The Multi-Agent Transportation Problem

Pascal Bachor, Rolf-David Bergdoll, Bernhard Nebel

Albert-Ludwigs-Universität Freiburg
Freiburg im Breisgau, Germany
{bachorp, bergdolr, nebel}@cs.uni-freiburg.de

Abstract

We introduce the *multi-agent transportation* (MAT) problem, where agents have to transport containers from their starting positions to their designated goal positions. Movement takes place in a common environment where collisions between agents and between containers must be avoided. In contrast to other frameworks such as *multi-agent pathfinding* (MAPF) or *multi-agent pickup and delivery* (MAPD), the agents are allowed to separate from the containers at any time, which can reduce the makespan and also allows for plans in scenarios that are unsolvable otherwise.

We present a complexity analysis establishing the problem's NP-completeness and show how the problem can be reduced to a sequence of SAT problems when optimizing for makespan. A MAT solver is empirically evaluated with regard to varying input characteristics and movement constraints and compared to a MAPD solver that utilizes *conflict-based search* (CBS).

Introduction

In the *multi-agent pathfinding* (MAPF) problem (Stern et al. 2019), the objective is to have a number of *agents* move inside a given environment from designated start to designated goal positions while avoiding collisions. The environment is modeled using an undirected graph in which movement takes place in discrete time steps. A collision occurs whenever two agents occupy the same node at the same point in time or when they use an edge at the same time step.

A generalization of this problem is the *multi-agent pickup and delivery* (MAPD) problem (Ma et al. 2017; Liu et al. 2019), which models a warehouse logistics setting where packets have to be picked up and delivered. In addition to finding collision free trajectories, transportation tasks have to be assigned to agents and the agents have to fulfill them.

In a production logistics setting, some of the assumptions of the MAPD setting might not apply. For example, a transported payload might be so large that it is an obstacle after being delivered to its target position. This happens, e.g., when containers are used to deliver large workpieces to construction sites. Because of that, it may be necessary to temporarily move objects out of the way. In order to deal with such issues, we introduce the *multi-agent transportation* (MAT) problem. In MAT, a set of agents and a number of objects called *containers* are given. These containers shall be transported to their respective goal positions by the agents, whereby neither agents nor containers should collide with each other. In order to solve such a problem, we do not require a fixed assignment between agents and containers as in the MAPD setting.

One such scenario is depicted in Figure 1. Here, in order to get both containers C_1 (yellow) and C_2 (blue) to their respective goal positions (colored and marked accordingly), the agent A_1 has to move container C_1 out of the way first, before it can bring container C_2 to its goal position. Finally, it needs to move C_1 back to its goal position.

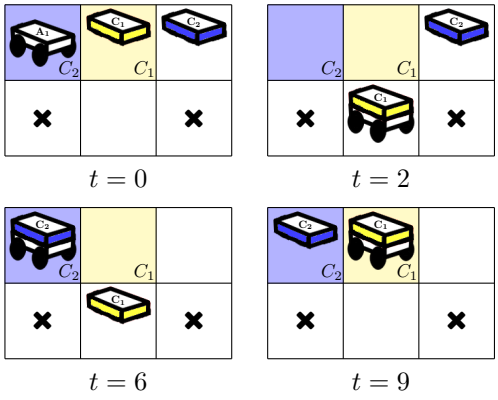

Figure 1: MAT example with a single agent A_1 and two containers C_1 and C_2. The black crosses indicate *blockades* that can never be occupied by any agent or container. We show the state at different points in time when executing the (unique optimal) plan.

MAT is a generalization of both MAPF and (offline) MAPD, the quintessential differences being that containers can collide with each other and that agents and containers are free to separate at any time. In so far, the problem has some similarity to the *package-exchange robot-routing problem* (Ma et al. 2016). However, in that framework, packages are not dropped and picked up again, but exchanged across edges. In this framework all instances can be solved.

Another problem that appears similar to MAT is the *multi-agent collective construction* problem (Lam et al. 2020). In this framework, blocks can be dropped and picked up again, but the environment is three-dimensional and there are no collisions between blocks because agents can step over existing blocks.

The rest of the paper is structured as follows. In the next section, we will formally define the MAT problem and provide some illustrative examples. We will show NP-completeness of MAT and provide an algorithm for solving it by reduction to a series of SAT problems. In doing so, we use similar reductions as have been used for solving MAPF (Surynek 2014; Surynek et al. 2016; Barták et al. 2017). We empirically evaluate the runtime of our implementation with regard to varying input characteristics, such as the number of containers or the size of the environment, and movement constraints, such as whether or not the containers block each other. Finally, we will compare it with a MAPD solver that utilizes *conflict-based search* (CBS).

Problem Definition

The multi-agent transportation problem (MAT) is formally defined as follows. Given

- an undirected, connected, simple graph (V, E),
- a set of *agents* A,
- a set of *containers* C,
- starting positions $s_0 : A \cup C \to V$, and
- goal positions $g : C \to V$,

we want to find a *legal* sequence $s_0, \ldots, s_k$ of MAT *states* such that $s_{k|C} = g$. The latter meaning that all of the containers are at their goal position in the final state.

Here, a state is any function $s_t : A \cup C \to V$ which is injective on A and injective on C. A sequence of states is considered legal if for any s_t and s_{t+1} the following constraints are met.

1. For any $a \in A$, $s_t(a) = s_{t+1}(a)$ or $(s_t(a), s_{t+1}(a)) \in E$, i.e. an agent either stays in place or moves along an edge.
2. For any $c \in C$, if $s_t(c) \neq s_{t+1}(c)$, then there exists an $a \in A$ such that $(s_t(a), s_{t+1}(a)) = (s_t(c), s_{t+1}(c))$, i.e. a container can only move together with an agent.
3. For any $a, a' \in A$ where $a \neq a'$, $(s_t(a), s_{t+1}(a)) \neq (s_{t+1}(a'), s_t(a'))$, i.e. no two agents move along the same edge at the same time.

Our constraints allow parallel and cyclic movement using previously occupied nodes. Using the terminology of Stern et al. (2019), we assume *vertex* and *swapping* conflicts, but no *following* or *cycle* conflicts.

We call a legal sequence $s_0, \ldots, s_k$ of MAT states a MAT *plan*. The length k of such plan is the plan's *makespan*. In the context of this paper, a plan is considered *optimal* if its makespan is minimal.

Examples

An example instance with two agents and two containers is shown in Figure 2. For this instance, there exists a unique optimal solution with makespan 5, which we indicate with arrows for the movement of the agents. If the containers were not allowed to be transported by multiple agents, as in MAPD, the minimal makespan would be 7.

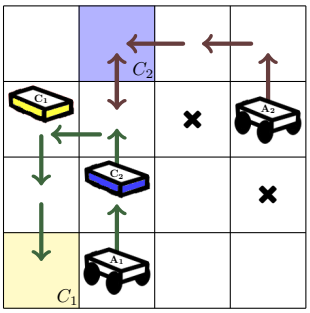

Figure 2: MAT example instance with a unique optimal solution

Moreover, there are cases in which collaboration is a must. Consider a setting as depicted in Figure 3, where the agents block each other. Here we can have a MAT plan, but no MAPD plan. The container must be handed over from one agent to the other at the node x, the 'counter'.

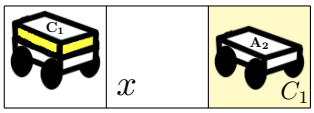

Figure 3: Over-the-counter example

Computational Complexity

MAT appears to be computationally harder than MAPF. However, we will show that MAT is nevertheless NP-complete.

In order to show NP-hardness, one can reduce MAPF with cyclic rotations, where the optimizing variant is known to be NP-hard (Surynek 2010, Theorem 1), to MAT. This can be done by viewing a MAPF agent as a pair of a MAT agent and a MAT container starting at the same position.

Lemma 1. *Deciding whether there exists a MAT plan with makespan k is NP-hard.*

Proving NP-membership is more involved. We have to show that plans can be polynomially bounded. For that purpose we will reduce MAT to a problem where so-called *pebbles* have to be moved on an undirected, connected graph, whereby only one pebble can occupy one node at a given point in time. The problem treated in the work of Kornhauser, Miller, and Spirakis (1984), the *pebble motion* (PM) problem, allows moves that take one pebble from its current

position to an adjacent unoccupied position, so-called *simple moves*. The *pebble motion with rotations* (PMR) problem (Yu and Rus 2014) additionally allows the pebbles to move simultaneously on cycles of the graph. Such moves are called *rotations*. A PMR instance is given as a triple $\langle G, S, D\rangle$, where $G = \langle V, E\rangle$ is a graph, S is the start configuration, and D is the goal configuration.

In contrast to PMR, in MAT we may not be able to rotate containers on large cycles, because we don't have enough agents that can transport the containers simultaneously. For this reason, we introduce the *pebble motion with limited rotations* (PMLR) problem. A PMLR instance is a PMR instance with an additional parameter k that specifies the largest possible rotation.

In order to establish a polynomial bound for this problem, we utilize group theory. We start by introducing the necessary definitions.

A *permutation* is a bijective function $\sigma\colon X \to X$. We say that a permutation is an m-*cycle* if it exchanges distinct elements $x_1, \ldots, x_m$ in a cyclic fashion, i.e., $\sigma(x_i) = x_{i+1}$ for $1 \leq i < m$, $\sigma(x_m) = x_1$ and $\sigma(y) = y$ for all $y \notin \{x_i\}_{i=1}^m$. Such cyclic permutation is denoted $(x_1\ x_2\ \cdots\ x_m)$.

The composition of two permutations σ and τ, denoted $\sigma\tau$, is the function mapping x to $\tau(\sigma(x))$. ϵ is the identity, which maps every element to itself, and σ^{-1} is the *inverse* of σ, i.e., $\sigma^{-1}(y) = x$ if and only if $\sigma(x) = y$. The k-fold composition of σ with itself is denoted σ^k.

We will also consider the *conjugate of* σ *by* τ, denoted σ^τ, which is defined to be $\tau^{-1}\sigma\tau$. We will use exponential notation as in the book by Mulholland (2021): $\sigma^{\alpha+\beta} := \sigma^\alpha\sigma^\beta$.

Given a set of permutations T, the *permutation group generated by* T, $\mathbf{G}$, is the closure of T under composition. Such $\mathbf{G}$ forms a group under composition with ϵ being the identity element and σ^{-1} being the inverse of a given element σ. The *diameter* of $\mathbf{G}$ then is the maximum over the number of compositions required to generate each element of $\mathbf{G}$.

In our context, two permutation groups are of particular interest. One is $\mathbf{S}_n$, the *symmetric group* over n elements, which consists of all permutations. $\mathbf{S}_n$ is generated by the set of all 2-cycles with diameter $\mathcal{O}(n^2)$. Another group is $\mathbf{A}_n$, the *alternating group* over n elements, which is the set of all permutations generated by compositions of cycles of odd length, so-called *even permutations*. A generator of this group is the set of all 3-cycles. $\mathbf{A}_n$ is a *subgroup* of $\mathbf{S}_n$, denoted $\mathbf{A}_n \leq \mathbf{S}_n$, i.e., it contains only permutations from $\mathbf{S}_n$ and is closed under composition and inverse.

A permutation group $\mathbf{G}$ is said to be k-*transitive* if for all pairs of k-tuples $(x_1, \ldots, x_k), (y_1, \ldots, y_k)$, there exists a permutation $\sigma \in \mathbf{G}$ such that $\sigma(x_i) = y_i, 1 \leq i \leq k$. In case of 1-transitivity we simply say that $\mathbf{G}$ is transitive.

Given a PMLR instance $I = \langle G, S, D, k\rangle$ and a set X of nodes of G with $|X| = p$, where p is the number of pebbles, we can transform S to S' such that in S' exactly the nodes in X are occupied by some pebble using $\mathcal{O}(n^2)$ simple moves (Kornhauser 1984, Subsection 3.1). Therefore, w.l.o.g., we can assume that in every PMLR state we consider the same nodes are occupied. Let $\mathbf{G}(I)$ be the permutation group generated by simple moves and rotations with at most k pebbles in I. Up to a polynomial factor, all possible configurations of I can be obtained with a number of operations that is limited by the diameter of $\mathbf{G}(I)$.

Lemma 2. *Let $I = \langle\langle V, E\rangle, S, D, k\rangle$ be a PMLR instance such that $\mathbf{G}(I)$ is transitive. Then $\mathbf{G}(I)$ has a polynomial diameter.*

As in the work of Kornhauser et al. (1984), we consider the *transitive substructures* that are formed by the subgraphs containing only pebbles from S that form a transitive permutation group. The subgraph contains all nodes that are reachable by this set of pebbles. The solutions for these transitive substructures forms the overall solution, if it exists.

Corollary 1. *Let $I = \langle\langle V, E\rangle, S, D, k\rangle$ be a PMLR instance. Then $\mathbf{G}(I)$ has a polynomial diameter.*

Proof of Lemma 2. Let $n = |V|$. We will consider only graphs with $n > 7$. For graphs with $n \leq 7$, the diameter is obviously constant.

If $\langle V, E\rangle$ is an n-cycle, or *cycle graph*, the diameter of $\mathbf{G}(I)$ is $O(n^2)$, if only simple moves are possible. It is $O(n)$ if the cycle can be rotated.

For the remaining cases, we proceed with a case analysis.

Case $p = n$: In this case the graph is entirely filled with pebbles. This rules out simple moves, as they require an empty node. Using transitivity, we can infer that the graph is two-edge connected and that every node is part of some cycle. If we are able to rotate all of the cycles, the lemma immediately follows from (Yu and Rus 2014, Proposition 6). We will therefore assume otherwise.

We will show that in this case, (i) one can generate a 3-cycle from polynomially many rotations and that (ii) the permutation group is 2-transitive. By (Driscoll and Furst 1983, Definition 2.6, Theorem 3.2), this implies that the diameter is polynomial.

Let us first consider a graph such as in Figure 4, which we will call *basic graph*. W.l.o.g., we will assume that the left

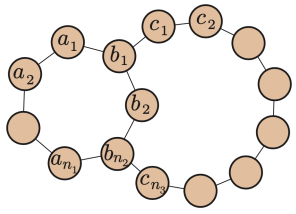

Figure 4: A basic two-edge-connected graph (Yu and Rus 2014, Figure 8)

cycle is not larger than the right one, i.e., $n_1 \leq n_3$. Further, we assume that one of the n_i's is not equal to 2. The case $n_1 = n_2 = n_3 = 2$ will be taken care of later.

The available cyclic permutations corresponding to rotations are α, β and their inverses α^{-1}, β^{-1}, where

$$\alpha = (b_1\ \cdots\ b_{n_2}\ \ a_{n_1}\ \cdots\ a_1),$$
$$\beta = (b_1\ \cdots\ b_{n_2}\ \ c_{n_3}\ \cdots\ c_1).$$

For (i) let the desired 3-cycle be

$$\sigma \coloneqq \begin{cases} \alpha, & \text{if } n_1 = 1 \\ \beta^{-1}\,\alpha\,\beta\,\alpha^{-1}, & \text{if } n_2 = 1 \\ (\beta^{-1}\,\alpha\,\beta\,\alpha^{-1})^{\alpha^{-1}(\epsilon+\beta^2)} =: \chi & \text{otherwise.} \end{cases}$$

For the first case, note that $n_1 = 1$ implies $n_2 = 2$. This follows from the fact that if $n_2 = 1$ the graph would not be two-edge connected and if $n_2 > 2$ all rotations would be possible, violating previous assumptions. For the third case, we obtain the following 3-cycle:

$$\begin{aligned} \chi &= ((a_1\ \ b_1)(b_{n_2}\ \ c_{n_3}))^{\alpha^{-1}(\epsilon+\beta^2)} \\ &= ((a_1\ \ a_2)(b_{n_2-1}\ \ c_{n_3}))^{\epsilon+\beta^2} \\ &= (a_1\ \ a_2)(b_{n_2-1}\ \ c_{n_3})(a_1\ \ a_2)(c_{n_3}\ \ c_{n_3-2}) \\ &= (b_{n_2-1}\ \ c_{n_3-2}\ \ c_{n_3}) \end{aligned}$$

and therefore

$$\sigma = \begin{cases} (a_1\ \ b_1\ \ b_{n_2}), & \text{if } n_1 = 1 \\ (a_1\ \ c_{n_3}\ \ b_1), & \text{if } n_2 = 1 \\ (b_{n_2-1}\ \ c_{n_3-2}\ \ c_{n_3}) & \text{otherwise.} \end{cases}$$

For (ii) we show that for any given elements x_1, x_2 there exists a sequence of moves such that x_1 ends up at a_1 and x_2 at b_1. Start by moving x_1 onto c_1, which is always possible because $\mathbf{G}(I)$ is transitive. If x_2 now lies on the right cycle, there exists some k such that $\beta\alpha^{-1}\beta^k$ gives a configuration as required. Otherwise, there exists some k such that $\alpha^k\beta\alpha^{-1}$ does the job.

If we can move any elements x_1, x_2 onto a_1, b_1, we can also move a_1, b_1 to arbitrary positions by using the inverse operations. By concatenating such sequences of (inverted) operations we can ultimately move arbitrary x_1, x_2 to arbitrary y_1, y_2 and $\mathbf{G}(I)$ is 2-transitive.

As mentioned, (i) and (ii) imply that the diameter is polynomial in this case. If the basic graph is embedded in any transitive graph, then our arguments from above apply to any two-edge connected graph with $n > 7$, provided it is not a cycle graph (which we covered in the beginning) or a graph made out of basic graphs such that $n_1 = n_2 = n_3 = 2$.

In the latter case, the two-edge connected graph must contain one of the subgraphs as shown in Figure 5 (note that adding anything different to the $n_1 = n_2 = n_3 = 2$ graph results in one of the cases above with one $n_i \neq 2$). A 3-cycle for each case can now be constructed as follows:

$$\sigma \coloneqq \begin{cases} \delta^{-1}\,\beta\,\delta^2\,\alpha^{-1}\,\beta\,\alpha^2 & = (x_2\ \ x_4\ \ x_7), \quad \text{(I)} \\ \zeta\,\beta^{-1}\,\zeta^{-1}\,\beta\,\alpha^2 & = (x_2\ \ x_3\ \ x_5), \quad \text{(II)} \\ \eta\,\beta^{-1}\,\eta^{-1}\,\alpha^{-1}\,\beta\,\alpha & = (x_1\ \ x_3\ \ x_7). \quad \text{(III)} \end{cases}$$

2-transitivity can be established as above, so that also in this case, we have a polynomial diameter.

Case $p < n$: If the graph is biconnected and $n > 7$, then we know by (Kornhauser, Miller, and Spirakis 1984, Theorem 1a, 1b) that any even permutation can be generated using polynomially many moves. This implies that $\mathbf{G}(I) \geq \mathbf{A}_p$. As shown in (Yu and Rus 2014, Theorem 2, last paragraph) it follows that $\mathbf{G}(I)$ has polynomial diameter.

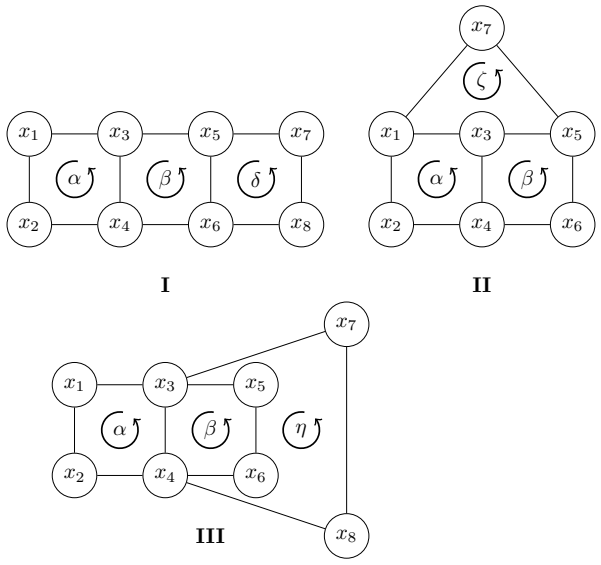

Figure 5: All relevant minimal extensions of basic graphs with $n_1 = n_2 = n_3 = 2$

If the graph is not biconnected, then it must be separable. For any subcomponent of G that is transitive w.r.t. simple moves we know by (Kornhauser, Miller, and Spirakis 1984, Theorem 1) that any permutation can be generated with polynomially many compositions. For any minimal seperable subcomponent containing a rotatable cycle we show that also any permutation can be generated with polynomially many compositions.

Consider the component as in Figure 6. As there is at least one empty node, it is possible to arrive at a configuration such that an empty node is adjacent to the cycle. Now we can move a pebble of choice out of the cycle and back in at a different position using linearly many operations. With that, any two pebbles can be swapped and therefore any permutation can be generated using polynomially many compositions.

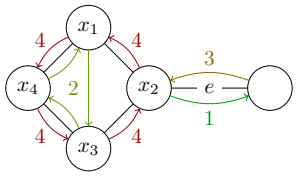

Figure 6: Swapping pebbles x_1 and x_2 (adapted from (Kornhauser, Miller, and Spirakis 1984, Figure 10))

By dividing G into partially overlapping components of these two kinds, we can compose any 2-cycle from linearly many permutations of the individual components. It follows that any $\sigma \in \mathbf{G}(I) = \mathbf{S}_p$ can be generated using polynomially many compositions. □

Using this lemma, showing NP-membership is straightforward.

Lemma 3. *Deciding whether there exists a MAT plan with makespan k is in NP.*

Proof. Membership follows if we can show that all MAT plans can be polynomially bounded. Let us assume we have n nodes in the graph and a agents.
Case $a = n$: In this case the agents block each other such that simple moves are not possible. The problem can be reduced to PMR with agents as pebbles such that the result follows from (Yu and Rus 2014, Theorem 7).
Case $a < n$: The problem can be reduced to PMLR with containers as pebbles. Corollary 1 applies and we obtain a polynomial upper bound for the movements of the containers. The additional movements of the anonymous agents has an overhead of at most $O(n)$ for each move of a container such that the total number of movements is polynomially bounded. □

From Lemma 1 and Lemma 3 we obtain the following.

Theorem 1. *Deciding whether there exists a MAT plan with makespan k is NP-complete.*

Algorithmic Solution

For a candidate makespan $T \in \mathbb{N}$, we build a propositional formula $\varphi(T)$ whose satisfying assignments constitute exactly the successful plans with makespan at most T. In order to find the optimal makespan, exponential search is employed, a method that has also been used in SAT planning (Rintanen 2014). The exponential search proceeds by first computing an upper bound for the optimal makespan by jumping to successively higher candidate makespans. The distance between the tested makespans can be adjusted via the *jumping parameter* f and an initial lower bound may be supplied, which can be obtained from *preprocessing*, as described below. Once an upper bound has been determined, binary search for the minimal T s.t. $\varphi(T)$ is satisfiable is performed. The algorithm is sketched in Algorithm 1.

Algorithm 1 Exponential search for the optimal makespan

```
procedure MAKESPAN(lower_bound=0, f=2)
    T ← lower_bound
    while ¬ SAT(φ(T)) do
        lower_bound ← T + 1
        T ← max{⌈T · f⌉, 1}
    end while
    upper_bound ← T
    return binary_search(...)
end procedure
```

The formula can be built using the propositional variables $\{\mathtt{s}_t(o) = v\}_{t\in\{0,\dots,T\},o\in A\cup C,v\in V}$ analogously to the previously defined MAT states. It is defined in a series of six axioms shown in Figure 7. Axiom (1) defines initial and final state. Axiom (2) enforces that s_t is functional. Here and later, we utilize the *AMO* (at-most-one) constraint, which states that at most one of a given set of propositional variables can be true. There exist many possible encodings of AMO (see the paper by Frisch and Giannaros (2010) for an extensive discussion) such as the *sequential* encoding from the work by Sinz (2005) that we used, which is of size linear in the number of given variables, but introduces linearly many auxiliary variables. Axiom (3) ensures that s_t is injective on A and injective on C. Axiom (4) states that objects either stay at a node or move to an adjacent node. Axiom (5) forbids that two agents use one edge at the same time, and axiom (6) finally describes the restrictions on transporting a container.

$$\bigwedge_{c\in C} \mathtt{s}_T(c) = g(c) \wedge \bigwedge_{o\in A\cup C} \mathtt{s}_0(o) = s_0(o) \tag{1}$$

$$\bigwedge_{t=0}^{T} \bigwedge_{o\in A\cup C} \mathrm{AMO}\left(\{\mathtt{s}_t(o) = v\}_{v\in V}\right) \tag{2}$$

$$\bigwedge_{t=0}^{T} \bigwedge_{v\in V} \mathrm{AMO}\left(\{\mathtt{s}_t(a) = v\}_{a\in A}\right) \wedge \mathrm{AMO}\left(\{\mathtt{s}_t(c) = v\}_{c\in C}\right) \tag{3}$$

$$\bigwedge_{t=0}^{T-1} \bigwedge_{o\in A\cup C} \bigwedge_{v\in V} (\mathtt{s}_t(o) = v \rightarrow \mathtt{s}_{t+1}(o) = v \vee \bigvee_{(v,w)\in E} \mathtt{s}_{t+1}(o) = w) \tag{4}$$

$$\bigwedge_{t=0}^{T-1} \bigwedge_{\{v,w\}\in E} \bigwedge_{\substack{a,b\in A\\ a\neq b}} \neg(\mathtt{s}_t(a) = v \wedge \mathtt{s}_{t+1}(a) = w \wedge \mathtt{s}_t(b) = w \wedge \mathtt{s}_{t+1}(b) = v) \tag{5}$$

$$\bigwedge_{t=0}^{T-1} \bigwedge_{c\in C} \bigwedge_{v\in V} \bigwedge_{(v,w)\in E} (\mathtt{s}_t(c) = v \wedge \mathtt{s}_{t+1}(c) = w \rightarrow \bigvee_{a\in A} \mathtt{s}_t(a) = v) \wedge \bigwedge_{a\in A} (\mathtt{s}_t(c) = v \wedge \mathtt{s}_{t+1}(c) = w \wedge \mathtt{s}_t(a) = v \rightarrow \mathtt{s}_{t+1}(a) = w) \tag{6}$$

Figure 7: Axioms

Preprocessing

We employ a *preprocessing* mechanism similar to the one used by Barták et al. (2017), which allows us to determine a lower bound on the effective distance $d(o, v)$ of every object $o \in A\cup C$ to every node $v \in V$. For an agent a this is just the distance between $s_0(a)$ and v in the graph. For a container c, because it cannot move by itself, $d(c, v)$ is increased by the minimum distance between $s_0(c)$ and any agent (in case $v \neq s_0(c)$). Using the fact that $t < d(o, v)$ implies $\mathtt{s}_t(o) \neq v$, we can shortcut the propositional reasoning with these implied unit clauses.

Additionally, preprocessing yields the lower bound on the makespan $\max\{d(s_0(c), g(c))\}_{c\in C}$, which can be utilized to head start the exponential search. The distances can be computed in low-order polynomial time using breadth-first-search.

Experimental Results

In order to evaluate the feasibility of our approach, we performed three sets of experiments. First, we analyzed the runtime necessary to solve varying MAT instances in order to get an idea about the scalability of our approach. Second, we compared MAT solving with solving simplifications of MAT, namely without container conflicts (assuming that packets are small) and with a fixed association between containers and agents (after target assignment). Third, we compared our SAT-based implementation with a CBS-based (Sharon et al. 2015) MAPD implementation on instances with as many agents as there are containers.

In order to run our experiments, we randomly generated quadratic grid graphs from the following parameters:

- g, the side length of the grid,
- b, the percentage of grid cells that are blockades,
- a, the number of agents,
- c, the number of containers.

All starting and goal positions are distributed among the free cells uniformly at random ensuring only that the start configuration is legal and that each container could reach its destination if we would allow the objects to pass over each other. We generated 10 such solvable problem instances for each combination $4 \leq g \leq 12$, $b \in \{10, 20\}$ and $1 \leq a, c \leq 10$, where we call a given (g, b, a, c)-tuple a *parameter set*.

We used the SAT solver *Cryptominisat* (Soos, Nohl, and Castelluccia 2009), which can be used incrementally allowing us to retain all axioms concerning previous time steps when jumping to a higher T. All tests presented in this paper were conducted using the solver's default configuration and 4 threads on Intel Xeon Gold 6242 processors. The program was allowed to run for at most 10 minutes in total per instance and to use at most 16 GB of memory. We make the source code and all test results available online[1].

Input Characteristics and Scalability

In the following we will analyze the program's runtime relative to the parameters. We aim at identifying MAT scenarios that are more or less difficult to solve for our program or, perhaps, in general.

The generated instances are quite diverse, which is also reflected in the resulting runtimes which are spread over several orders of magnitude. First of all, the size of the environment, which can vary by factors of up to 9 between $g^2 = 16$ and $g^2 = 144$, is a bad indicator for the runtime.

The runtime and solvability correlate much more strongly with the number of objects. In Figure 8 we show the mean runtimes and the percentage of unsolved instances by the number of agents and the number of containers.

It is striking that the most difficult instances including almost all of the unsolved ones are such that there are less agents than containers, i.e. $a < c$. In this case any additional agent can reduce the runtime significantly. In case $a \geq c$, however, a further increase of the number of agents seems to make the instances slightly more difficult.

[1] https://github.com/bachorp/mat

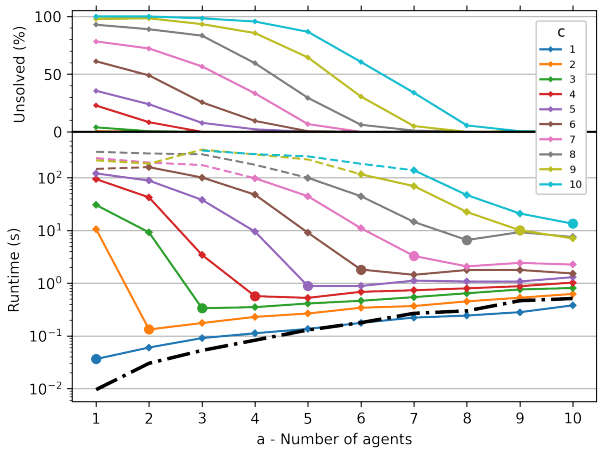

Figure 8: Effect of the number of agents a and the number of containers c on the solvability and the mean ($n = 180$) runtime of the solved instances. The lines are dashed if less than half of the instances were solved and values for which the number of agents equals the number of containers ($a = c$) are indicated with a dot. The black line denotes the mean runtime when solving MAPF instances with the respective number of agents.

We also modified our solver to solve MAPF. Note that a MAT instance with $a = c$ (runtimes denoted with a fat dot in Figure 8) does not necessarily correspond to a MAPF instance (runtimes denoted with a black line in Figure 8). Most of the time, agents and containers will start at distinct positions such that there is no clear pairing of agents and containers. Finding the optimal, possibly temporary, pairings in addition to non-colliding paths comes at a cost that can clearly be seen by the big difference between the MAT and MAPF runtimes.

Comparison with MAT Variants

As mentioned above, the property that containers can collide leads to more possible conflicts and probably to higher runtime and makespan of the generated plans. This can be seen in the two scatter plots in Figure 9. Container collisions are a major source for higher makespan and higher overall runtimes. In other words, container collisions make it much harder to solve the problem.

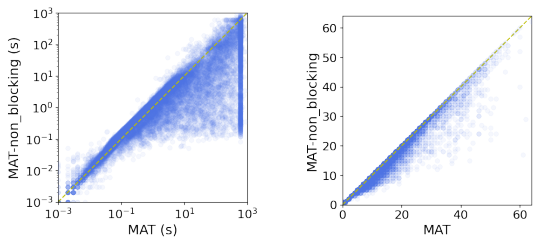

Figure 9: CPU runtime and makespan for non-blocking containers vs. MAT

Another source of difficulty might be the fact that there is no fixed assignment of agents to containers. As we have demonstrated in the beginning, this can be necessary to find a solution or allow for a shorter makespan.

We compared MAT with a variation, where a container cannot be transported by multiple distinct agents. As can be seen in Figure 10, for the CPU time, there is no clear trend. On the other hand, there are apparently a number of instances that allow for a shorter makespan if the agents are allowed to cooperate in the transportation of the agents.

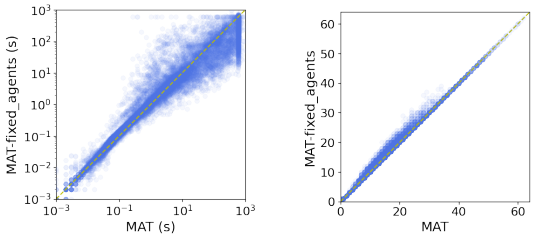

Figure 10: CPU runtime and makespan for MAT with a fixed agent per container vs. MAT

Comparison with a CBS-based MAPD Solver

Although we would have loved to compare our MAT implementation with state-of-the-art implementations of a CBS-based MAPD solver, we were not successful in identifying a solver that exactly fits our setting and that optimizes for makespan.

We therefore opted to adapt *conflict-based search with optimal task assignment* (CBS-TA) (Hönig et al. 2018) to our definition of optimizing MAPD. To do so, we (1) enhanced the low level search to search consecutive paths from an agent to an container's start and from that container's start to its goal and (2) modified the algorithm to optimize for makespan. To find makespan-optimal task assignments, we checked for each possible makespan c whether there exists a task matching consisting only of assignments with a cost lesser or equal to c using the Hopcroft-Karp algorithm (Hopcroft and Karp 1973). The remainder of the required modifications is straightforward. The theoretical properties of CBS-TA, i.e. completeness and optimality under the respective objective function, still apply and can be proven analogously.

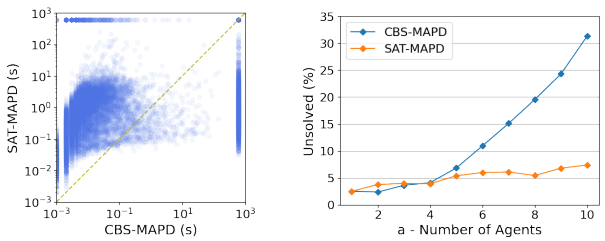

Figure 11: CPU runtime and success rate for CBS-MAPD vs. SAT-MAPD

We compared this version of CBS-TA, which we call CBS-MAPD, with a MAPD variant of MAT, which we call SAT-MAPD, where assignments are fixed and containers are non-blocking, which is equivalent to the problem solved by CBS-MAPD. As can be seen in Figure 11, CBS-MAPD shows much faster runtimes in most cases. This can be attributed to the search-based nature of the approach, which works especially fast on relatively easy instances. However, overall SAT-MAPD does not fall behind in terms of success rate and even seems to gain the advantage with an increasing number of agents.

Conclusion and Outlook

We introduced MAT, the *multi-agent transportation* problem. It is a generalization of the well-known MAPF and MAPD problems. One main difference to MAPD is that the transported payloads, called containers, can collide with each other. This addresses the requirement in some applications, e.g. production logistics, that the containers are obstacles once they have been brought to their target position. The second difference is that payload assignment is not fixed, i.e., agents can separate from containers at any point in time. This is necessary to solve instances where containers have to be moved out of the way (Figure 1) or the environment is very tight (Figure 3). Further, it helps to reduce execution costs (Figure 2).

Although the problem (in its optimizing version) appears to be computationally harder than MAPF, it turns out that it is still NP-complete. We achieved this result using previous results on pebble motion problems and an analysis of the cases with limited rotations employing group theory.

We devised a solver for MAT by reducing the problem to a sequence of SAT problems similar to what has been done in the context of MAPF. As expected, the additional degrees of freedom of MAT compared to MAPF result in higher runtimes. The runtime is highest when there are significantly more containers than agents.

By comparing variants of MAT (by simply changing the axioms), we demonstrated the effect that blocking containers and the ability of agents to cooperate in the transportation of the containers have on runtime and makespan.

Finally, we explored whether CBS could potentially be more efficient than our SAT method by comparing the two approaches in a setting which is similar to MAPD.

In the future, we plan to work on optimizations of the system and to solve some open theoretical problems. Although we intended to solve MAT problems only for a small number of agents, scalability needs to be improved. One envisioned way is to design a MAT solver based on CBS. Many more techniques and optimizations that have been used for solving MAPF should be applicable to MAT as well.

On the theoretical side, we have shown that the plan length can be polynomially bounded, but we have not shown that feasibility can be decided in polynomial time. However, we are optimistic that it is possible to come up with a linear-time feasibility test, similar to the cases for the pebble motion problem on trees (Auletta et al. 1999) and PMR (Yu and Rus 2014).

Acknowledgements

Thanks to Dillon Chen for pointing out a mistake in a previous version of this paper.

References

Auletta, V.; Monti, A.; Parente, M.; and Persiano, P. 1999. A Linear-Time Algorithm for the Feasibility of Pebble Motion on Trees. *Algorithmica*, 23(3): 223–245.

Barták, R.; Zhou, N.; Stern, R.; Boyarski, E.; and Surynek, P. 2017. Modeling and Solving the Multi-agent Pathfinding Problem in Picat. In *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017*, 959–966. IEEE Computer Society.

Driscoll, J. R.; and Furst, M. L. 1983. On the Diameter of Permutation Groups. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, 152–160. ACM.

Frisch, A. M.; and Giannaros, P. A. 2010. SAT Encodings of the At-Most-k Constraint: Some Old, Some New, Some Fast, Some Slow. In *Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation*, 36.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2018. Conflict-Based Search with Optimal Task Assignment. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, 757–765. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM.

Hopcroft, J. E.; and Karp, R. M. 1973. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.*, 2(4): 225–231.

Kornhauser, D. 1984. *Coordinating Pebble Motion on Graphs, The Diameter of Permutation Groups and Applications*. Master's thesis, MIT, Cambridge.

Kornhauser, D.; Miller, G. L.; and Spirakis, P. G. 1984. Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications. In *25th Annual Symposium on Foundations of Computer Science*, 241–250. IEEE Computer Society.

Lam, E.; Stuckey, P. J.; Koenig, S.; and Kumar, T. K. S. 2020. Exact Approaches to the Multi-agent Collective Construction Problem. In *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020*, volume 12333 of *Lecture Notes in Computer Science*, 743–758. Springer.

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19*, 1152–1160. International Foundation for Autonomous Agents and Multiagent Systems.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*, 837–845. ACM.

Ma, H.; Tovey, C. A.; Sharon, G.; Kumar, T. K. S.; and Koenig, S. 2016. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI 2016*, 3166–3173. AAAI Press.

Mulholland, J. 2021. *Permutation Puzzles: A Mathematical Perspective*. Burnaby, B.C., Canada: Self Published.

Rintanen, J. 2014. Madagascar: Scalable planning with SAT. In *Proceedings of the 8th International Planning Competition (IPC-2014)*, 1–5.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, 219: 40–66.

Sinz, C. 2005. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, 827–831. Springer.

Soos, M.; Nohl, K.; and Castelluccia, C. 2009. Extending SAT Solvers to Cryptographic Problems. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, 244–257. Springer.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019*, 151–159. AAAI Press.

Surynek, P. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press.

Surynek, P. 2014. A Simple Approach to Solving Cooperative Path-Finding as Propositional Satisfiability Works Well. In *PRICAI 2014: Trends in Artificial Intelligence - 13th Pacific Rim International Conference on Artificial Intelligence*, volume 8862 of *Lecture Notes in Computer Science*, 827–833. Springer.

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2016. Efficient SAT Approach to Multi-Agent Path Finding Under the Sum of Costs Objective. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, 810–818. IOS Press.

Yu, J.; and Rus, D. 2014. Pebble Motion on Graphs with Rotations: Efficient Feasibility Tests and Planning Algorithms. In *Algorithmic Foundations of Robotics XI - Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics, WAFR 2014*, volume 107 of *Springer Tracts in Advanced Robotics*, 729–746. Springer.