In *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 1711–1715.

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

[31] Thierry Siméon, Stéphane Leroy, and Jean-Paul Laumond. 2002. Path coordination for multiple mobile robots: a resolution-complete algorithm. *IEEE Trans. Robotics Autom.* 18, 1 (2002), 42–49.

[32] Israela Solomon and Dan Halperin. 2018. Motion Planning for Multiple Unit-Ball Robots in $\mathbb{R}^d$. In *Workshop on the Algorithmic Foundations of Robotics, WAFR*. 799–816.

[33] Kiril Solovey and Dan Halperin. 2016. On the hardness of unlabeled multi-robot motion planning. *Int. J. Robotics Res.* 35, 14 (2016), 1750–1759.

[34] Kiril Solovey, Jingjin Yu, Or Zamir, and Dan Halperin. 2015. Motion Planning for Unlabeled Discs with Optimality Guarantees. In *Robotics: Science and Systems*.

[35] 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 *SOCS*. AAAI Press, 151–159.

[36] Hanlin Wang and Michael Rubenstein. 2020. Walk, Stop, Count, and Swap: Decentralized Multi-Agent Path Finding With Theoretical Guarantees. *IEEE Robotics Autom. Lett.* 5, 2 (2020), 1119–1126.

[37] Jingjin Yu. 2020. Average case constant factor time and distance optimal multi-robot path planning in well-connected environments. *Auton. Robots* 44, 3-4 (2020), 469–483.

[38] Jingjin Yu and Daniela Rus. 2014. Pebble Motion on Graphs with Rotations: Efficient Feasibility Tests and Planning Algorithms. In *Workshop on the Algorithmic Foundations of Robotics, WAFR*. 729–746.