Multi-Train Path Finding Revisited

Zhe Chen[1], Jiaoyang Li[2], Daniel Harabor[1], Peter J. Stuckey[1], Sven Koenig[2]

[1]Monash University, Australia
[2]University of Southern California, USA
{zhe.chen,daniel.harabor,peter.stuckey}@monash.edu, {jiaoyanl,skoenig}@usc.edu

Abstract

Multi-Train Path Finding (MTPF) is a coordination problem that asks us to plan collision-free paths for a team of moving agents, where each agent occupies a sequence of locations at any given time. MTPF is useful for planning a range of real-world vehicles, including rail trains and road convoys. MTPF is closely related to another coordination problem known as k-Robust Multi-Agent Path Finding (kR-MAPF). Although similar in principle, the performance of optimal MTPF algorithms in practice lags far behind that of optimal kR-MAPF algorithms. In this work, we revisit the connection between them and reduce the performance gap. First, we show that, in many cases, a valid kR-MAPF plan is also a valid MTPF plan, which leads to a new and faster approach for collision resolution. We also show that many recently introduced improvements for kR-MAPF, such as lower-bounding heuristics and symmetry reasoning, can be extended to MTPF. Finally, we explore a new type of pairwise symmetry specific to MTPF. Our experiments show that these improvements yield large efficiency gains for optimal MTPF.

Introduction

Train planning is an important component in the operational planning process for rail networks. In this problem, we are asked to compute collision-free paths for rolling stock vehicles that transport goods and passengers in a rail network (Lusby et al. 2011; Laurent et al. 2021). In robotics and computer games, a related problem arises, which involves planning the motions of non-holonomic agents, such as snakes or convoys (Singh, Gong, and Choset 2018; Takemori, Tanaka, and Matsuno 2018; Mokhtar et al. 2020). Here, each agent consists of several flexible segments that follow the same path as the head. Although substantially different in practice, all these problems share common features: (i) many agents are operating simultaneously; (ii) the agents have different lengths, which means that they occupy more than one location at a time; and (iii) movements must be coordinated to avoid collisions between the agents and inside the agents themselves.

Multi-Train Path Finding (MTPF) (Atzmon, Diei, and Rave 2019) is an abstract model for solving these types of problems on a grid or, in general, graph. MTPF is closely related to k-Robust Multi-Agent Path Finding (kR-MAPF) (Atzmon et al. 2018), itself an NP-hard problem, where agents occupy just one location at a time but the computed plan must remain collision-free during execution for up to k unexpected delays for any agent. Figure 1 illustrates the difference.

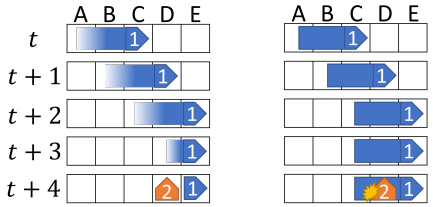


Figure 1: A kR-MAPF plan with no collisions (left) and an MTPF plan with a collision at D at timestep $t + 4$ (right). In both figures, agent 1 moves from left to right across the map. Its location is C at timestep t, and it waits for two timesteps once at E. Meanwhile, agent 2 enters D at timestep $t + 4$. The number of vertices occupied by an agent in kR-MAPF shrinks when the agent is waiting. However, an agent in MTPF always occupies multiple vertices at each timestep.

In this work, we show that kR-MAPF is a strict relaxation of MTPF. We then describe a new Conflict-Based Search (Sharon et al. 2015) strategy for solving MTPF significantly more efficiently. Our first contribution exploits the relaxation to simplify collision resolution. In particular, we show that, in many cases, agents can be replanned using a kR-MAPF model, which speeds up re-planning. Our second contribution concerns symmetric collisions between agents, which is one of the principle difficulties that make MAPF (Li et al. 2019b, 2020) and kR-MAPF (Chen et al. 2021) algorithms inefficient in practice. We generalize kR-MAPF symmetry reasoning techniques to handle agents with different lengths or robustness requirements and then apply them to prune the MTPF search space aggressively. Our last contribution explores a new kind of symmetric collisions that occur only in MTPF and shows how to resolve them more efficiently. Together, the three techniques significantly improve the efficiency of solving MTPF optimally, as we show

in a range of experiments and across a variety of domains.

Train Scheduling vs. Train Planning

Existing research on rail operations focuses on train scheduling (Yang et al. 2016; Cacchiani, Qi, and Yang 2020; Lusby et al. 2011), which includes network planning, line planning, timetable generation, train routing, and platform assignment. Our work focuses on optimal collision-free *path planning* for agents with lengths. In the literature, this problem is called Multi-Train Path Finding (Atzmon et al. 2018), although the setup applies more broadly (e.g., road convoys, snake robots, etc.). We aim to compute paths that allow train-like agents to move from start to goal without collisions. Higher-level train-planning, like line planning or network planning, is performed based on path information provided by path planning. Lower-level train-planning is more about train management and operation. Both are beyond the scope of this research.

Multi-Train Path Finding (MTPF)

We assume an operating environment that can be modeled as an undirected or directed graph $G = (V, E)$, where each vertex has up to 4 neighbors[1] (e.g., a gridmap or rail network). m agents $A = \{a_1, ..., a_m\}$ operate on the graph. Every agent a_i is assigned a start vertex $s_i \in V$, a goal vertex $g_i \in V$, and a *body length* $k_i \in \mathbb{N}$. Its total length is $k_i + 1$, accounting for its head. Time is discretised into unit-size timesteps. At each timestep, the agents move to a neighboring vertex or wait at their current vertex. Each move or wait action takes unit time.

A path π_i of agent a_i consists of a set of *occupation lists*. We denote the $k_i + 1$ vertices that a_i occupies at timestep t as $O_i(t) = [v_0, \ldots, v_{k_i}]$ and the first vertex in $O_i(t)$ as $h_i(t)$ (called the head vertex). If the head of the agent is at vertex v_0 at timestep t and moves to v_{-1} at timestep $t + 1$, then the new occupation list is $O_i(t + 1) = [v_{-1}, v_0, \ldots, v_{k_i-1}]$. If the agent waits, then $O_i(t + 1) = O_i(t)$.

Each agent's occupation list includes only its head vertex when the agent is at its start vertex. When the agent moves away from its start vertex, it grows in size to occupy $k_i + 1$ vertices eventually. This model fits rail networks where trains begin in private sliding (offmap parking) or road networks where convoys start from private parking stations. It is also possible that the initial occupation list consists of k_i+1 vertices, but this goes beyond the data provided by our set of benchmark instances.

Compared to previous models for MTPF (Atzmon, Diei, and Rave 2019) and kR-MAPF (Atzmon et al. 2018; Chen et al. 2021), which assume that all agents have the same length and robustness guarantee (respectively), our approach allows each agent a_i to have an individual value for k_i. We also consider three variants of MTPF, each one of which corresponds to a different real-world scenario of different computational difficulty:

- MTPF Variant 1: Existing research (Atzmon, Diei, and Rave 2019) considers agents that shrink to total length 1 and continue to occupy only their goal vertices once they arrive there. This is analogous to the standard MAPF problem.
- MTPF Variant 2: A second variant assumes that agents enter private slidings when they arrive at their goal vertices and thus disappear from the map, which means that they shrink to total length 0 and no longer occupy any vertices. This is the MTPF variant used in the NeurIPS 2020 Flatland Challenge (Laurent et al. 2021).
- MTPF Variant 3: A third variant assumes that each agent parks its head at its goal vertex, thus occupying $k_i + 1$ vertices once arrived there. This variant is distinct from a variant where the goal fully specifies the final vertices of the head and body of the agent.

A plan π is a set of paths π_i for each agent a_i. It is a valid MTPF plan iff it does not contain any conflicts of the following three kinds.

Definition 1 (Occupation Conflicts)**.** *An occupation conflict $\langle a_i, a_j, v, t\rangle$ occurs iff agents a_i and a_j both occupy vertex v at timestep t, i.e., $v \in O_i(t) \cap O_j(t)$.*

Definition 2 (Self Conflicts)**.** *A self conflict $\langle a_i, v, t\rangle$ occurs iff agent a_i occupies vertex v at timestep t more than once, i.e., v appears more than once in $O_i(t)$.*

Definition 3 (Edge Conflicts)**.** *If $k_i = 0$ and $k_j = 0$, then an edge conflict occurs iff agents a_i and a_j traverse the same edge in opposite directions at the same timestep. Otherwise, an edge conflict implies an occupation conflict. We ignore edge conflicts in the remainder of the paper, but they are handled correctly by our implementation.*

Our task is to find an optimal MTPF plan, which is a valid MTPF plan that minimizes some objective. We use the *Sum of Individual Costs (SIC)* $\sum_i et_i$, where et_i is the *end time* (equivalent to the number of timesteps in the path) for agent a_i.

Previous Approaches

We provide a brief overview of existing work for MTPF (which we tackle) and a closely related variant known as kR-MAPF (from which we adapt various ideas).

MT-CBS

Multi-Train Conflict-Based Search (MT-CBS) (Atzmon, Diei, and Rave 2019) is an algorithm capable of computing optimal MTPF plans. It is a variant of the popular CBS algorithm (Sharon et al. 2015) for optimal MAPF. We provide a brief description of MT-CBS, since it is closely related to our own contributions.

In MT-CBS, each agent is assigned a path from its start vertex to its goal vertex. Coordination between agents is due to constraints in a binary Constraint Tree (CT). Each node n in the CT is associated with a set of constraints C and a plan π. If n is chosen for expansion and π has no conflicts, then MT-CBS returns π as an optimal MTPF plan. Otherwise, it selects an occupation conflict $\langle a_i, a_j, v, t\rangle$ in π and determines a disjunction of constraints $c_i \vee c_j$ which is true in

[1]This restriction is only required for our *rectangle symmetry breaking* technique. Otherwise, the 4-neighbor restriction can be relaxed, and all other techniques remain applicable.

any valid MTPF plan, but each of whose disjuncts prevents the chosen conflict from repeating. n is then expanded by adding two child nodes to the CT, one with the constraint set $C \cup \{c_i\}$ and the other one with the constraint set $C \cup \{c_j\}$. Only one of the two agents a_i and a_j is replanned in each child node to determine a new path for it. To address the occupation conflict, MT-CBS uses the two *occupation constraints*

- $c_i \equiv v \notin O_i(t)$ and
- $c_j \equiv v \notin O_j(t)$.

Clearly, any valid MTPF plan π must satisfy $c_i \vee c_j$ because, otherwise, both agents occupy vertex v at timestep t, leading to an occupation conflict. The authors of (Atzmon, Diei, and Rave 2019) also modified the low-level path planner to avoid assigning agents paths with self conflicts (which we describe in detail later).

kR-MAPF and K-CBS

kR-MAPF (Atzmon et al. 2018) is a variant of MAPF that aims to handle the situation where agents can be unexpectedly delayed during the execution of the plan. Such delays can happen for a number of reasons, e.g., due to mechanical difference, failure, or an agent being otherwise interrupted on the way to its goal vertex. A valid kR-MAPF plan avoids conflicts due to such delays as long as no agent is delayed by more than k timesteps, where the delay limit k is a user-provided parameter. K-CBS (Atzmon et al. 2018) computes optimal kR-MAPF plans that provide such guarantees. It detects k-delay conflicts, which occur between two agents a_i and a_j at vertex v at timestep t iff $h_i(t) = h_j(t') = v$ and $0 \leq t' - t \leq k$, and resolves each of them by the two sets of head constraints

- $c_i \equiv \forall t' \in [t, t + k], h_i(t') \neq v$ and
- $c_j \equiv \forall t' \in [t, t + k], h_j(t') \neq v$.

Unfortunately, K-CBS suffers from a variety of issues that can make it inefficient. Recent work (Chen et al. 2021) improves it via the addition of speed-up techniques such as admissible heuristics (Felner et al. 2018), conflict prioritization (Boyarski et al. 2015), and symmetry reasoning (Li et al. 2020). The resulting algorithm K-CBSH-RCT represents the state-of-the-art in this area.

Low-Level Solver in MT-CBS and K-CBS

A critical component of CBS for optimal MAPF is the planning and re-planning of single-agent paths in its low level, subject to constraints. For an agent a_i with $k_i = 0$, a simple A* search is sufficient. Here, constraints are modeled as temporal obstacles, and a path of smallest end time is returned, where ties are broken by a secondary criterion, such as the number of times the path uses spatio-temporal vertices assigned to the paths of other agents in the current plan. We call this solver Low-Level Path Planning (LLPP).

LLPP is also applicable to K-CBS since resolving conflicts caused by delays requires adding only head constraints. However, this is not the case for MTPF, where self conflicts can occur. To avoid self conflicts and to distinguish occupations with the same head vertex but different body vertices, an agent with body length $k_i > 0$ must remember all its $k_i + 1$ occupied vertices during the search . In addition, if an agent is constrained by an occupation constraint $v \notin O_i(t)$, we must record all vertices occupied by the agent to ensure satisfaction of the constraint. This makes path planning for MTPF much more time-consuming. We call this solver Low-Level Train Planning (LLTP).

MTPF Plans from kR-MAPF Plans

We first extend kR-MAPF to handling agents of varying k values and then show that kR-MAPF is a relaxation of MTPF.

kR-MAPF with Varying Robustness

kR-MAPF assumes that the delay of each agent is at most k timesteps. We now generalize kR-MAPF to handling agents a_i with different delay limits k_i.

Definition 4 (Δ-Delay Conflicts)**.** *A Δ-delay conflict $\langle a_i, a_j, v, t, \Delta \rangle$ occurs iff agents a_i and a_j plan to enter the same vertex v at timesteps t and $t' = t + \Delta$, $\Delta \in [0, k_i]$, respectively, i.e., $h_i(t) = h_j(t') = v$ and $0 \leq t' - t \leq k_i$.*

A plan is a valid kR-MAPF plan iff it contains no Δ-delay conflict. We resolve a Δ-delay conflict $\langle a_i, a_j, v, t, \Delta \rangle$ by the two head constraints

- $c_i \equiv \forall t' \in [t, t + k_j], h_i(t') \neq v$ and
- $c_j \equiv \forall t' \in [t, t + k_i], h_j(t') \neq v$.

Theorem 1. *Any valid kR-MAPF plan π satisfies $c_i \vee c_j$.*

Proof. Assume the contrary, namely that a valid kR-MAPF plan π violates both constraints, so agent a_i occupies vertex v at timestep $t_i \in [t, t + k_j]$ and agent a_j occupies vertex v at timestep $t_j \in [t, t + k_i]$. If $t_i \leq t_j$, then the earliest timestep when agent a_i occupies vertex v is timestep $t_i = t$ and the latest timestep when agent a_j occupies vertex v is timestep $t_j = t + k_i$. Then, $t_j - t_i \leq t + k_i - t = k_i$ and agents a_i and a_j have a Δ-delay conflict. If $t_j < t_i$, then $t_i - t_j \leq t + k_j - t = k_j$, and agents a_j and a_i have a Δ-delay conflict. Since there is a Δ-delay conflict in either case, π is not a valid plan. Contradiction. □

kR-MAPF as a Relaxation of MTPF

We now show that kR-MAPF is a strict relaxation of MTPF.

Lemma 1. *A plan with a Δ-delay conflict has an occupation conflict.*

Proof. Given a Δ-delay conflict $\langle a_i, a_j, v, t, \Delta \rangle$, we know that $h_i(t) = v$. Because agent a_i has body length k_i for the MTPF instance, it occupies vertex v at least until timestep $t + k_i$. Since $\Delta \leq k_i$, there is an occupation conflict $\langle a_i, a_j, v, t + \Delta \rangle$. □

The reverse does not hold. For example, Figure 1(right) shows an occupation conflict $\langle a_1, a_2, D, t + 4 \rangle$, which is not a Δ-delay conflict, as indicated by Figure 1(left). With Lemma 1, we have the following key theorem.

Theorem 2. *A valid MTPF plan is a valid kR-MAPF plan.*

Algorithm 1: Lazy Train Path Finding by CBS.

```
LT-CBS(G, A):
    foreach a_i ∈ A:
        M_i = LLPP
        // M records each agent's low-level
        π_i = a shortest path for agent a_i planned by M_i
    Q.insert((π, ∅, M))
    while Q ≠ ∅:
        (π, C, M) = Q.pop()
        // The top entry is also removed
        if π has a Δ-delay conflict:    // or edge conflict
            select a Δ-delay conflict x ≡ ⟨a_i, a_j, v, t, Δ⟩ in π
            (c_i, c_j) = constraints for resolving x
            replan-insert(a_i, C ∪ {c_i}, M, π)
            replan-insert(a_j, C ∪ {c_j}, M, π)
        elseif π has a self conflict:
            choose self conflict ⟨a_i, v, t⟩ in π
            M_i = LLTP
            replan-insert(a_i, C, M, π)
        elseif π has a (head) occupation conflict:
            choose occupation conflict x ≡ ⟨a_i, a_j, v, t⟩ in π
            (c_i, c_j) = constraints for resolving x
            replan-insert(a_i, C ∪ {c_i}, M, π)
            M_j = LLTP
            replan-insert(a_j, C ∪ {c_j}, M, π)
        else: return π
    return ⊥

replan-insert(a_i, C, M, π):
    π'_i = a shortest path for agent a_i that satisfies C planned
        by M_i
    if π'_i exists: Q.insert((π − {π_i} ∪ {π'_i}, C, M))
    return
```

Lazy Train Path Finding

As discussed, LLTP for finding valid MTPF plans is much more time-consuming than LLPP for finding valid kR-MAPF plans, particularly as the body lengths of agents grow. Therefore, when solving MTPF, we delay using the expensive LLTP by searching for valid kR-MAPF plans first since any valid MTPF plan is also a valid kR-MAPF plan. That is, when running CBS, we first resolve Δ-delay conflicts with LLPP. Only when a plan is a valid kR-MAPF plan, we resolve its occupation and self conflicts with LLTP. Algorithm 1 shows our algorithm. which we call Lazy Train Path Finding by CBS (LT-CBS).

To start with, we construct an initial plan π where every agent a_i takes its shortest path, as computed by LLPP. A priority queue of nodes Q stores triples of the type (π, C, M), where π is the plan, C is the set of constraints, and M stores the low-level solver (either LLPP or LLTP) for each agent. The root node stores the initial plan, no constraints, and LLPP for all agents.

Like MT-CBS, we pop the node with the lowest f-value from the priority queue. The f-value in MT-CBS is the SIC of the plan of the node (i.e., $f = g$). In LT-CBS, we use $f = g + h$, where the h-value is an admissible heuristic adapted from MAPF (Felner et al. 2018) that underestimates the minimum increase of the SIC of the plan when resolving all of its conflicts.

When expanding a node, we resolve a Δ-delay conflicts with highest priority. This means computing constraints to resolve the conflict (using symmetry reasoning where applicable; we discuss this shortly) and replanning the paths of the two affected agents. The result is two new child nodes which are added to the priority queue. Each child node uses the same low-level solvers as the parent node.

We resolve a self conflict with second-highest priority. If any agent a_i has a self conflict, we change its low-level solver M_i to LLTP and find a new path for it. We then add to the priority queue a new node with the revised path and no additional constraints. All descendant nodes of this node will use LLTP as the low-level solver for agent a_i.

We resolve an occupation conflict with the lowest priority. MT-CBS uses LLTP to plan paths for both agents involved in an occupation conflict. But, by resolving only *head occupation conflicts*, we only need to change the low-level solver of one of the two agents to LLTP.

If there is no conflict, then we return the current plan.

Definition 5 (Head Occupation Conflicts)**.** *An occupation conflict $\langle a_i, a_j, v, t\rangle$ is a head occupation conflict iff $v = h_i(t)$ or $v = h_j(t)$, that is, the conflict involves the head of some agent. Without loss of generality, we assume a head occupation conflict $\langle a_i, a_j, v, t\rangle$ always has $v = h_i(t)$.*

Lemma 2. *An MTPF plan with an occupation conflict has a head occupation conflict.*

Proof. Consider an occupation conflict $\langle a_i, a_j, v, t\rangle$. If $v = h_i(t)$ or $v = h_j(t)$, then the conflict is a head occupation conflict, and we are done. Otherwise, both agents occupied vertex v already one timestep earlier since their heads moved through v. So $v \in O_i(t-1) \cap O_j(t-1)$, indicating that the two agents have an occupation conflict $\langle a_i, a_j, v, t-1\rangle$. Recursing the argument gives the desired result. □

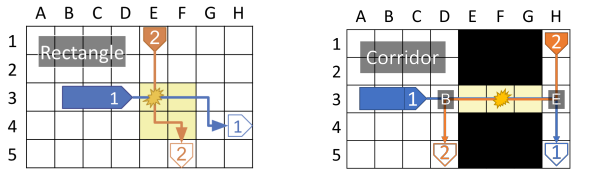
Because of Lemma 2, we restrict the consideration of occupation conflicts in LT-CBS to head occupation conflicts. We resolve a head occupation conflict $\langle a_i, a_j, v, t\rangle$ by a head constraint and an occupation constraint

- $c_i \equiv h_i(t) \neq v$ and
- $c_j \equiv v \notin O_j(t)$.

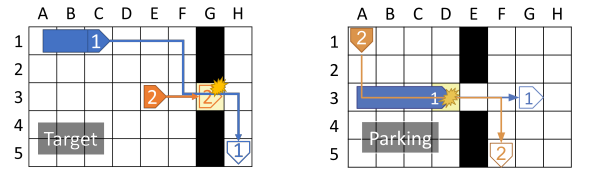
Atzmon, Diei, and Rave (2019) show that any valid MTPF plan satisfies $c_i \vee c_j$. We have to use LLTP to plan paths for a_j, but, for a_i, which only gets a head occupation constraint, we can still use the same low-level solver as in the parent node.

Lemma 3. *Any valid MTPF plan satisfies the constraints of at least one node in priority queue in every iteration of LT-CBS.*

Proof. The lemma holds for the first iteration of LT-CBS because priority queue contains only the root node, which has no constraints. We have shown that any valid MTPF plan must satisfy at least one of the disjunctive constraints that are used to resolve the chosen conflict in a given node n, so



(a) Rectangle conflict. (b) Corridor conflict.



(c) Target conflict. (d) Parking conflict.

Figure 2: Different symmetric conflicts.

any valid MTPF plan that satisfies the constraints of n also satisfies the constraints of at least one of its child nodes. The lemma can be then proved by induction. □

Lemma 4. *The number of nodes expanded by LT-CBS with f-values no larger than a given constant is finite.*

Proof. Since graph G is finite, the number of plans with SIC no larger than the given constant is finite, thus the number of conflicts that can occur in these plans is also finite. A conflict never appears again after resolving it via expansion, so the number of CT nodes with g-values no larger than the given constant is no larger than the number of these conflicts, i.e., it is also finite. Moreover, the h-values of the nodes are always non-negative, so the number of nodes with f-values no larger than the given constant is finite. □

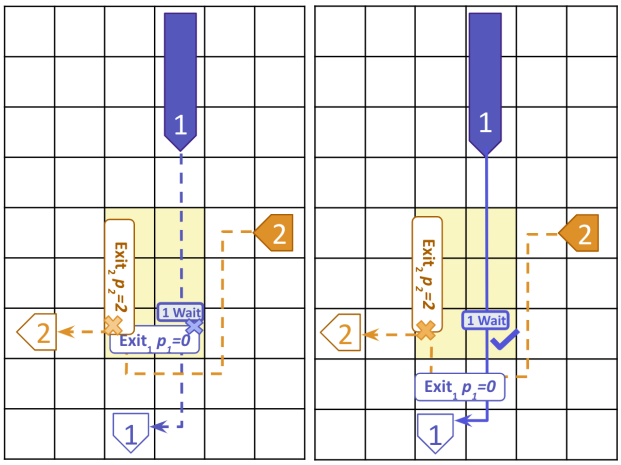
Theorem 3. *LT-CBS is complete and optimal.*

Proof. We follow the proof for CBS (Sharon et al. 2015). The low-level solvers, LLPP and LLTP, are both A*-based searches and complete and optimal. The high level of LT-CBS is also an A* search with completeness guarantee due to Lemma 3. Therefore, the first chosen node with a valid MTPF plan has the minimum SIC; LT-CBS is optimal. From Lemma 4, a valid MTPF plan must be found after a finite number of expansions if it exists; LT-CBS is complete. □

We will introduce several symmetry reasoning techniques for LT-CBS in next section. As long as they guarantee that any valid MTPF plan must satisfy the disjunction of the new symmtey-breaking constraints, Theorem 3 holds, and LT-CBS is complete and optimal.

Symmetry Breaking

Now, we introduce constraint reasoning techniques to resolve symmetric conflicts in a single branching step. Figure 2 shows examples of four different types of symmetric conflicts. Left untreated, each of them can cause the CT to grow exponentially, leading to timeout failure for CBS (Li et al. 2021). The rectangle, corridor, and target conflicts have



(a) (b)

Figure 3: (a) Rectangle conflict between two agents. We place barrier constraints that forbid each agent from exiting the rectangle area at their currently optimal time. Each barrier is extended in the temporal dimension by a number of timesteps equal to the body length of the other agent. The dashed lines indicate a valid MTPF plan (where the blue agent waits for one timestep at the blue-crossed vertex) that violates both barrier constraints at the respective crossed vertices. (b) The blue barrier is pushed away by one step from the yellow rectangle area. Now, the blue path does not violate the blue barrier constraint. Thus, the plan satisfies the disjunction of the two barrier constraints.

been previously considered for (global k) kR-MAPF (Chen et al. 2021). We extend these results to kR-MAPF with varying robustness and to MTPF. Given the space limit, we provide only a sketch of each technique with a focus on how we modify it from using a global k for all agents to using different k_i for different agents a_i. The parking conflict is new and specific to MTPF.

Rectangle Conflicts

This conflict occurs when two agents cross an area in orthogonal directions at the same time. Each agent has many shortest paths, but each one results in a new conflict with the other agent. Figure 2a shows an example: All shortest paths of agent 1 collide with all shortest paths of agent 2 in the yellow shaded rectangle area. To resolve this conflict efficiently, we can use a pair of barrier constraints (each of which is a set of head constraints) that prevents one of the conflicting agents from exiting the rectangle area at its currently optimal timestep (and maybe timesteps afterward, depending on the body length of the other agent). For example, the barrier constraints for the two agents in Figure 2a are

- $B_1 \equiv h_1(2) \neq \text{F3} \wedge h_1(3) \neq \text{F4}$ and
- $B_2 \equiv h_2(3) \neq \text{E4} \wedge h_2(4) \neq \text{E4} \wedge h_2(4) \neq \text{F4} \wedge h_2(5) \neq \text{F4}$.

Here, B_2 blocks two timesteps per exit vertex for agent 2 because, even if agent 2 waits for one timestep and then

enters the rectangle area, it still collides with the body of agent 1. Any valid MTPF plan satisfies $B_1 \vee B_2$, and a set of collision-free paths can be found if we replan for agent 1 with respect to B_1 or for agent 2 with respect to B_2. That is, such pair of barrier constraints can resolve the rectangle conflicts in a single branching step while preserving the completeness and optimality of LT-CBS.

However, this technique does not always work. Figure 3a shows a counterexample where the pair of dashed paths is a valid MTPF plan that violates both barrier constraints. To address this issue, Chen et al. (2021) propose to "push away" the barrier constraints from the rectangle area, as shown in Figure 3b. Formally, we define a series of *entrance* and *exit* barrier constraints:

- a_i Exits: $B_i[l_i, p_i]$,
- a_i Entrances: $B'_i[l_i, p_i]$,
- a_j Exits: $B_j[l_j, p_j]$, and
- a_j Entrances: $B'_j[l_j, p_j]$,

where $p_i \in [0, k_j]$ and $p_j \in [0, k_i]$ are the "thickness" (i.e, how many timestep are disallowed from the earliest timestep that the constrained agent could reach the constrained vertex) of the barriers, and $l_i = \lfloor \frac{p_j}{2} \rfloor$ and $l_j = \lfloor \frac{p_i}{2} \rfloor$ are the distances for which the barriers are pushed away from the rectangle area. Intuitively, an entrance barrier and an exit barrier for an agent are a pair of barrier constraints that it must violate to enter and exit the rectangle area, each one "thicker" than the last in the temporal dimension but each one also being "pushed" further away from the rectangle area than the last. To resolve a rectangle conflict, we require that every optimal path that uses an exit barrier must pass through the corresponding entrance barrier. We then branch on the exit barriers with the largest p satisfying this requirement to eliminate as many conflicts as possible. Please refer to (Chen et al. 2021) for the details of these barrier constraints. The completeness and optimality proofs from Chen et al. (2021) still hold here as we only change the range of p_i and p_j.

Corridor Conflicts

A *corridor* is a sequence of vertices $C = [B, v_1, \ldots, v_n, E]$, where each interior vertex v_i is only connected to the two adjacent neighbors in the list. Its length l is the distance between its endpoints. A *corridor conflict* is a Δ-delay conflict $\langle a_i, a_j, v, t, \Delta \rangle$ with $v \in C$ and agents a_i and a_j moving in opposite directions inside the corridor. Figure 2b shows an example. Chen et al. (2021) develop a constraint reasoning technique to resolve such conflicts in (global k) kR-MAPF. We extend this technique to agents with individual k values. The new constraints are

- $c_i \equiv \forall t \in [0, \min(\tau_i^E - 1, t_j^B + l + k_j)], h_i(t) \neq E$ and
- $c_j \equiv \forall t \in [0, \min(\tau_j^B - 1, t_i^E + l + k_i)], h_j(t) \neq B$,

where agent a_i (resp. a_j) moves from B to E (resp. E to B), t_x^y is the earliest timestep for agent a_x to reach vertex y, and τ_x^y is the earliest timestep for agent a_x to reach vertex y without traversing the corridor. Any valid MTPF plan satisfies $c_i \vee c_j$ because, otherwise, there is a valid MTPF plan that moves agent a_i to E at timestep $t_i \leq \min(\tau_i^E - 1, t_j^B + l + k_j)$ and a_j to B at timestep $t_j \leq \min(\tau_j^B - 1, t_i^E + l + k_i)$. Let us focus on a_i first. Since $t_i \leq \tau_i^E - 1 < \tau_i^E$, a_i can only traverse the corridor to reach E. The latest timestep for a_i to enter the corridor from B is $t_i - l \leq t_j^B + k_j$, which is one timestep earlier than the earliest timestep when a_j can completely leave B. Given that the plan is conflict-free, a_i must traverse the corridor before a_j. However, using the similar reasoning on a_j can prove that a_j must traverse the corridor before a_i, resulting in conflicting arguments. Therefore, $c_1 \vee c_2$ holds in any valid MTPF plan.

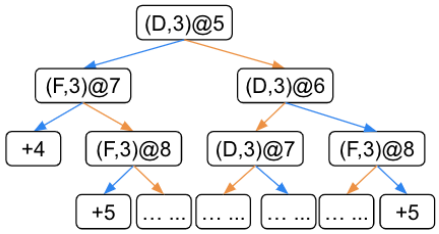


Figure 4: Constraint tree for resolving the parking conflict in Figure 2d. $(x,y)@t$ indicates a conflict at vertex (x,y) at timestep t. Blue branches add constrains to agent 1, and yellow to agent 2. $+C$ indicates a valid MTPF plan with a SIC that is C larger than the cost of the root node.

Target Conflicts

In MTPF Variants 1 and 3, the agents occupy their goal vertices forever after they complete their paths. A *target conflict* is a 0-delay conflict of the form $\langle a_i, a_j, g_i, t, 0 \rangle, t \geq et_i$; i.e., agent a_j runs into agent a_i which waits at its goal vertex. Figure 2c gives an example. To resolve target conflicts in kR-MAPF, Chen et al. (2021) propose branching on the end time et_i of agent a_i. We extend it to handle individual k_i values by branching via constraints

- $c_i \equiv et_i > t + k_j$ and
- $c_j \equiv et_i \leq t + k_j$.

For the child node with c_i, we replan for agent a_i by forcing it to complete its path after timestep $t + k_j$. For the child node with c_j, we require that agent a_i never completes its path later than $t + k_j$. c_j also indicates that any other agent $a_x, x \neq i$ cannot have $h_x(t_x) = g_i, t_x \geq t + k_j - k_x$ because, otherwise, a_x occupies g_i until timestep $t_x + k_x \geq t + k_j$, conflicting with a_i which has waited at g_i since timestep $t + k_j$. Clearly, $c_1 \vee c_2$ holds in any valid MTPF plan.

Parking Conflicts

In Variant 3, a new kind of head occupation conflict arises of form $\langle a_i, a_j, v, t \rangle, t \geq et_i, v \neq g_i$; i.e., agent a_j runs into the body of agent a_i which is "parked" at its goal vertex. Figure 2d gives an example. LT-CBS needs to split many times, as shown in Figure 4, since the child nodes that simply delay agent 2 by one timestep look more attractive than those that force agent 1 to change its parking position (recall only the head must be at the goal vertex). Parking conflicts are especially challenging when t is far greater than the end time et_i of agent a_i. Although similar to target conflicts, we cannot

treat situations in the same way, since the parked agent a_i could possibly park its body in a different final position.

To efficiently resolve parking conflicts, we introduce the *parking constraint* and use disjoint splitting (Li et al. 2019a) to generate $c_i \vee c_j$, namely

- $c_i \equiv v \in O_i(et_i) \wedge et_i \leq t$ and
- $c_j \equiv v \notin O_i(et_i) \vee et_i > t$.

Constraint c_i asks agent a_i to complete its path before timestep t and permanently "park" on vertex v. A consequence of this constraint is that any other agent occupying vertex v at or after timestep t must also be replanned. That is, c_i implies $\forall x \neq i \, \forall t_x \geq t, v \notin O_x(t_x)$. Constraint c_j asks a_i to not "park" at vertex v until after timestep t. If both c_i and c_j are simultaneously violated it means there exists an agent a_i with $v \in O_i(et_i), et_i \leq t$ and another agent a_j has $h_j(t') = v, t' > t$; i.e., there must exist a parking conflict. Clearly $c_i \vee c_j$ holds in any valid MTPF plan.

Experiments

We compare our approaches to MTPF with the original approach MT-CBS (Atzmon, Diei, and Rave 2019). We compare MT-CBS against three variants of LT-CBS, namely the basic variant LT-CBS defined in Algorithm 1, the strongest variant LT-CBSH-RCT, which uses admissible (H)euristics and symmetry reasoning for (R)ectangles, (C)orridors, and (T)arget conflicts, and a strawman variant LT-CBSH-RCT$^-$, which resolves only Δ-delay conflicts (i.e., ignores occupation and self conflicts) and thus finds optimal kR-MAPF plans. Because of Theorem 2, the optimal kR-MAPF plan found by LT-CBSH-RCT$^-$ is an optimal MTPF plan iff the plan does not contain any occupation and self conflicts. That is, LT-CBSH-RCT$^-$ succeeds iff its found plan is a valid MTPF plan. We use it to show the strength of the relaxation for MTPF. (P)arking conflicts reasoning is applicable to and thus considered in only MTPF Variant 3. All algorithms were written in C++. The experiments were performed on a server with Intel Xeon CPU (Skylake) and 64 GB RAM, with a runtime limit of 90 seconds.

We use 4 grid-based maps from a standard MAPF benchmark set (Stern et al. 2019), namely *random-32-32-10* (denoted random), *room-32-32-4* (denoted room), *warehouse-20-40-10-2-1* (denoted warehouse), and *den520d* (denoted game). We also test on Flatland Challenge (Laurent et al. 2021), a train planning problem where the trajectories of moving agents must be coordinated on a simplified rail network. Our experiments use flatland-rl v2.2.2 to generate problem instances. For each domain, we consider an increasing number of agents, and for each number of agents, we solve 25 different instances. We uniformly generate body lengths for agents a_i with

$$k_i = K_{max} - (i-1) \mod (K_{max}+1),$$

where K_{max} is an experiment parameter that defines the maximum body length.

MTPF Variant 1 on Grid-Based Maps

This setup is the same as (Atzmon, Diei, and Rave 2019), and we therefore compare the success rates of our algorithms against those of MT-CBS on our 4 grid-based maps in Figure 5. LT-CBS has a significant advantage over MT-CBS on some maps and is never worse on the other maps. The power of symmetry breaking is made clear by the success rate of LT-CBSH-RCT over that of LT-CBS. The power of relaxation is demonstrated by LT-CBSH-RCT$^-$, whose success rate is usually very close to LT-CBSH-RCT, although there is a notable difference on the room map. This shows that we do not need to use train planning that often. The main drawback of LT-CBSH-RCT$^-$ is its incompleteness: If the returned plan is not a valid MTPF plan, it reports failure, even if a valid MTPF plan exists.

Map	K_{max}	RSOD	LLPP	LLTP
Random	2	0.87	558.3 ± 518.52	1.62 ± 2.04
	4	0.94	887.06 ± 800.33	1.94 ± 3.35
	8	0.97	577.93 ± 923.68	21.87 ± 43.73
Warehouse	2	0.96	146.45 ± 89.51	0.32 ± 0.58
	4	0.91	171.21 ± 120.63	0.12 ± 0.10
	8	0.86	261.75 ± 246.46	0.14 ± 0.12
Room	2	0.66	2392.46 ± 1230.56	1.67 ± 1.11
	4	0.65	2130.81 ± 1187.86	170.67 ± 284.33
	8	0.64	1636.83 ± 995.56	63.53 ± 69.01
Game	2	1.0	87.49 ± 26.52	0.00 ± 0.00
	4	1.00	109.80 ± 52.05	0.00 ± 0.00
	8	1.00	40.30 ± 13.56	0.00 ± 0.00

Table 1: RSOD and average number of LLPP/LLTP searches $\pm 2 \times$*Standard Error* over all solved instances using LT-CBSH-RCT.

Table 1 reports the average number of LLPP and LLTP searches and the ratio of the instances solved by resolving only Δ-delay conflicts (RSOD) over all solved instances using LT-CBSH-RCT. First of all, LLTP is used on most of the maps, indicating that resolving head occupation or self conflicts is needed. LT-CBSH-RCT has high RSOD and barely uses LLTP for most of the solved instances on the warehouse and game maps. However, on the random and room maps, it requires more LLTP searches, indicating that there exist more head occupation or self conflicts that are not Δ-delay conflicts on these two maps.

Furthermore, we count the average number of nodes expanded per low-level search for MT-CBS and LT-CBS. MT-CBS expanded 355 nodes per search and LT-CBS expanded 180 nodes per search, which shows LLTP is more expensive than LLPP.

MTPF Variant 2 on Simplified Railway Systems

Figure 6b shows an example Flatland instance. The map contains several cities connected by rails. Each city has a station that provides off-map parking: before trains begin and after they reach their goal vertices. Each station can be the start or goal vertex for multiple trains. A departure timetable specifies the earliest timestep d_i when a train a_i can enter the map and leave its start station. If train a_i has the smallest ID over all trains departing from the same station, then it can enter the map at or after timestep $d_i = 1$. The train a_j with the next largest ID can enter the map at or after

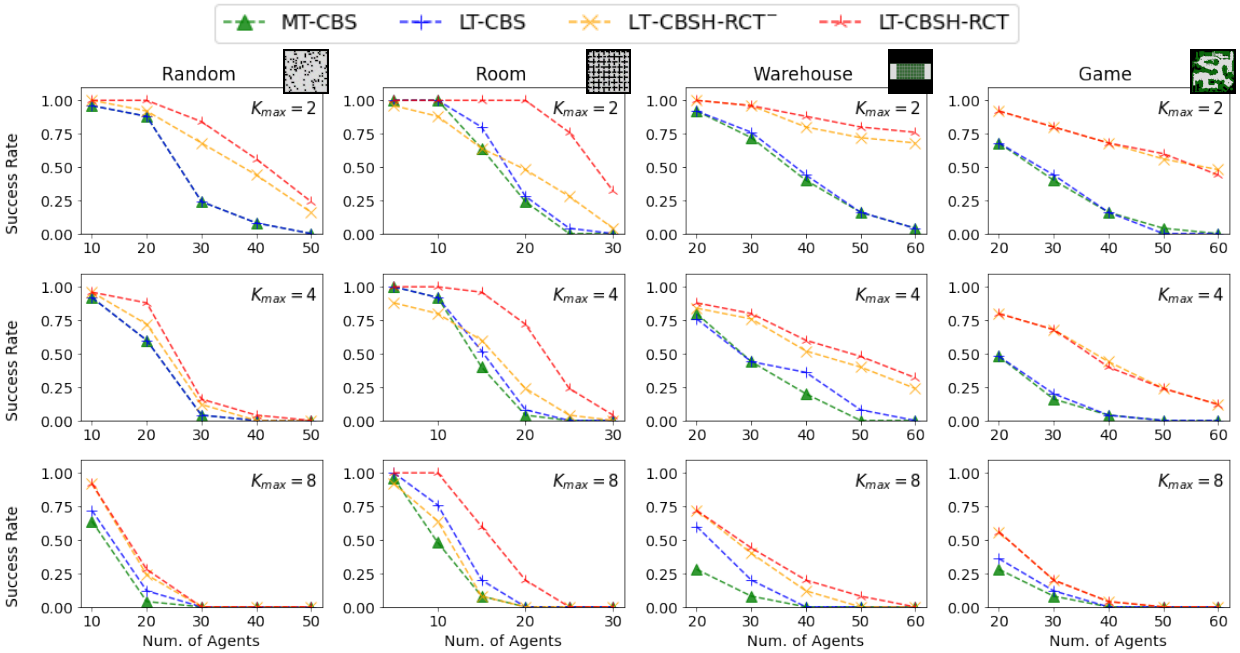
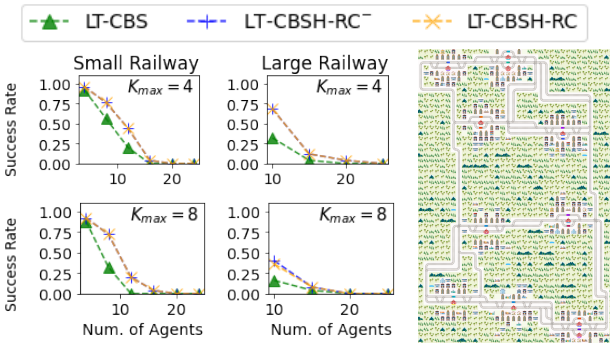


Figure 5: Success rates of four algorithms to MTPF Variant 1 on four grid-based maps with agents of varying body lengths.



(a) Success rate. (b) Railway system.

Figure 6: Success rates of three algorithms to MTPF Variant 2 on railway systems with trains of varying body lengths.

timestep $d_j = d_i + k_i + 1$. The same rules apply for later trains. We use the simulator generator with a random seed varying from 1 to 25 to generate two sets of 25 instances. The first set uses 50×50 grids with a maximum of 8 cities, denoted *small railways*, and the second set uses 100×100 grids with a maximum of 20 cities, denoted *large railways*. In both cases, the generator creates at most 2 rails between cities and 3 rails inside cities.

Figure 6a shows the success rates of LT-CBS, LT-CBSH-RC$^-$, and LT-CBSH-RC. We exclude Target conflict reasoning as they are impossible in Variant 2. Here, LT-CBSH-RC overlaps with LT-CBSH-RC$^-$, both of which are significantly better than LT-CBS. This indicates that we seldom meet occupation or self conflicts in the solved instances (although they could still happen in general). Again, symmetry reasoning helps to solve more instances. Increasing the map size brings limited extra traversable spaces but increases travel distance. Thus, all algorithms achieve higher success rates in small railways.

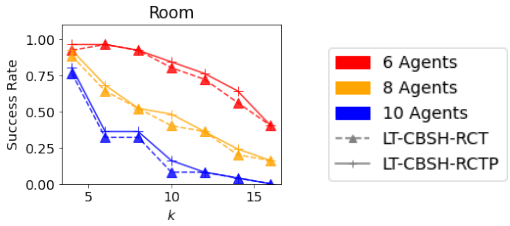


Figure 7: Success rates of two algorithms to MTPF Variant 3 on the room map with agents of the same body length k.

MTPF Variant 3 with Long Agents

We study the influence of parking conflicts on the room map. This map contains many narrow doorways where agents can park and block other agents. All agents have the same length k, and the number of agents is set to 6, 8, and 10. As shown in Figure 7, parking conflicts can not be resolved efficiently by LT-CBSH-RCT. Meanwhile, LT-CBSH-RCTP, where P indicates parking conflicts reasoning, solves more instances within the given runtime limit.

Conclusion

In this paper, we showed that k-Robust MAPF (kR-MAPF) is a strict relaxation of Multi-Train Path Finding (MTPF). To achieve this result, we first extended kR-MAPF to allow for agents with varying robustness. We then proposed

Lazy Train CBS (LT-CBS), a new optimal MTPF algorithm which can often replace expensive Low-Level Train Planning (LLTP) with much less expensive Low-Level Path Planning (LLPP). This change allows LT-CBS to solve substantially more problems in the same amount of time than MT-CBS, a state-of-the-art MTPF planner from the recent literature. To improve LT-CBS further, we introduced a number of generalized symmetry-breaking techniques. This includes resolution strategies for rectangle, corridor and target symmetries, which had previously been considered only for MAPF and kR-MAPF with varying robustness, as well as newly identified *parking symmetry* conflicts, which only occur in MTPF. Together, these changes improve the efficiency of state-of-the-art MTPF algorithms substantially.

Future research will focus on other kinds of symmetries that arise in MTPF, e.g., two agents traversing a wide corridor in the same direction but their paths intersect. We will also overcome the issue that current rectangle reasoning sometimes cannot remove all collision plans in one split to guarantee optimality.

Acknowledgments

The research at Monash University was partially supported by the Australian Research Council under grants DP190100013 and DP200100025 as well as a gift from Amazon. The research at the University of Southern California was partially supported by the National Science Foundation under grant 1409987, 1724392, 1817189, 1837779, 1935712, and 2112533 as well as a gift from Amazon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

Atzmon, D.; Diei, A.; and Rave, D. 2019. Multi-Train Path Finding. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS)*, 125–129.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2018. Robust Multi-Agent Path Finding. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS)*, 2–9.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. E. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 740–746.

Cacchiani, V.; Qi, J.; and Yang, L. 2020. Robust Optimization Models for Integrated Train Stop Planning and Timetabling with Passenger Demand Uncertainty. *Transportation Research Part B: Methodological*, 136: 1–29.

Chen, Z.; Harabor, D.; Li, J.; and Stuckey, P. J. 2021. Symmetry Breaking for k-Robust Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 12267–12274.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. S.; and Koenig, S. 2018. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 83–87.

Laurent, F.; Schneider, M.; Scheller, C.; Watson, J.; Li, J.; Chen, Z.; Zheng, Y.; Chan, S.-H.; Makhnev, K.; Svidchenko, O.; Egorov, V.; Ivanov, D.; Shpilman, A.; Spirovska, E.; Tanevski, O.; Nikov, A.; Grunder, R.; Galevski, D.; Mitrovski, J.; Sartoretti, G.; Luo, Z.; Damani, M.; Bhattacharya, N.; Agarwal, S.; Egli, A.; Nygren, E.; and Mohanty, S. 2021. Flatland Competition 2020: MAPF and MARL for Efficient Train Coordination on a Grid World. In *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, 275–301.

Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2020. New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 193–201.

Li, J.; Harabor, D.; Stuckey, P. J.; Felner, A.; Ma, H.; and Koenig, S. 2019a. Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 279–283.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021. Pairwise Symmetry Reasoning for Multi-Agent Path Finding Search. *Artificial Intelligence*, 301: 103574.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019b. Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 6087–6095.

Lusby, R. M.; Larsen, J.; Ehrgott, M.; and Ryan, D. 2011. Railway Track Allocation: Models and Methods. *OR Spectrum*, 33(4): 843–883.

Mokhtar, H.; Krishnamoorthy, M.; Dayama, N. R.; and Kumar, P. R. 2020. New Approaches for Solving the Convoy Movement Problem. *Transportation Research Part E: Logistics and Transportation Review*, 133: 101802.

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

Singh, A.; Gong, C.; and Choset, H. 2018. Modelling and Path Planning of Snake Robot in Cluttered Environment. In *International Conference on Reconfigurable Mechanisms and Robots (ReMAR)*, 1–6.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. S.; et al. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceeding of the Annual Symposium on Combinatorial Search (SoCS)*, 151–158.

Takemori, T.; Tanaka, M.; and Matsuno, F. 2018. Gait Design for a Snake Robot by Connecting Curve Segments and Experimental Demonstration. *IEEE Transactions on Robotics*, 34(5): 1384–1391.

Yang, L.; Qi, J.; Li, S.; and Gao, Y. 2016. Collaborative Optimization for Train Scheduling and Train Stop Planning on High-Speed Railways. *Omega*, 64: 57–76.