Institute of Formal Methods in Computer Science

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Optimized Multi-Agent Resource Extraction

Manuel Kubica

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Stefan Funke
Supervisor:	Claudius Proissl, M.Sc.
Commenced:	April 1, 2022
Completed:	October 1, 2022

Kurzfassung

Ressourcengewinnung durch steuerbare Arbeitskräfte (Multi-Agent Resource Extraction) kombiniert das Sammeln von Ressourcen mit dem Finden von Pfaden für mehrere Agenten (Multi-Agent Pathfinding). Dabei werden mehrere Agenten in einem Graphen koordiniert, um Ressourcen zu entfernen und sie in einem Lager abzulegen. Die Agenten können miteinander kollidieren, und während eine Ressource nicht entfernt ist, stellt sie ein Hindernis für die Agenten dar. Dieses Problem wurde durch Echtzeit-Strategiespiele inspiriert, könnte aber auch in automatisierten Lagerhäusern oder bei Problemen mit Abholung und Lieferung Anwendung finden.

In dieser Arbeit haben wir die Komplexität dieses Problems untersucht, um sowohl eine beliebige Lösung als auch die schnellstmögliche Lösung zu finden. In diesem Zusammenhang haben wir zwei Einschränkungen für das Problem eingeführt: Beim einen sind die Startpositionen eingeschränkt, und beim anderen ist sowohl die Kapazität jedes Agenten als auch die Größe jedes Ressourcenvorkommens auf eins begrenzt.
Wir haben gezeigt, dass die Suche nach der schnellstmöglichen Lösung NP-schwer ist, selbst wenn wir beide Einschränkungen verwenden. Im Gegensatz zu anderen verwandten Problemen haben wir festgestellt, dass die Entscheidung, ob eine Lösung existiert, NP-vollständig ist. Dies ist auch dann der Fall, wenn die beiden Einschränkungen einzeln angewendet werden. Nur durch die Anwendung beider Einschränkungen konnten wir eine Teilmenge von Problemen finden, die nicht NP-schwer ist. Für diese Teilmenge von Problemen haben wir einen $O(n^5)$-Algorithmus entwickelt, der eine gültige Lösung findet. Als Teil dieses Algorithmus haben wir auch einen weiteren Algorithmus entwickelt, der entscheidet, ob ein Agent einen bestimmten Knoten erreichen kann.

Abstract

Multi-Agent Resource Extraction combines collecting resources with Multi-Agent Pathfinding. For this, multiple agents are coordinated in a graph to remove resources and deposit them into a warehouse. Agents can collide with each other and as long as a resource is not removed, it is an obstacle for the agents. This problem is inspired by real-time strategy computer games, but could also apply to automated warehouses or pickup and delivery problems.

In this work, we have studied the complexity of Multi-Agent Resource Extraction for finding a feasible solution as well as finding the fastest possible solution. In this context, we introduced two restrictions to our problem: One where the starting positions are constrained, and another where the capacity of each agent and the size of each resource deposit is limited to one.
We showed that finding the fastest possible solution is NP-hard, even if we impose both restrictions. Unlike other related problems, we concluded that deciding if a solution exists is NP-complete. This is still the case when the two restrictions are applied individually. Only by applying both, we were able to find a subset of problems that is not NP-hard.
For that subset of problems, we developed an $O(n^5)$ algorithm that finds a feasible solution. As part of this algorithm, we also developed another algorithm that decides whether an agent can reach a specific vertex.

Contents

List of Figures

Acronyms

1-PDTSP One-Commodity Pickup and Delivery Traveling Salesman Problem. 14

3DM 3-Dimensional Matching. 28

AMAPF Anonymous MAPF. 15

CBM Conflict Based Min-Cost-Flow. 15

CBS Conflict Based Search. 15

GMP1R Graph Motion Planning with 1 Robot. 14

ILP Integer Linear Programming. 16

MAPD Multi-Agent Pickup and Delivery. 14

MAPF Multi-Agent Pathfinding. 13

MARE Multi-Agent Resource Extraction. 13

PMG Pebble Motion on Graphs. 14

PMR Pebble Motion with Rotations. 14

RPP Rearranged Pebble Permutation. 19

RTS Real-Time Strategy. 13

TAPF Target-Assignment and Pathfinding. 15

TECC 2-Edge-Connected Component. 19

TSP Traveling Salesman Problem. 16

1 Introduction

This work is about a Multi-Agent Pathfinding (MAPF) variant. In MAPF the goal is to coordinate the paths of multiple agents without causing collisions between them.
Due to the wide variety of possible applications, MAPF has attracted a lot of interest over the years. Variants of MAPF range from automated warehouses to a puzzle game called 15-puzzle. Although the variants have the basic idea in common, they differ a lot in the allowed moves, their goals, and how a collision is defined.

This work is a result of looking at a MAPF problem that is rooted in Real-Time Strategy (RTS) computer games. Examples of such games are Age of Empires or Starcraft II [Bli10; Mic97]. In these games, players control their own troops and workers by giving them targets where to go. However, the agents (i.e. troops or workers) still need to find the collision-free paths to the targets on their own and thus this constitutes a MAPF problem.
One specific case where this problem is especially evident is when workers remove resources i.e. trees, crystals, or gold. The algorithms commonly used in RTS games are quite efficient when only some workers remove resources. Although when there are many agents assigned to the same target, the overall resource gain decreases because the agents start to impede each other.

Based closely on this, we modeled a simplified and discretized version that only focuses on the resource extraction called Multi-Agent Resource Extraction (MARE). Discretized means that instead of a 2D plane we only consider a graph and simplified means that only one type of resource is removed by a single group of agents.
Even though this problem seems quite specific, there are also other possible applications. For example, the same problem also applies to a fleet of forklift-like robots in a warehouse that need to move stacks of boxes to several drop-off locations. In this case, the resources are the stacks of boxes.

We defined the following research questions for the MARE problem:

- How complex is solving MARE optimally?
- How complex is solving MARE sub-optimally?
- Are there any sensible restrictions that can be made to reduce the runtime complexity?
- If such a restriction exists, is there a polynomial time algorithm?

Solving the problem optimally means that we try to find the fastest possible solution and in the case of solving it sub-optimally we are looking for any solution.
To address these questions, we first look at existing research to find related problems that can be adapted or partly used to develop our own reductions and algorithms.
Based on this, we conduct a complexity analysis of the problem to answer the first two questions. The secondary goal of the analysis is to get a better understanding of the structure of MARE and thus find a possible avenue for answering the last two questions.

1.1 Related Work

MARE is related to several problems that have been widely studied in recent years.
The problem that is most closely related to MARE is the Multi-Agent Pickup and Delivery (MAPD) problem. But if a problem instance of MARE has only one agent the problem shares more properties with the One-Commodity Pickup and Delivery Traveling Salesman Problem (1-PDTSP). There are, however, still differences that will be expanded on in later chapters.
The following sections mostly focus on these two problems and the problems that are related to them.

1.1.1 Multi-Agent Pathfinding

The problem of MAPF consists of a graph with multiple agents where each agent has its own goal location and time is divided into discretized steps. A solution requires a sequence of vertices for each agent called a path. But the solution is only valid if no two paths are in conflict and MAPF problems already differ in what type of conflicts are forbidden.
The most common conflicts are vertex conflicts that prevent two agents from occupying the same vertex at the same time, swapping conflicts where neighboring agents can not swap their positions, and following conflicts where an agent can not move to a position that was occupied by another agent in the previous time step (see Stern et al. [SSF+19] for a complete list).
Two typical combinations are preventing vertex and swapping conflicts or all three named here. Disallowing all three is also called Pebble Motion on Graphs (PMG) [KMS84] and is a well-studied special case of MAPF, which itself is a generalized version of the well-known 15-puzzle. The case where only vertex and swapping conflicts are disallowed is sometimes called Pebble Motion with Rotations (PMR) [YR15].
There also is another abstraction of PMG called Graph Motion Planning with 1 Robot (GMP1R) [PRST94]. In this case, only one agent is chosen that needs to reach a goal location and all other agents are nothing more than movable obstacles.

In case of PMG, there is an algorithm that is based on an abstraction of the problem to permutation groups [KMS84]. This results in an algorithm that solves the problem in polynomial time and that can check feasibility in linear time. Similarly, for PMR there also is an algorithm that uses permutation groups to arrive at the same result [YR15]. However, the algorithm also builds on another algorithm that originally solved the PMG problems for trees [AMPP99].

Although not directly relevant to this work, there exists a plethora of different algorithms that solve MAPF problems optimally or approximately using certain objectives. Two common criteria are sum of cost and makespan.
Sum of cost is defined as the number of moves across all robots and makespan is defined as the arrival time of the last robot.
Unlike when searching for a feasible solution, both of these criteria make the problem of MAPF NP-hard when disallowing vertex and swapping conflicts [Sur10; YL13]. Evidently, these two criteria can, in general, not be optimized at the same time [YL13].
Concerning PMG, it was shown that it is NP-hard under the sum of cost objective as well [RW90]. Unlike most MAPF variants, including MARE, PMG and PMR do not allow more than one move (or rotation in case of PMR) per time step. This makes no difference for feasibility checks or complete algorithms but it is the reason that PMG does not consider the makespan objective.

One approach for solving MAPF optimally is a two-level search algorithm called Conflict Based Search (CBS) [SSFS15]. The idea behind CBS is to search a path for each agent ignoring all others and then detect conflicts between the paths. When a conflict is found, CBS imposes a constraint to prevent the conflict and then searches for paths ignoring all other agents again but obeying the constraints. This results in the high-level algorithm searching over the sets of constraints and the low-level algorithm looking for a solution given a set of constraints.
For other algorithms and a survey on MAPF see Stern [Ste19].

1.1.2 Anonymous MAPF

Another important variant of MAPF is Anonymous MAPF (AMAPF), also called unlabeled MAPF. In contrast to classical MAPF, AMAPF does not have a fixed assignment between agents and goal locations. Instead, the agents can freely distribute themselves among the goal locations. In case of AMAPF, the similarity to MARE is that in both problems the agents are interchangeable.
This problem is optimally solvable in polynomial time for both makespan and sum of cost objectives [YL12]. But there are exceptions, e.g. when in a single move an agent can traverse multiple unoccupied vertices and not only one. If so, the problem is again NP-hard when optimizing the sum of costs [CDP06].

The algorithm by Yu et al. [YL12] solves anonymous MAPF in polynomial time by converting it into a multi-commodity flow problem. This conversion is done using a time-expanded network and then applying a network flow algorithm to find a solution.
Roughly explained, a time-expanded network is created by first fixing a number of time steps and then creating a copy of the original graph for each time step. The edges of the graph also no longer connect a copy of the graph with itself but instead connect the graph of the previous time step with the one of the next time step.
This time expanded network is further modified to accommodate the different disallowed conflicts. Finally, this construction allows the conversion of the solution for the flow back to optimal conflict-free paths for each agent.

1.1.3 Colored MAPF

Colored MAPF or also called Target-Assignment and Pathfinding (TAPF) is a combination of AMAPF and classical MAPF.
In colored MAPF, there are teams of agents and each team has goal locations that have to be reached by any robot of the corresponding team. This means that if there is only one team, the problem is equivalent to AMAPF and if each agent has a team for itself, it is the same as classical MAPF. In this case, the similarity to MARE is not directly obvious but, if some agents carry resources and some do not, these agents can be seen as two teams. The agents that carry resources constitute one team and the agents that do not constitute the other team.
Similar to classical MAPF, a feasibility check is still possible in linear time [GH10].

One approach for solving colored MAPF is to use the structure of the problem and build an algorithm by combining algorithms of the two problems it is based on. For instance, this is done by combining the CBS algorithm (see Section 1.1.1) and the network flow approach (see Section 1.1.2).
The resulting algorithm is called Conflict Based Min-Cost-Flow (CBM) [MK16].

First, the algorithm combines all agents from one team to a meta-agent using the algorithm based on network flow. Second, CBS is performed between all meta-agents to resolve conflicts between them.

1.1.4 Multi-Agent Pickup and Delivery Problem

The problem that has the most similarities to MARE is the MAPD problem.
MAPD is another variant of MAPF which again can have different conflicts but typically only disallows swapping and vertex conflicts.
In contrast to MAPF, the goal for agents is to complete different tasks where each task consists of a pickup and delivery location and a release time. Agents can freely choose one task among the released tasks and complete it by first visiting the pickup location and then the delivery location. As soon as all tasks are completed the problem is solved.
Even though MAPD is very similar to MARE, most research focuses on optimal algorithms and is therefore not directly helpful for our research questions.
One approach to find an approximate solution for MAPD is to first determine an order in which the tasks are picked up and then solve a special Traveling Salesman Problem (TSP) problem [LMLK19].

1.1.5 One-Commodity Pickup and Delivery Traveling Salesman Problem

The 1-PDTSP is based on the well-known TSP and due to this connection, it is also NP-hard to solve [HS04].
The idea of the 1-PDTSP is that there exists one depot with a single capacitated vehicle that needs to fulfill the pickups or deliveries each customer has. The vehicle thereby starts with an arbitrary amount of commodity and has to visit each customer exactly once. As soon as all customers are satisfied, it has to return to the depot.
The problem is almost identical to MARE as long as there is only one agent in the MARE instance.

One possible approach to solve this problem is to use an Integer Linear Programming (ILP) formulation and then use a branch-and-cut algorithm to find the solution to it [HS04; HS07].
For a more complete list of the 1-PDTSP and pickup and delivery problems in general see one of the existing surveys [BCGL07; PDH08a; PDH08b].

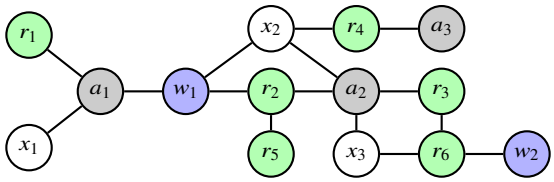
2 Preliminaries

Before continuing with the complexity analysis, we first introduce the notation for MARE and parts of the notation for PMR [YR15].
The MARE notation is rather basic and only defines a problem instance. Instead of defining a new notation for the more complex cases, we borrow parts of the PMR notation. This is possible because we only require a more complex notation for cases where there is a question about the possible moves of an agent without removing more resources. For example, we later use this notation to decide whether an agent can reach a certain vertex. Using the existing notation also allows us to use the lemmas and theorems introduced by Yu et al. [YR15] without modification.

2.1 Multi-Agent Resource Extraction

2.1.1 Basics

Each MARE instance I consists of a graph $G = (V, E)$, the carrying capacity of agents c and the size of resources d. The vertices V are further divided into three subsets W, R, and X corresponding to warehouses, resources, and holes. The holes (also called empty vertices) are the vertices that do not contain a warehouse or a resource. Additionally, the agents are defined as a separate set A with an injective function s that assigns each agent an element of X as its starting position.
Each agent starts with carrying zero resources and is never allowed to exceed c. All resources start with d resources and are obstacles as long as they still have at least one resource left. At each time step, an agent can remove/deposit a single resource from an adjacent resource/warehouse vertex but can not move while doing so.
Furthermore, each vertex can only be occupied by one agent/resource (also called vertex conflict) and adjacent agents can not swap their position in a single time step (swapping conflict). This means that agents can only move at most one vertex at a time and rotations along cycles are allowed even if the cycle is fully occupied by agents. In this work, cycles always refer to cycles with at least three vertices.
Lastly, a MARE instance is solvable if there exists a set of legal moves that result in a configuration where all resources are deposited in a warehouse.



Resources are marked green, warehouses blue, agents gray, and holes white.

Figure 2.1: Example of a MARE instance

2.1.2 Variants

Due to the wide variety of real-world problems and different requirements, there are several variants of MARE that can be considered.

The two variants Decision MARE and Optimal MARE are the most simple ones as they impose no restrictions on the instance. In case of Optimal MARE, the makespan objective minimizes the time step when the last resource is deposited into a warehouse.

Definition 1 (Decision MARE)
Instance: A MARE instance
Question: Is there a solution?

Definition 2 (Optimal MARE)
Instance: A MARE instance
Goal: If a solution exists find the optimal solution under the makespan objective.

However, Chapter 3 shows that it is important to distinguish between smaller subclasses of the problem. The first restriction is to set carrying capacity c and the resource deposit size d both to one. Such an instance is called 1-1-MARE and again both optimal and decision variants of 1-1-MARE can be considered. One case where this restriction is applicable is when looking at robots that move boxes, as mentioned in Chapter 1, and the boxes can not be stacked.

Definition 3 (1-1-MARE)
A MARE instance with $c = d = 1$.

Another reasonable modification is to restrict the starting positions of the agents. Often the agents do not start at arbitrary positions but instead start e.g. in close proximity to a single warehouse. In case of MARE, we can model this by only allowing agents to start in a shared connected component without resources and where there exists at least one warehouse. We call such an instance Restricted MARE.

Definition 4 (Restricted MARE)
A MARE instance where there exists a path between each pair of agents and a path from each agent to a warehouse. All of these paths do not contain resources.

2.2 Pebble Motion with Rotations

2.2.1 Basics

As mentioned before, the movement of agents in PMR is exactly the same as for MARE. However, in PMR there are no resources and the goal is to reach a goal configuration from a given starting configuration. The notation explained here is a slightly modified excerpt of the notation by Yu et al. [YR15] containing only the parts relevant to this work.
Each instance of PMR consists of a tuple (G, S, D) with G being the graph, S the starting configuration and D the goal configuration. A configuration is defined as a function that assigns

each agent its current vertex. E.g. if agent a is on v at S, then $S(a) = v$ and $S^{-1}(v) = a$. Furthermore, the variable p denotes the total number of agents in a given instance. A solution to such an instance is a sequence $\langle S, \ldots, D \rangle$ of configurations that does not use any illegal moves.

There are also a few more helpful functions.
First, there are the vertices of $v \rightsquigarrow w$. Unlike in Yu et al. [YR15], $v \rightsquigarrow w$ always denotes the vertices of the shortest path between v and w excluding v and w.
Lastly, $V(G)$ are the vertices, $holes(G)$ are the number of holes, and $N(G)$ is the number of vertices in a given graph or subgraph.

2.2.2 Skeleton tree

One of the most important ideas introduced by Yu et al. [YR15] is the usage of a skeleton tree (also called block-cut tree) to simplify certain PMR instances.
Before the simplification, the PMR instance is converted to an Rearranged Pebble Permutation (RPP) instance that is solvable iff the PMR instance is solvable [YR15, p.18]. In an RPP instance all vertices of 2-Edge-Connected Components (TECCs) are occupied and thus the skeleton tree can only be created if $N(TECCs) \leq p$. A TECC is a component of a graph where after removing an arbitrary edge the component is still connected.
Next, the original graph G is converted to a graph T_G by combining each TECC in G into a single vertex in T_G. Thus the result no longer contains any cycles, is a tree and only the bridges of G are left. The agents stay on their respective vertices or if they are on a TECC they are combined as well. In T_G the newly combined vertices are called composite vertices and all other degree three or higher vertices are called forks. An example can be seen in Figure 2.2.

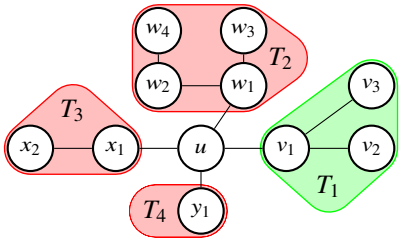
(a) RPP instance with bridges marked red

(b) Resulting skeleton tree T_G with composite vertices marked bold

Figure 2.2: RPP instance and its corresponding skeleton tree

Based on this construct, Yu et al. [YR15] were able to adapt an existing algorithm that only worked for trees [AMPP99] to solve arbitrary PMR instances in polynomial time.
The reason why this construction is so helpful is that in PMR rotating agents along a cycle allows movement independent of the number of agents in the graph. Thus with an increasing number of agents, the movement is mostly limited by the bridges in the graph and not by the TECCs.

In addition to the basic principle, we also use several helper functions introduced by Yu et al. [YR15] to label certain parts of T_G. First the function $F(u)$ returns a set of all connected components that remain after removing u from T_G. Functions T and $\overline{T}$ further divide F such that $T(u, v) \in F(u)$ and the returned tree contains v. Lastly, $\overline{T}$ contains all elements of $F(u)$ except $T(u, v)$. This can be seen in Figure 2.3.



Functions are defined as follows: $T(u, v_1) = T_1$, $\overline{T}(u, v_1) = \{T_2, T_3, T_4\}$ and $F(u) = \{T_1, T_2, T_3, T_4\}$.

Figure 2.3: Example skeleton tree for helper functions T, $\overline{T}$ and F

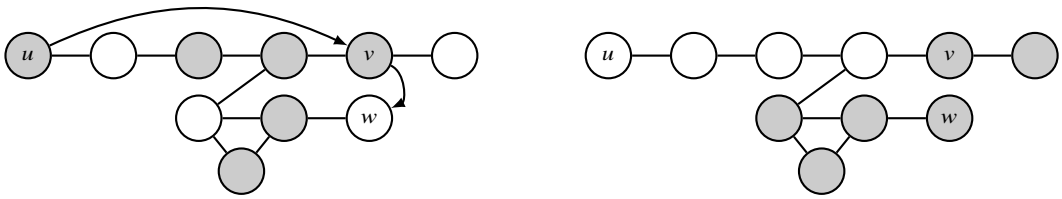
Even though the construction of skeleton trees seems quite restrictive because it is only applicable if $N(TECCs) \leq p$, all other cases are way simpler to solve. As an example RPP and thus PMR is always solvable if G is not a cycle and $N(TECCs) > p$ [YR15, p.17].

2.2.3 Equivalence between agents

Yu et al. introduced lemmas that show if or when equivalence exists between agents. Two agents are equivalent iff it is possible to swap the position of two agents without affecting others.
These lemmas are later used in Section 4.2 for a proof by contradiction by showing that two agents are equivalent and not equivalent at the same time.
The first lemma helps to identify whether two agents are equivalent even if the sequence of configurations to only swap the agents is not apparent.

Lemma 1 (Lemma 16 in Yu et al. [YR15])
Let (G, S, D) be an instance of RPP in which G is not a cycle and $N(TECCs) \leq p < n - 1$. Let u, v, and w be vertices of G such that the path between u and v and the path between v and w are not edge-disjoint. Assume u and v are occupied by agents and moves exist that take S to a new configuration in which agent $S^{-1}(u)$ is moved to v and $S^{-1}(v)$ is moved to w. Then S can be taken to a configuration S' in which S and S' are the same except agents on u and v are exchanged.

An example of configurations can be seen in Figure 2.4. In Section 4.2, we further extend this lemma to cover an even wider variety of cases. An important consequence of this lemma is that if two agents are able to swap places but affect others by doing so they are equivalent and thus can swap without affecting others.



(a) Configuration S with arrows indicating the vertices where the agents are at D

(b) Configuration D that is reachable from S and where $S^{-1}(u)$ is at v and $S^{-1}(v)$ is at w.

Agents are marked gray and the path between u and v and the path between v and w are not edge-disjoint.

Figure 2.4: Example of configurations S and D for Lemma 1

Lemma 2 also allows identifying equivalent agents but in addition, it allows to show the opposite too. It achieves this by defining 9 conditions that can be checked in amortized constant time [YR15, p.23]. However, it is only applicable if a skeleton tree can be created and there is a free path between both agents.
The only change is in condition 6 where a minor mistake of the original lemma is fixed.

Lemma 2 (Lemma 19 in Yu et al. [YR15])
Let $p_1 \coloneqq S'^{-1}(u)$, $p_2 \coloneqq S'^{-1}(v)$ for some $u, v \in V(T_G)$ such that $u \rightsquigarrow v$ contains no other agents. Then p_1, p_2 are equivalent with respect to S' if and only if at least one of the following conditions holds:

1. *There exists a fork vertex w in $u \rightsquigarrow v$ such that both $T(w, u)$, $T(w, v)$ are not full or at least one other tree of $F(w)$ is not full.*
2. *Let w be a composite vertex such that u is in $w \rightsquigarrow v$ and no other fork vertex or composite vertex is in $w \rightsquigarrow u$. There exists such a w that $T(u, w)$ has $d(w, u) + 1$ empty vertices.*
3. *Symmetric to 2 with u and v switched.*
4. *Let w be a fork vertex such that u is in $w \rightsquigarrow v$ and no other fork vertex or composite vertex is in $w \rightsquigarrow u$. There exists such a w that $T(u, w)$ has $d(w, u) + 2$ empty vertices.*
5. *Symmetric to 4 with u and v switched.*
6. *Vertex u is a fork vertex. Then at least two trees of $F(u)$ have empty vertices and there are at least two empty vertices outside $T(u, v)$.*
7. *Symmetric to 6 with u and v switched.*
8. *Vertex u is a composite vertex. Then at least one tree of $\overline{T}(u, v)$ has an empty vertex.*
9. *Symmetric to 8 with u and v switched.*

3 Computational Complexity

Next we look at the computational complexity of MARE and its variants introduced in Section 2.1.2. As part of this chapter we answer what complexity classes Optimal Restricted 1-1-MARE, Decision 1-1-MARE and Decision Restricted MARE have.
The goal is to get a deeper understanding of what influences the computational complexity of a MARE variant and maybe to find one variant that differs from the rest.

3.1 Optimal Restricted 1-1-MARE

Now we show what complexity class Optimal Restricted 1-1-MARE is a part of.
One can easily assume that this problem is NP-hard due to it being tightly related to several well-known NP-hard problems (e.g. makespan MAPF). The proof provided here will confirm this assumption by reducing the hamiltonian path problem to Optimal Restricted 1-1-MARE.
The reduction will also show that this problem is NP-hard when only considering a single agent. But even this result can be expected due to single-agent MARE being similar to 1-PDTSP.

Theorem 1
Optimal Restricted 1-1-MARE is NP-hard.

Definition 5 (Hamiltonian path problem)
Instance: Undirected connected graph $G = (V, E)$
Question: Is there a path that visits each vertex exactly once?

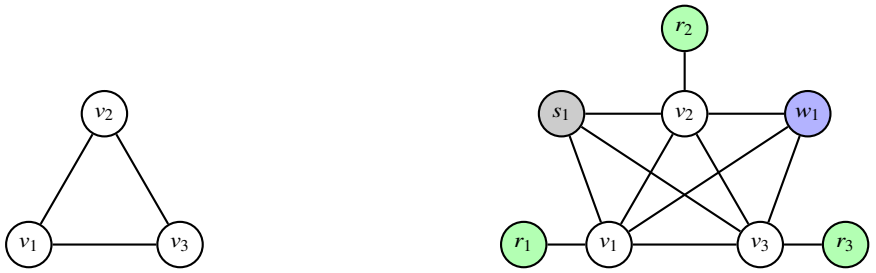
The hamiltonian path problem for undirected graphs is well-known to be NP-complete [Wes+01, p.503]. Additionally, the reduction provided here requires the graph to be connected. This does not affect the NP-completeness because unconnected graphs never have a hamiltonian path and a check of whether a graph is unconnected can be performed in linear time.

Construction of 1-1-MARE instance

The first step of the reduction is to construct a Restricted 1-1-MARE instance for a given undirected connected graph $G = (V, E)$ of a hamiltonian path problem:

1. Create a copy of G called G'.
2. For each vertex $v_i \in V$ add a resource vertex $r_i \in R$ that is connected to v_i.
3. Add a warehouse vertex $w_1 \in W$ that is connected to all vertices V.
4. Add a vertex s_1 that is connected to all vertices V.
5. Place agent a_1 on vertex s_1.

The resulting graph G' has $2 \cdot |V| + 2$ vertices and $|E| + 3 \cdot |V|$ edges and therefore can be created in polynomial time. Figure 3.1 shows an example of such a construction.



(a) Original graph G

(b) Resulting 1-1-MARE instance with resources marked green, warehouses blue, and agents gray

Figure 3.1: 1-1-MARE instance constructed from a hamiltonian path problem

The idea behind this construction is that agent a_1 has to visit all vertices in $v_i \in V$ because each vertex v_i is connected to a unique resource. Furthermore, by connecting the warehouse w_1 to all vertices v_i, the agent can always deposit the resource without requiring additional moves. Lastly, the vertex s_1 is connected to all v_i to allow a_1 to freely chose the v_i that it visits first.

Correctness of construction

The second step is to prove the correctness of the construction.

Lemma 3
Graph G has a hamiltonian path
$\Leftrightarrow$ *The constructed Restricted 1-1-MARE instance for graph G has a solution with* $\leq 3 \cdot |V|$ *time steps.*

Proof. Assume graph G has a hamiltonian path. Then we can construct a solution for the corresponding Restricted 1-1-MARE instance as follows:

1. Let a_1 move from s_1 to the first vertex of the hamiltonian path of G and then let a_1 traverse the entire path
2. At each vertex $v_i \in V$ on this path let a_1 first remove the resource r_i and then deposit it into the warehouse w_1

This results in a solution where step 1 takes $|V|$ time steps and step 2 takes $2 \cdot |V|$ time steps consequently the solution requires $\leq 3 \cdot |V|$ time steps.

Contrarily, if graph G does not have a hamiltonian path, then agent a_1 requires more than $|V|$ to visit all vertices v_i. Combined with the $2 \cdot |V|$ required to remove and deposit the resources this results in a solution with $> 3 \cdot |V|$ time steps. □

Proof (Theorem 1). As shown above, one can construct an instance of Restricted 1-1-MARE and use the length of the optimal solution to decide whether the original graph G has a hamiltonian path. Consequently, Optimal Restricted 1-1-MARE has to be at least as hard to solve as the hamiltonian path problem. □

3.2 Decision 1-1-MARE

In this section, we address the question of whether Decision 1-1-MARE is in the same complexity class as deciding the solvability of a MAPF instance. Surprisingly, we show that this is not the case by reducing the set cover problem to Decision 1-1-MARE.

Theorem 2
Decision 1-1-MARE is NP-complete.

Definition 6 (Set cover problem)
Instance: Set U (universe), a set of subsets of U called S and a number $k \in \mathbb{N}_{\leq |S|}$.
Question: Is there a set $M \subseteq S$ with $|M| \leq k$ and $\bigcup_{N \in M} N = U$?

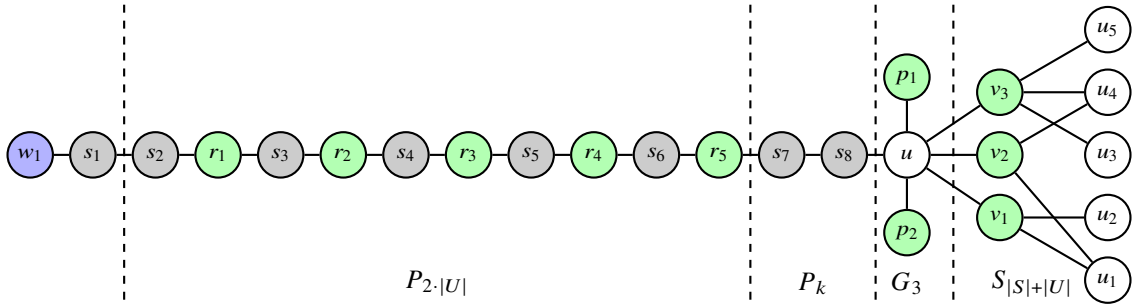
The set cover problem is another well-known NP-complete problem [Kar72].

Construction of 1-1-MARE instance

The first step is to convert a given set cover problem into a corresponding 1-1-MARE instance:

1. Create a bipartite graph $S_{|S|+|U|}$ representing the set cover problem where $u_1, \ldots, u_{|U|}$ are holes representing the elements of U and $v_1, \ldots, v_{|S|}$ are resources representing the elements of S.
2. Add another subgraph G_3 with a vertex u which is connected to two resources p_1 and p_2. Connect u to all vertices v_i of $S_{|S|+|U|}$.
3. Add a path of k agents P_k and connect one end to vertex u.
4. Add another path with $2 \cdot |U|$ vertices where the vertices alternate between agents and resources and connect the end with a resource to the other end of P_k.
5. Add a vertex with the agent a_1 followed by a vertex with a warehouse to the unconnected end of $P_{2 \cdot |U|}$.

This results in a graph G' with $3 \cdot |U| + |S| + k + 5$ vertices and $2 \cdot |U| + |S| + k + 4 + \sum_{T \in S} |T|$ edges and therefore the 1-1-MARE instance can be constructed in polynomial time.
Figure 3.2 shows an example of such an instance.



1-1-MARE instance resulting from a set cover problem with $S = \{\{1,2\},\{1,4\},\{3,4,5\}\}$, $U = \{1,2,3,4,5\}$ and $k = 2$. The vertices $u_1, \ldots, u_5$ represent the elements of U and v_1, v_2, v_3 represent S.
Subgraphs are marked by dashed lines. Resources are again marked green, warehouses blue, and agents gray.

Figure 3.2: 1-1-MARE instance constructed from a set cover problem

The basic idea of using a set cover problem encoded as a bipartite graph for a reduction of a MAPF variant is inspired by Calinescu et al. [CDP06].
An important part of the construct is the agent a_1 with its start position s_1. Agent a_1 needs to first reach vertex u for other agents to deposit their resource and thus forces all other agents into the bipartite graph without depositing their resource.
Resulting from this the k agents in the subgraph P_k need to decide which resources of $v_1, \ldots, v_{|S|}$ they remove to make space for the agents that start in the subgraph $P_{2\cdot|U|}$. The agents of subgraph $P_{2\cdot|U|}$ need to remove the resources in front of them to clear the path for agent a_1 and therefore can not remove any of the resources $v_1, \ldots, v_{|S|}$.
This leads to a solution of the set cover problem because now only the k agents in subgraph P_k can remove resources to make space for all agents in $P_{2\cdot|U|}$.

Correctness of construction

The second step is to prove that the construction is correct.

Lemma 4
The constructed 1-1-MARE instance is solvable ⇒ There exists a time step t where a_1 is at vertex u

Proof. Assume there does not exist such a time step t. Then agent a_1 is never at vertex u and therefore a_1 only visits vertices s_1, $P_{2\cdot|U|}$ or P_k. This however means that no other agent can pass a_1 to deposit resources at w_1 and therefore the constructed 1-1-MARE instance is not solvable. □

Lemma 5
Let $\hat{t}$ be the minimal t as defined in Lemma 4.
At $\hat{t}$ no agent has deposited resources and all agents are at G_3 or $S_{|S|+|U|}$ and agent a_1 has not removed any resources.

Proof. No agent could pass agent a_1 before time step $\hat{t}$. Therefore all agents except a_1 could not deposit resources yet and these agents must be in the subgraphs G_3 or $S_{|S|+|U|}$.
Agent a_1 is at vertex u for the first time therefore a_1 was not able to remove any resources. □

Lemma 6
At timestep $\hat{t}$ there are at most k resources removed in subgraph G_3 and $S_{|S|+|U|}$.

Proof. As shown by Lemma 5, at $\hat{t}$ agent a_1 has not removed any resources and no agent has deposited resources. This leaves $|U| + k$ agents where no agent can remove more than one resource. But there are $|U|$ resources in subgraph $P_{2\cdot|U|}$ that need to be removed before a_1 can reach vertex u. This leads to a maximum of k resources that could be removed in G_3 and $S_{|S|+|U|}$. □

Lemma 7
There exists a subset S that solves the set cover problem
⇔ The constructed 1-1-MARE instance is solvable

Proof. Assume the constructed 1-1-MARE instance is solvable. If there is a solution to the 1-1-MARE instance, the three lemmas above show that there exists a time step $\hat{t}$ where $|U| + k + 1$ agents are in G_3 and $S_{|S|+|U|}$ with at most k removed resources.
Due to there being only $|U| + 1$ holes in G_3 and $S_{|S|+|U|}$, this means that there are exactly k resources removed and all vertices are occupied in these two subgraphs.
Hence, all vertices $u_1, \ldots, u_{|U|}$ are reachable from u using a selection of removed resources. Because the size of the selection is $\leq k$, this forms a subset M which is a solution to the underlying set cover problem.

Assume there exists a subset M that solves the set cover problem. Without loss of generality we assume that $|M| = k$. This is allowed because if the solution has $|M| < k$ it can be converted to a solution with exactly k elements by adding unnecessary elements of S to M.
If so, one can construct a solution to the corresponding 1-1-MARE instance as follows:

1. Take the agent in P_k that is closest to vertex u and move the agent to u.
2. Let the agent at u remove a resource at vertex v_i corresponding to an element of M.
3. Move the agent at u to a newly reachable vertex u_j or if there is none to v_i.
4. Repeat steps 1-3 until there are no more agents in P_k.
5. Let the agents in $P_{2\cdot|U|}$ remove all resources that are adjacent to them.
6. Move the agents in $P_{2\cdot|U|}$ onto the empty vertices of $S_{|S|+|U|}$.
7. Let agent a_1 remove all remaining resources in G_3 and $S_{|S|+|U|}$ by moving back and forth between s_1 and u.
8. Move a_1 to vertex p_1.
9. Choose an agent on a vertex v_i and let the agent first deposit its resource and then move to p_2.
10. Take all agents on vertices u_j that are adjacent to v_i and first deposit their resource and then move back to their position.
11. Move the agent at p_2 back to vertex v_i.
12. Repeat steps 9-11 until all agents have deposited their resources.

This results in a solution that solves the corresponding 1-1-MARE instance. □

Proof (Theorem 2). As shown above, one can construct an instance of 1-1-MARE for a given set cover problem in polynomial time. Lemma 7 shows that this construction is also correct and therefore Decision 1-1-MARE is NP-hard.

For NP-completeness one still needs to show that Decision 1-1-MARE is in the complexity class of NP. This is the case if a given solution is verifiable in polynomial time.
This is the case because one could simulate it and check if all moves are valid. □

3.3 Decision Restricted MARE

Next we answer what complexity class Decision Restricted MARE is a part of.
In this case, the reduction introduced in Section 3.2 is not applicable without modification because it requires a path of agents alternating with resources. Thus this section adapts the reduction to also work for Decision Restricted MARE.

Definition 7 (3-Dimensional Matching (3DM))
Instance: Disjoint sets U, V, W of equal cardinality and $T \subseteq U \times V \times W$.
Question: Is there a subset M of T with $|M| = |U|$ such that for distinct $(u, v, w), (u', v', w') \in M$ one has $u \neq u'$, $v \neq v'$ and $w \neq w'$?

The definition of 3DM is adopted from Korte et al. and is another NP-complete problem [KVKV11, p.405]. Instead of using a general set cover problem we use the more restrictive 3DM problem. In contrast to the general set cover problem 3DM has two additional properties that are required for the new reduction to work. It is an exact cover problem and all subsets in T have exactly three elements.

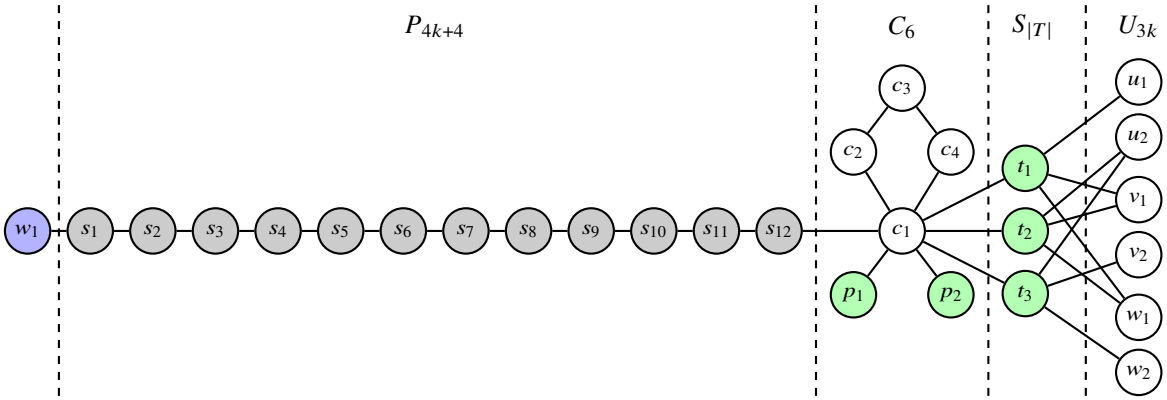
Theorem 3
Decision Restricted MARE is NP-complete.

The first step is to construct a Decision Restricted MARE instance based on a 3DM instance. This instance is very similar to the constructed instance in Section 3.2.
Summarized the differences are as follows:

1. The queue of agents/resources does only contain agents and is called P_{4k+4}.
2. Instead of a single vertex u it has a cycle consisting of four vertices that is part of C_6.
3. The reduction uses the 3DM problem instead of the set cover problem.

The remaining construction is the same as before and it is obvious that this can again be done in polynomial time. An example of such a construction with the naming of the subgraphs can be seen in Figure 3.3.
However, the following reduction only shows the NP-completeness for $d = 4$ and $c = 1$ and can also be adapted without significant change to work when $d = 4k$ and $c = k$ with $k \in N$. This means that there still might be a subset of problems in Decision Restricted MARE that is not NP-complete.

In regards to the basic idea of the reduction, the agent a_1 at s_1 again pushes all other agents into the bipartite graph by not allowing other agents to pass before it reaches c_1. The difference is how it is ensured that the agents only fit into the bipartite graph if they found a solution to the underlying 3DM instance.
The basic idea is that four agents are required to remove a resource and thus if a tree was removed it is only possible to remove another tree if another four agents can move up. This means that four agents always have to remove a resource that gives access to an additional three holes otherwise not enough agents can move up.



Decision MARE instance resulting from a 3DM instance with $T = \{(u_1, v_1, w_1), (u_2, v_1, w_1), (u_2, v_2, w_2)\}$, $U = \{u_1, u_2\}$, $V = \{v_1, v_2\}$ and $W = \{w_1, w_2\}$.
Subgraphs are marked by dashed lines. Resources are again marked green, warehouses blue and agents gray.

Figure 3.3: Decision MARE instance constructed from a 3DM instance

Proof (Theorem 3). First assume that the constructed Decision Restricted MARE instance is solvable. If so, then there again exists a minimal time step $\hat{t}$ where a_1 is at s_{4k+4} and is about to move to c_1.
First, we again determine the maximal amount of resources that are removed at $\hat{t}$. This time there are $4k + 3$ agents in the three rightmost subgraphs (C_6, $S_{|T|}$ and U_{3k}) combined. As a result, there are at most k resources removed. Thus the maximal amount of removed resources is k and the maximal amount of holes in the three rightmost subgraphs is $4 + k + 3k = 4k + 4$.
Because at $\hat{t}$ the vertex c_1 is empty, all other holes in the three rightmost subgraphs have to be occupied. This means that there are exactly k resources removed to make $3k$ vertices accessible in U_{3k}. Thus p_1 and p_2 are not removed and the selection of removed resources in $S_{|T|}$ is a solution to the underlying 3DM instance.

Lastly, we show that if the underlying 3DM instance is solvable, then the constructed Decision Restricted MARE is also solvable. For this we construct a solution based on the 3DM solution M as follows:

1. Move four agents from P_{4k+4} into C_6.
2. Let the four agents remove a resource in $S_{|T|}$ that corresponds to an element of M.
3. Move the four agents into the four newly reachable holes.
4. Repeat steps 1-3 until there are only four agents left in P_{4k+4}.
5. Move the last four agents into C_6.
6. Let agent on c_1 remove and deposit resources on p_1, p_2 and the remaining resources in $S_{|T|}$.
7. Park that agent on p_1.
8. Deposit the remaining resources by using p_2.

Thus the constructed Decision Restricted MARE is solvable iff the underlying 3DM instance is solvable. Combined with the fact that Decision Restricted MARE is also verifiable in polynomial time this means that it is indeed NP-complete. □

3.4 MARE and 2-Edge-Connectivity

Even though Theorem 2 shows that Decision 1-1-MARE and therefore Decision MARE is in general NP-hard, this chapter introduces a special case where this is not the case and a corresponding lemma.

The connectivity of a graph or its subgraphs is often used in research on MAPF [KMS84; Sur09; YR15]. Due to a great number of real-world environments being highly connected, the restriction to 2-edge-connected graphs can be useful.
In case of MARE, this restriction in combination with a restriction on the starting positions of agents leads to a subset of MARE problems that are always solvable.
For this, we first introduce a simple lemma using the notation of PMR.

Theorem 4
Given a 2-edge-connected graph $G = (V, E)$ of a PMR instance.
An agent on vertex $v_i \in V$ can move to any vertex $v_j \in V$.

Proof. Due to Menger's theorem, there exist two edge-disjoint paths from v_i to v_j [Wes+01, p.168]. Based on these two paths, one can construct a chain of cycles where consecutive cycles only have one common vertex. Consecutive cycles share the same vertices that the two paths do. An example can be seen in Figure 3.4.
Because PMR allows for rotation along fully occupied cycles, it is always possible to rotate the agent on vertex v_i along the first cycle of the chain until it reaches the common vertex between the first and the second cycle. This can be repeated until the agent is in the last cycle of the chain and can then be rotated to the vertex v_j. □

(a) Graph with two edge-disjoint paths from v_1 to v_4 marked green and blue

(b) Resulting chain of two cycles marked red and black

Figure 3.4: Example of cycles based on two edge-disjoint paths

Corollary 1
Given a MARE instance, let G' be the subgraph induced by the vertices that are reachable by an agent without removing resources. Additionally, all components of G' are adjacent to at least one warehouse. If all components of G' are 2-edge-connected, then the MARE instance is solvable.

Proof. By definition, each component of G' has vertex v_w that is adjacent to a warehouse. Due to Theorem 4 in each component of G', a single agent can freely move. This allows that agent to sequentially remove all resources that are reachable without passing another component of G' by moving between the resources and the vertex v_w.
By repeating this for all components of G', one can remove and deposit all resources. □

4 Algorithm for Restricted 1-1-MARE

Before continuing, let us first recap what we learned from the last chapter. Both 1-1-MARE and Restricted MARE are NP-hard when trying to find a feasible solution.
Assume we want to prove that the combination of the two (Decision Restricted 1-1-MARE) is also NP-complete. In this case, the reduction used for Decision 1-1-MARE is no longer applicable because a key component is the queue of alternating agents and resources.
This leaves the idea to use a reduction similar to Decision Restricted MARE. The key to such a reduction is that if the agents choose the wrong resource and do not get the needed amount of new holes, the instance is no longer solvable. However, in Decision 1-1-MARE even this is not possible because the one new hole created by removing any resource is enough to prevent the instance from becoming unsolvable.
Because both approaches for a reduction do not work, it is conceivable that Restricted 1-1-MARE is not NP-hard.

Algorithm 4.1 Algorithm for Restricted 1-1-MARE

```
1:  function CalculateFeasibleSolution(I)          // I is the Restricted 1-1-MARE instance
2:      while resource exists ∨ agent carrying resources exists do
3:          changed ← false
4:          for all all agents a carrying resources do          // try to deposit resources
5:              for all all vertices v adjacent to warehouse do
6:                  if a can reach v then
7:                      move a to v and deposit resource
8:                      changed ← true
9:                  end if
10:             end for
11:         end for
12:         for all all agents a not carrying resources do      // try to remove resources
13:             for all all vertices v adjacent to resource do
14:                 if a can reach v then
15:                     move a to v and remove an adjacent resource
16:                     changed ← true
17:                 end if
18:             end for
19:         end for
20:         if not changed then
21:             return unsolvable
22:         end if
23:     end while
24: end function
```

In essence Algorithm 4.1 is a greedy algorithm that just tries to remove and deposit the resources in an arbitrary order. It returns unsolvable if it reaches a state where no more resources can be removed or deposited.
The algorithm could be applied to 1-1-MARE or Restricted MARE without change and thus it is not clear why it works for Restricted 1-1-MARE.

4.1 Correctness and Time Complexity

This section will prove that the algorithm indeed works for Restricted 1-1-MARE and that it requires $O(n^5)$ time steps. For this, we first introduce two lemmas that are helpful to show the correctness and then prove the theorem. The main problem for this section is to show that if the algorithm returns unsolvable the problem is indeed unsolvable.

Theorem 5
Algorithm 4.1 finds a feasible solution to a Restricted 1-1-MARE instance I in $O(n^5)$ time steps.

Lemma 8
Let I be a 1-1-MARE instance with its initial state S and moves exist that take S to a state D. Let M be a subset of agents that removed a resource missing at D.
Then, without removing/depositing more resources, there exist moves that take D to a new state D' where each agent $a \in M$ is on a former resource vertex it removed. All other agents are on their starting positions.

Proof. We construct D' as follows. Because there exist moves that took S to D, we can use these moves in reverse (replacing moves that previously deposited or removed resources with waiting). When using the moves in reverse, there are states where an agent of M has previously removed a resource and we want the agent to be at the vertex of the resource at D'. Instead of replacing the action of removing with waiting, we can move the agent onto the removed resource and for all further moves pretend that the agent is a resource by no longer moving it.
Applying these modified moves to D results in the desired state D'. □

Lemma 9
Let I be a Restricted 1-1-MARE instance with its initial state S. Two separate sequences of moves exist that take S to S_u or S_s respectively. All resources removed at S_s are also removed at S_u.
If there exists a resource r, that is removed in both S_u and S_s by agents a_u and a_s respectively, then if a_s can reach a vertex v, a_u can reach v as well.

Proof. Assume a_u can not reach a vertex v, but a_s can.
Using Lemma 8 we can create two states, one where a_u is on the vertex of r, and another where the vertex is occupied by a_s instead. In both states, all other agents are on their starting positions. Because I is restricted, it is possible to modify both instances such that the exact same vertices are occupied without moving the respective agent away from the vertex of r. As a result, all vertices reachable by a_s at S_s are also reachable by a_u at S_u.
This is a contradiction because now a_u can reach and not reach v at the same time. □

Proof (Theorem 5). First show that the Algorithm 4.1 terminates.
This is the case because in each iteration of the while-loop there is either a resource removed or deposited or if not, the algorithm returns unsolvable. Therefore, the algorithm has to terminate due to the finite amount of resources.

For the time complexity, assume that reachability can be decided in $O(n^2)$ and a corresponding plan can be created in $O(n^3)$. If so, it is evident that the algorithm requires $O(n^5)$. The assumption will be shown as part of Section 4.2.

If the algorithm finds a solution, it is obvious that I is solvable. Hence all that remains to show is that if Algorithm 4.1 returns unsolvable, the instance I is indeed unsolvable. Assume the algorithm returns unsolvable and a solution for I exists.
Let S_u be the state of I when the algorithm returns unsolvable. If at S_u all resources are already removed, let S_s be the final state of the solution to I. If not, let S_s be the first state in the solution right before the agent a_s removes a resource that is not removed at S_u.
As a result, all resources removed at S_s are also removed in S_u.

Next we try to modify S_u and S_s to create new states S'_u and S'_s without removing or depositing additional resources such that each hole at S'_s is also a hole at S'_u and that at S'_u there is an empty agent called a_u on vertex $S'_s(a_s)$.
If we are able to do this, agent a_u that carries resources can reach a warehouse and we arrive at a contradiction. This is because there exist moves that take a_s, starting at S'_s, to a warehouse and thus the same moves can be used to move the empty agent a_u starting at S'_u to a warehouse. The moves can be applied to S'_u because all resources that are removed at S'_s are also removed in S'_u and each hole in S'_s is also a hole in S'_u.

The desired states S'_u and S'_s are created as follows.
First create S'_s by using Lemma 8 on S_s with the set M_s being all resources that are still carried in both instances S_u and S_s. For S'_u, we first use Lemma 8 on S_u with the set M_u being a combination of the set M_s and all agents that carry resources at S_u that are not removed at S_s.
Now each hole in S'_s is also a hole S'_u because the only additional elements of M_u are agents that end up on resources that are not removed at S'_u.

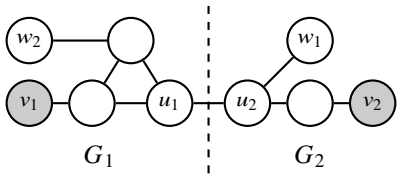
However, for the contradiction to work, the agent a_u must be empty. For this, we first show that at S'_u there are only empty agents on the starting positions or, in other words, that M_u contains all agents that carry resources at S_u.
If this is not the case, then there is a resource that is removed in both states for which the agent that removed the resource at S_u carries it, and the agent at S_s already deposited it. But Lemma 9 says that if the agent at S_s is able to reach a warehouse, the agent at S_u can reach the warehouse as well. Thus such a pair of agents can not exist and M_u contains all agents that carry resources at S_u.
Lastly, the vertex $S'_s(a_s)$ might be empty at S'_u and not contain an empty agent. To remedy this, we move one of the empty agents, which has a free path to $S'_s(a_s)$, onto $S'_s(a_s)$. Such an agent always exists because we showed that at S'_u there are only empty agents on initial positions. □

4.2 Reachability of Vertices

As part of the algorithm, we still require to check if an agent can reach a vertex or not. For PMG (i.e. when rotations along fully occupied cycles are not allowed) this problem is already discussed as the GMP1R problem [PRST94]. However, for PMR this is not the case.
To solve this for PMR, we build upon the lemmas introduced in [YR15]. Although we solve the problem for PMR it is also applicable to MARE because both allow the same movements of agents. The following section will show that reachability can be decided in linear time and a plan can be calculated in $O(n^3)$ time steps.

Lemma 10
Let (G, S, D) be an instance of RPP in which G is not a cycle and $N(TECCs) \leq p < n - 1$. Let v_1, v_2, w_1 and w_2 be vertices of G such that the path between v_1 and w_1 and the path between v_2 and w_2 have at least one bridge edge in common. The bridge is used by the paths in opposite directions. Assume v_1 and v_2 are occupied by agents and moves exist that take S to a new configuration S' in which agent $a_1 \coloneqq S^{-1}(v_1)$ is moved to w_1 and $a_2 \coloneqq S^{-1}(v_2)$ is moved to w_2. Then a_1 and a_2 are equivalent.

Proof. Let the edge $e = \{u_1, u_2\}$ be a bridge that the two paths have in common, is used in different directions and where u_1 is closer to v_1 than u_2. The sequence $X = \langle S = S_1, S_2, \ldots, S_m = S' \rangle$ consists of the configurations that take S to S'. Because e is a bridge, removing e from G creates two components which are subgraphs of G. The resulting subgraph containing v_1 is called G_1 and the other one G_2.
Figure 4.1 shows an example for this notation.



Nodes occupied by agents are shown in gray. The dashed line separates the two subgraphs.
Agent a_1 is on v_1 and agent a_2 is on v_2.

Figure 4.1: Example where Lemma 10 is applicable

Without loss of generality assume that a_1 uses e before a_2. Call the first configuration of X where a_1 uses the edge S_i with $S_i(a_1) = u_1$ and $S_{i+1}(a_1) = u_2$. Because a_2 did not yet use e, this means that at S_{i+1} agent a_2 is still in G_2.
But because w_2 is in G_1, there has to be a configuration in X after S_i where a_2 moves along e from u_2 to u_1. Let us call such a configuration S_j with $S_j(a_2) = u_2$ and $S_{j+1}(a_2) = u_1$.
Clearly, at S_{j+1} the agent a_1 is either in G_1 or in G_2.
If a_1 is in G_2, then Lemma 1 is applicable. We define the variables of the lemma as follows: $u \coloneqq S_i(a_2)$, $v \coloneqq u_1 = S_i(a_1) = S_{j+1}(a_2)$ and $w \coloneqq S_{j+1}(a_1)$. The path u to v and the path v to w contain e because $v \in V(G_1)$ and $u, w \in V(G_2)$. Hence a_1 and a_2 are equivalent.
Contrarily, if a_1 is in G_1 at S_{j+1}, then a_1 is also in G_1 at S_j because at most one agent can switch between the two subgraphs given two consecutive configurations. This means that at S_j both agents are still in the same subgraph as they were at S. Consequently, the sequence X can be truncated to

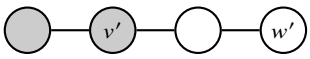
$X' = \langle S_j, \ldots, S_n \rangle$ and the argument above can be repeated for X' instead of X.
This leads to the conclusion that a_1 and a_2 are always equivalent because X is finite and in each iteration, the corresponding X' has at least one configuration less than X. □

Definition 8 (Overpass)
Let G be a graph and S a starting configuration of an RPP instance with $N(TECCs) \leq p$. An Overpass is defined as a path of vertices $v_1, \ldots, v_m$ in T_G with $m \geq 1$ where the following conditions are satisfied:

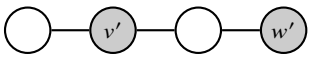
- *v_1 and v_m are arbitrary vertices.*
- *If $m > 2$ then v_2 and v_{m-1} are non-composite vertices.*
- *If $m > 4$ then $v_3, \ldots, v_{m-2}$ are non-composite vertices with a degree of two.*

Lemma 11
Let G be a graph and S a starting configuration of an RPP instance with $N(TECCs) \leq p$. Let v and w be vertices of G with v' and w' being the corresponding vertices of T_G and $a \coloneqq S^{-1}(v)$. Let $u_1, \ldots, u_m$ be a longest Overpass of length m on the path from v' to w'.
Additionally, $holes(G) \geq m - 1 \Rightarrow (holes(T(v', w')) \geq m - 1 \vee v'$ is a composite vertex).
Then a can reach w starting with S iff $holes(G) \geq m - 1$.

Before starting with the actual proof first we give the reason why the condition $holes(G) \geq m-1 \Rightarrow (holes(T(v', w')) \geq m - 1 \vee v'$ is a composite vertex) is required.
The basic statement of the lemma is that if there are enough holes in the graph then the agent can reach a vertex. However, this is not the sole criteria and the actual positions of agents do also matter as can be seen in Figure 4.2.



(a) Instance where the agent on v' can reach w'



(b) Instance where the agent on v' can not reach w'

Two graphs with agents on v' that want to reach w' and agents marked gray.

Figure 4.2: Two examples with different reachability but same the amount of holes

To circumvent this the condition ensures that if there are enough holes the lemma is only applicable when an instance is similar to Figure 4.2a. The exception to this is if v' is a composite vertex because then the holes move past the agent on v' without affecting the agent. This exception is proven as part of the following proof.

Proof (Lemma 11). Necessity is shown by looking at three cases.

Given $p = n$, there are no holes and hence the only possible moves are rotations along fully occupied cycles. Because a can reach w, the vertices v and w are part of the same TECC and thus $v' = w'$. This results in $m = 1$ and therefore $holes(G) \geq m - 1$ is satisfied.

Given $p = n - 1$, there is exactly one hole and thus $holes(G) \geq m - 1 \Leftrightarrow m \leq 2$. The proof is done by contradiction. Assume a can reach w and $m > 2$.
Because u_2 is a non-composite vertex, a can only move to u_2 when the hole is at u_2 by moving the hole to u_1. This implies that after this move $T(u_1, u_2)$ is full and a can not move to another vertex of $T(u_1, u_2)$ until the hole is back in $T(u_1, u_2)$.

However, u_2 is still a non-composite vertex and hence the only way for the hole to move to $T(u_1, u_2)$ is to move the agent a from u_2 back to u_1.
As a consequence, the hole and a can never be in $T(u_1, u_2)$ at the same time. This leads to a contradiction because for a to move from u_2 to u_3 this has to be the case.

Given $p < n - 1$. We start by proving that if w is reachable starting with S, then $holes(G) \neq m - 2$. Assume a can reach w and $holes(G) = m - 2$. Consequently $m \geq 2$ and G is not a cycle.
When the $m - 2$ holes are not on $u_2, \ldots, u_{m-1}$ then it is possible to reach a configuration S' from S by using a spanning tree of G where the holes are on $u_2, \ldots, u_{m-1}$. Clearly, w is reachable by a starting with S iff w is reachable by a starting with S'. Therefore it is possible to assume that at S all $m - 2$ holes are on the vertices $u_2, \ldots, u_{m-1}$. Furthermore, the spanning tree only moved the agents further away from $u_2, \ldots, u_{m-1}$ and thus the agent a is still in $T(u_2, u_1)$.
Let α' and β' be the agents of T_G at u_1 and u_m respectively. The vertices in $u_1 \rightsquigarrow u_m$ are empty and therefore Lemma 2 can be applied to show that α' and β' are not equivalent by disproving all nine conditions:

1. Only u_2 and u_{m-1} can be fork vertices in $u_1 \rightsquigarrow u_m$. In these cases, $T(u_2, u_m)$ or $T(u_{m-1}, u_1)$ are the only trees of their respective fork vertex that are not full. Thus the condition is always false.

2/3. Assuming w is a composite vertex and u_m is in $w \rightsquigarrow u_1$. Then w is in $\overline{T}(u_m, u_1)$ and therefore $d(w, u_1) + 1 > m - 2$. Thus the condition is always false. The same applies when u_1 and u_m are switched.

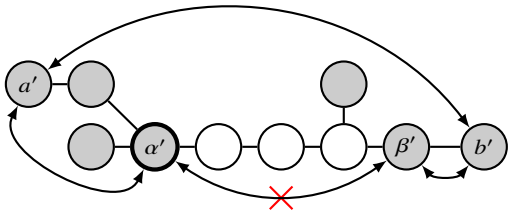
4/5. Same argument as for 2/3 only with fork vertices instead of composite vertices.

6/7. Assuming u_1 is a fork vertex then the condition is always false because $F(u_1)$ only has one tree that is not full. The same applies when u_m is a fork vertex.

8/9. Assume u_1 is a composite vertex. Because $T(u_1, u_m)$ is the only tree that is not full, the condition is always false. The same applies when u_m is a composite vertex.

The next part is to show that at S there exists an agent b of G in $T(u_{m-1}, u_m)$ which is equivalent to a. Let D be a configuration reachable from S where a is at w. At S, the tree $T(u_{m-1}, u_m)$ is full and because D exists, there has to be an agent b that is in $T(u_{m-1}, u_m)$ at S and is in $T(u_m, u_{m-1})$ at D. This means that Lemma 10 is applicable using the bridge $\{u_{m-1}, u_m\}$ and therefore a and b are equivalent.
Next take a look at a' and α'. When $a' = \alpha'$ they are equivalent and if $a' \neq \alpha'$, equivalence is shown as follows. Because of a and b being equivalent, there exists a sequence $X = \langle S = S_1, \ldots, S_k \rangle$ where S_1 and S_k are the same except that a and b are switched. In this sequence agent a moves out of $T(u_2, u_1)$ into $T(u_1, u_2)$. To make this possible an agent α in G that corresponds to α' in T_G has to move to u_2 first otherwise $T(u_2, u_1)$ would always be full and a would never be able to leave its current TECC. Call the configuration of X where α is at u_2 for the first time S_i. Using S_i and S_k, Lemma 10 is applicable and hence a and α and the corresponding a' and α' are equivalent.
Following the same argument, agents b' (the agent in T_G corresponding to b) and β' are also equivalent.
This constitutes a contradiction because now agents α' and β' are not equivalent and equivalent simultaneously. This means that if w is reachable starting with S then $holes(G) \neq m - 2$. Figure 4.3 visualizes this contradiction.

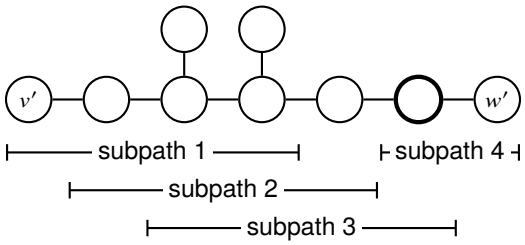
We showed that $holes(G) \neq m - 2$. Furthermore, note that an unsolvable configuration cannot become solvable by adding agents. Thus it follows that the original condition $holes(G) \geq m - 1$ is satisfied.



Example graph for T_G with the relations between the agents and agents marked gray.
A normal arrow shows that the agents are equivalent and a crossed-out arrow that they are not.

Figure 4.3: Example for the contradiction used in the proof of Lemma 11

Sufficiency is shown by constructing a plan as follows:
Start by dividing the path from v' to w' into unique subpaths where consecutive subpaths overlap. Each subpath is an Overpass that can not be further extended by adding another vertex of the path from v' to w'. Clearly, all of these subpaths are at most m long. Figure 4.4 shows an example.



A section of a graph T_G. The sections of the four subpaths are marked below.
The bold vertex is a composite vertex.

Figure 4.4: Example for a path divided into subpaths

The following section shows how agent a can successively traverse all subpaths. For this, we establish a condition when a is able to traverse a subpath and we show that this condition is still satisfied after the traversal. Also for simplicity T_G is used to explain the construction of the plan instead of G. This is possible by applying Theorem 4 each time a has to move to another vertex of a TECC.

Let $u_1^i, \ldots, u_{m_i}^i$ be the i-th subpath and agent a is at u_1^i. Furthermore, $holes(T(u_1^i, w')) \geq k_i - 1$ is satisfied or u_1^i is a composite vertex with $holes(T(u_1^i, v')) + holes(T(u_1^i, w')) \geq k_i - 1$. Let k_i be the length of the longest subpath after subpath i or subpath i itself.

The first step is to show that it is always possible to move $k_i - 1$ holes into any location of $T(u_1^i, w')$ with a being on the same vertex or TECC afterward.
When $holes(T(u_1^i, w')) \geq k_i - 1$ this is easily possible using a spanning tree of $T(u_1^i, w')$.
When u_1^i is a composite vertex there are $k_i - 1$ holes in $T(u_1^i, v')$ and $T(u_1^i, w')$ combined. Because each TECC has at least two edge-disjoint paths [Wes+01, p.168]. These two paths differ in at least one vertex x. Holes can now be moved from $T(u_1^i, v')$ to $T(u_1^i, w')$ without affecting a by moving a to x and then using the other path through the TECC to rearrange the holes.
With this knowledge, a can now cross the subpath.

If $u^i_{m_i}$ is a composite vertex or $u^i_{m_i} = w'$, then move $m_i - 1$ holes to vertices $u^i_2, \ldots, u^i_{m_i}$ and if $k_{i+1} > m_i$ move an additional $k_{i+1} - m_i$ holes so that they are in $T(u_{m_i}, w')$.
In the remaining case, $u^i_{m_i-1}$ must be a fork vertex. If so, then $u^i_{m_i-1}$ has another neighboring vertex u' that is not on the path. Move $m_i - 1$ holes to $u^i_2, \ldots, u^i_{m_i-1}, u'$ and if $k_{i+1} > m_i$ move an additional $k_{i+1} - m_i$ holes so that they are in $T(u_{m_i-1}, u_{m_i})$. Lastly, replace the computed start of the next subpath whereby $u^{i+1}_1 = u'$ instead of $u^{i+1}_1 = u^i_{m_i-2}$. In both cases, a can move along the respective holes to reach the other side.
When $u^i_{m_i} \neq w'$ and after a has passed the subpath, then u^{i+1}_1 fulfills the same condition as u^i_1 did before.

At S, when v' is a composite vertex the condition $holes(T(v', w')) \geq m - 1$ might not be satisfied. If the condition is not satisfied, then for each tree of $\overline{T}(v', w')$ its holes can be moved to $T(v', w')$ without moving a outside v' by applying the same argument as for moving holes from $T(u^i_1, v')$ to $T(u^i_1, w')$.
Now the entire condition is satisfied at S and therefore a plan can be constructed by repeating the steps until a is at w'. □

Theorem 6
Let G be a graph and S a configuration of PMR. Let a be an agent with $S(a) = v$. Deciding whether a can reach w starting with S can be done in linear time. If a can reach w, a plan can be computed in $O(n^3)$ steps.

Proof. If $N(TECCs) > p$, RPP and thus PMR are always solvable [YR15, p.17]. Therefore w is always reachable by a and feasibility can be checked in linear time. In this case, a plan can be computed in $O(n^3)$ using the PMR algorithm [YR15] by creating an arbitrary goal configuration where a is at w.

If $N(TECCs) \leq p$, assume that S is a RPP instance by calculating a corresponding RPP configuration in $O(|E|)$ [YR15]. First look at calculating a plan assuming that a can reach w.

For Lemma 11 to be applicable, the condition $holes(G) \geq m-1 \Rightarrow (holes(T(v', w')) \geq m-1 \vee v'$ is a composite vertex) must be satisfied. To achieve this in $O(n^2)$ steps, create a spanning tree of G rooted in v. The next step is to push a into a subtree until all further subtrees are full or a is in a TECC. This can be done such that the resulting configuration S' is still an RPP instance because a can stop when it is on a TECC and it is not required to create a hole in a TECC when a pushes forward. Therefore the lemma is applicable.

Next take a look at the complexity of each operation that is described in the sufficiency proof for Lemma 11.
Due to the symmetry of an Overpass the subpaths can be computed in $O(n)$. The graph T_G can be calculated in $O(|V| + |E|)$ [YR15].
In case that v is a composite vertex it is required that $holes(T(v', w')) \geq m - 1$. This can be achieved in $O(n^3)$ by moving each hole along its shortest path into $T(v', w')$ and each time moving a out of the way in $O(n^2)$.
Lastly, there is the movement of a across a subpath. Due to there being at most $n - 1$ subpaths each operation can only have a complexity of $O(n^2)$.
The redistributing of holes depends on what condition is true. When $holes(T(u^i_1, w')) \geq k_i - 1$ is satisfied, the redistribution can easily be done in $O(n^2)$ using a spanning tree of $T(u^i_1, w')$. When u^i_1 is a composite vertex, $holes(T(u^i_1, v')) + holes(T(u^i_1, w')) \geq k_i - 1$ is satisfied, a can be moved

out of the way in $O(n^2)$, and then the holes can also be rearranged using a spanning tree excluding the new vertex of a in $O(n^2)$.
Now that the holes are moved, the agent can cross the subpath in $O(n)$. This only leaves the operation where a needs to move to another vertex of a TECC which is again possible in $O(n^2)$. Consequently calculating a plan requires $O(n^3)$.

In case of feasibility, all necessary calculations only require linear time except to convert S into an S' where Lemma 11 is applicable. When only interested in feasibility, other agents are all interchangeable except a and S' can be calculated differently in $O(n)$.
For this first calculate a spanning tree rooted in v and then calculate for each root of a subtree the number of holes in itself. After this move a, without regard to other agents, until it is on a vertex that is part of a TECC or until all subtrees are full. Lastly, remove all agents passed by a and place the same number of agents on subtrees that are rooted in vertices that a passed.
Clearly, S' is still a RPP instance and a can reach w in S iff a can reach w in S'. This results in a feasibility check with a total runtime of $O(|V| + |E|)$. □

5 Conclusion and Outlook

In this work we studied the MARE problem, which is a multi-agent resource collection problem. We started by considering related work and established the similarities between MARE and PMR. Later, we exploited these similarities between the two to apply the lemmas about equivalence between agents to MARE instead of PMR.
Then, through the complexity analysis, we found answers to our first two research questions. The first question was how complex solving MARE optimally is. For this we concluded that, in this case, it inherits the property of MAPF and is NP-hard. This was done in Section 3.1 by reducing the hamiltonian path problem to Optimal Restricted 1-1-MARE. As a side product, this reduction shows that solving 1-1-MARE optimally is already NP-hard when only considering one agent.
Unexpectedly, the same is also true for the second question i.e. solving MARE sub-optimally. In this case, the reduction in Section 3.2 was based on the set cover problem by reducing it to Decision 1-1-MARE. This is in strong contrast to MAPF where this is not the case and thus we showed that MARE is more complex than traditional MAPF.
As part of Section 3.3, we were able to show NP-completeness for another variant called Decision Restricted MARE as well.

The previous two reductions were done in hopes that the restrictions to 1-1-MARE or to Restricted MARE reduce the runtime complexity. However, both did not suffice and only combining the two to Restricted 1-1-MARE was enough to reduce the runtime complexity.
Due to the findings established by PMR and the understanding of the problem structure created by the reductions, we were able to create a polynomial time algorithm for Restricted 1-1-MARE in Chapter 4 with a runtime complexity of $O(n^5)$. This also gave us the answers to the remaining research questions. Namely, it is indeed possible to find a restriction to MARE that has a polynomial time algorithm.
As a side product to the algorithm, in Section 4.2 we also answered the question of whether a vertex is reachable in PMR and concluded that this is decidable in $O(n)$ and solvable in $O(n^3)$.

In conclusion, this work solidifies how complex optimized resource collection in RTS games is because even MARE, the simplified and discretized version, is NP-hard for very restricted cases. This is further compounded by the fact that the Restricted 1-1-MARE algorithm is not applicable to RTS games because of the restriction to the carrying capacity and the resource deposit size.
In contrast to this, the algorithm is sensible for the forklift-like robots mentioned in the introduction. In this context, the algorithm could for example be used to detect and eliminate unsolvable instances before another algorithm attempts to find a better solution or the result could serve as a basis for an algorithm that does incrementally improve the solution.

There are, however, still some unanswered and some newly discovered questions that could further build upon this work. A first option would be to improve the algorithm for Restricted 1-1-MARE beyond a runtime complexity of $O(n^5)$ or even to find a tight bound for the algorithm.

Next, one could look at a continuous version of MARE to closer resemble the problem in RTS games and ask similar questions as outlined in this work. Continuous would mean using a 2D plane instead of a graph.
Another option is to explore other restrictions to MARE that might change the complexity. As an example, one could look at a MARE instance that consists of multiple connected Restricted MARE instances.
Lastly, one could further investigate how to find an approximate solution to one of the problems outlined here. This could be especially interesting for Restricted 1-1-MARE where it might even be possible to find a good bound on the approximation ratio.

Bibliography

[AMPP99] V. Auletta, A. Monti, M. Parente, P. Persiano. “A linear-time algorithm for the feasibility of pebble motion on trees”. In: *Algorithmica* 23.3 (1999), pp. 223–245 (cit. on pp. 14, 19).

[BCGL07] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, G. Laporte. “Static pickup and delivery problems: a classification scheme and survey”. In: *Top* 15.1 (2007), pp. 1–31 (cit. on p. 16).

[Bli10] Blizzard Entertainment. *StarCraft II*. 2010. URL: www.starcraft2.com/ (cit. on p. 13).

[CDP06] G. Calinescu, A. Dumitrescu, J. Pach. “Reconfigurations in graphs and grids”. In: *Latin American Symposium on Theoretical Informatics*. Springer. 2006, pp. 262–273 (cit. on pp. 15, 26).

[GH10] G. Goraly, R. Hassin. “Multi-color pebble motion on graphs”. In: *Algorithmica* 58.3 (2010), pp. 610–636 (cit. on p. 15).

[HS04] H. Hernández-Pérez, J.-J. Salazar-González. “A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery”. In: *Discrete Applied Mathematics* 145.1 (2004), pp. 126–139 (cit. on p. 16).

[HS07] H. Hernández-Pérez, J.-J. Salazar-González. “The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms”. In: *Networks: An International Journal* 50.4 (2007), pp. 258–272 (cit. on p. 16).

[Kar72] R. M. Karp. “Reducibility among combinatorial problems”. In: *Complexity of computer computations*. Plenum Press, 1972, pp. 85–103 (cit. on p. 25).

[KMS84] D. M. Kornhauser, G. Miller, P. Spirakis. “Coordinating pebble motion on graphs, the diameter of permutation groups, and applications”. MA thesis. M. I. T., Dept. of Electrical Engineering and Computer Science, 1984 (cit. on pp. 14, 30).

[KVKV11] B. H. Korte, J. Vygen, B. Korte, J. Vygen. *Combinatorial optimization*. Vol. 1. Springer, 2011 (cit. on p. 28).

[LMLK19] M. Liu, H. Ma, J. Li, S. Koenig. “Task and path planning for multi-agent pickup and delivery”. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2019 (cit. on p. 16).

[Mic97] Microsoft. *Age of Empires*. 1997. URL: https://ageofempires.com/ (cit. on p. 13).

[MK16] H. Ma, S. Koenig. “Optimal target assignment and path finding for teams of agents”. In: *arXiv preprint arXiv:1612.05693* (2016) (cit. on p. 15).

[PDH08a] S. N. Parragh, K. Doerner, R. F. Hartl. “A survey on pickup and delivery models part I: transportation between customers and depot”. In: *Journal für Betriebswirtschaft* 58.1 (2008), pp. 21–51 (cit. on p. 16).

[PDH08b] S. N. Parragh, K. F. Doerner, R. F. Hartl. “A survey on pickup and delivery models part ii: Transportation between pickup and delivery locations”. In: *Journal für Betriebswirtschaft* 58.2 (2008), pp. 81–117 (cit. on p. 16).

[PRST94] C. H. Papadimitriou, P. Raghavan, M. Sudan, H. Tamaki. “Motion planning on a graph”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE. 1994, pp. 511–520 (cit. on pp. 14, 34).

[RW90] D. Ratner, M. Warmuth. “The (n2- 1)-puzzle and related relocation problems”. In: *Journal of Symbolic Computation* 10.2 (1990), pp. 111–137 (cit. on p. 14).

[SSF+19] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. S. Kumar, et al. “Multi-agent pathfinding: Definitions, variants, and benchmarks”. In: *Twelfth Annual Symposium on Combinatorial Search*. 2019 (cit. on p. 14).

[SSFS15] G. Sharon, R. Stern, A. Felner, N. R. Sturtevant. “Conflict-based search for optimal multi-agent pathfinding”. In: *Artificial Intelligence* 219 (2015), pp. 40–66 (cit. on p. 15).

[Ste19] R. Stern. “Multi-agent path finding–an overview”. In: *Artificial Intelligence* (2019), pp. 96–115 (cit. on p. 15).

[Sur09] P. Surynek. “A novel approach to path planning for multiple robots in bi-connected graphs”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 3613–3619 (cit. on p. 30).

[Sur10] P. Surynek. “An optimization variant of multi-robot path planning is intractable”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 24. 1. 2010, pp. 1261–1263 (cit. on p. 14).

[Wes+01] D. B. West et al. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, 2001 (cit. on pp. 23, 30, 37).

[YL12] J. Yu, S. M. LaValle. “Multi-agent path planning and network flow”. In: *arXiv preprint arXiv:1204.5717* (2012) (cit. on p. 15).

[YL13] J. Yu, S. M. LaValle. “Structure and intractability of optimal multi-robot path planning on graphs”. In: *Twenty-Seventh AAAI Conference on Artificial Intelligence*. 2013 (cit. on p. 14).

[YR15] J. Yu, D. Rus. “Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms”. In: *Algorithmic foundations of robotics XI*. Springer, 2015, pp. 729–746 (cit. on pp. 14, 17–21, 30, 34, 38).

All links were last followed on September 28, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature