Multi-Agent Combinatorial Path Finding for Tractor-Trailers in Occupancy Grids

Xuemian Wu[1], Zhongqiang Ren[1†]

***Abstract*— This paper investigates a problem called Multi-Agent Combinatorial Path Finding for Tractor-Trailers (MCPF-TT), which seeks collision-free paths for multiple agents from their start to goal locations, visiting a set of intermediate target locations in the middle of the paths, while minimizing the sum of arrival times. Additionally, each agent behaves like a tractor, and a trailer is attached to the agent at each intermediate target location, which increases the "body length" of that agent by one unit. Planning for those tractor-trailers in a cluttered environment introduces additional challenges, since the planner has to plan each agent in a larger state space that includes the position of the attached trailers to avoid self-collision. Furthermore, agents are more likely to collide with each other due to the increasing body lengths, and the conventional collision resolution techniques turn out to be computationally inefficient. This paper develops a new planner called CBSS-TT that includes both novel inter-agent conflict resolution techniques, and a new single-agent planner TTCA* that finds optimal single-agent path while avoiding self-collision. Our test results show that CBSS-TT sometimes requires 60% fewer number of iterations while finding solutions with cheaper costs than the baselines.**

I. Introduction

Multi-Agent Path Finding (MAPF) seeks collision-free paths for multiple agents from their respective start locations to their respective goal locations while minimizing path costs. This paper considers a generalization of MAPF where agents need to visit a known set of intermediate target locations before reaching their goals. This generalization, called Multi-Agent Combinatorial Path Finding (MCPF), seeks "start-target-goal" paths for the agents rather than the "start-goal" paths as in MAPF. MAPF and MCPF arise in logistics applications. MCPF is challenging as it requires both collision avoidance among agents (as in MAPF), and target sequencing, i.e., solving Traveling Salesman Problems (TSPs) to specify the allocation and visiting orders of targets for all agents. Both TSP [1] and MAPF [2] are NP-hard to solve to optimality, and so is MCPF.

Although a few algorithms were developed for MCPF and other similar variants [4]–[12], most of them ignores or over-simplifies the task execution process at each target location. Consider robots collecting empty carts from workstations in a factory and transport the carts to the designated locations in a warehouse (Fig. 1), where the number of carts (body length of an agent) increases at each target location. Consequently, the planner has to consider the increased body length to avoid self-collision and inter-agent collision as well as target sequencing in these cluttered environments. With this motivation, we study a new problem called MCPF for tractor-trailers (MCPF-TT), where the agent body length increases after finishing the task at a target. MCPF-TT simplifies the motion model of robots and the representation of the workspace by using a four-neighbor occupancy grid, and limits its focus to the challenge of planning with varying body lengths.

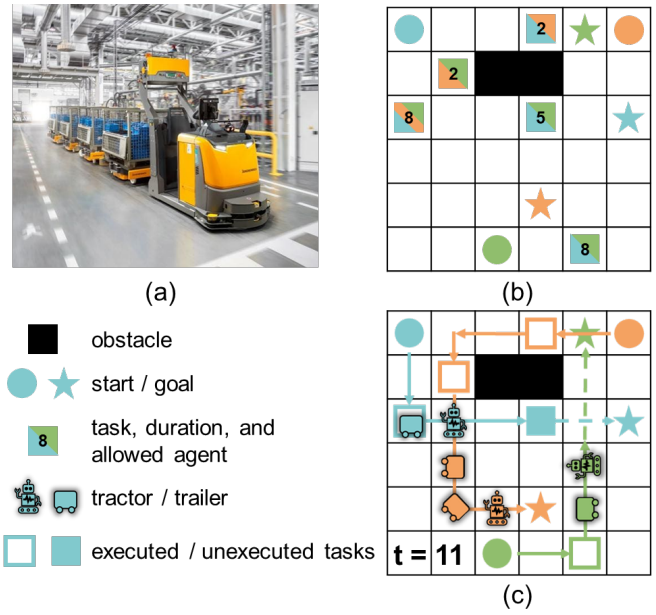

Fig. 1. (a) The motivation of MCPF-TT. A forklift pulling multiple carts can be described as the tractor-trailer model in this paper. The picture is from Mitsubishi Logisnext Americas [3]. (b) A toy example of MCPF-TT, which seeks start-target-goal paths for the agents. Each task has a task duration and the colors indicate assignment constraints. (c) At time $t = 11$, the robots with carts appended are along their paths in a cluttered environment.

The varying body length of agents in MCPF-TT introduces challenges to planning. First, to avoid self-collision and find an optimal individual path for each agent, the planner must explore a much larger state space that includes the number of carts attached to the robot and the position of each of those carts, as opposed to planning merely the position of the robot itself. This state space has varying dimensions and grows exponentially with respect to the number of attached carts, which burdens the computation. Second, the agents with long bodies are more likely to collide with each other, especially when executing tasks, which increases coupling among the agents, and burdens the computation in collision resolution and target sequencing.

To address these challenges, this paper builds upon our prior work on CBSS planner for MCPF [6], which consists

[1] Shanghai Jiao Tong University, China (wuxuemian0114@gmail.com, zhongqiang.ren@sjtu.edu.cn). This work was supported by the Natural Science Foundation of Shanghai under Grant 24ZR1435900, and the Natural Science Foundation of China under Grant 62403313.

† Corresponding author

of a high-level search that sequences the targets and resolves agent-agent collision, and a low-level search that plans individually optimal paths for each agent subject to constraints added by the high-level. On the low-level, we develop a new single-agent planner called Tractor-Trailer Combinatorial A* (TTCA*) that is guaranteed to find an optimal path for each agent with varying number of carts. On the high-level, we introduce new branching rules to resolve agent-agent collision more efficiently by considering the body length. We show that the entire approach is guaranteed to find an optimal solution for MCPF-TT.

We evaluate our approach in several different maps from a MAPF benchmark set. Our results show that with the help of TTCA* and the new branching rules, CBSS-TT can find conflict-free paths using up to 60% fewer iterations while finding better solutions, especially in small and cluttered environments.

A. Other Related Work

Several variants of MAPF were studied where an agent may occupy more than one vertices, similar to the notion of body length in MCPF-TT. Among them, k-Robust MAPF [13] considers potential delay in agents' motion by letting an agent occupy a location for k subsequent time steps after that agent traverses that location. Multi-Train Path Finding (MTPF) [14], [15] seeks collision-free paths for the agents that have a fixed body length longer than one. These variants of MAPF seeks start-goal paths without intermediate targets and thus differ from MCPF-TT. For conflict resolution, our new method differs from the ones in MTPF [14], which is elaborated later, and we compare them in our experiments.

II. Problem Formulation

A. Workspace and Agents

Let index set $I = \{1, 2, \dots, N\}$ denote a set of N agents. All agents share a workspace that is represented as an undirected graph $G^W = (V^W, E^W, c^W)$, where W stands for workspace. Each vertex $v \in V^W$ represents a possible location of an agent. Each edge $e = (u, v) \in E^W \subseteq V^W \times V^W$ represents an action that moves an agent from u to v. Time is discretized into unit-size time steps. The cost of an edge e is denoted as $c^W(e)$, which is one, indicating that it takes an agent one time unit to traverse that edge.

Let a superscript $i \in I$ over a variable denote the specific agent to which the variable belongs (e.g. $v^i \in V^W$ means a vertex corresponding to agent i). Let $v^i_o \in V^W$ denote the initial vertex (also called the origin) of agent i and V_o denote the set of all initial vertices of the agents. Let $v^i_d \in V^W$ denote the goal vertex (also called the destination) of agent i. There are N goal vertices in G^W denoted by the set $V_d \subset V^W$. In addition, let $V_t \subset V^W \setminus \{V_o \cup V_d\}$ denote the set of target vertices. Each target vertex is associated with a task that must be executed by an agent. For each vertex $v \in V_t \cup V_d$, let $f_A(v) \subseteq I$ denote the subset of agents that are capable to execute the task at v. For $v \in V_t \cup V_d$ and an agent $i \notin f_A(v)$, agent i can still occupy v at any time but cannot execute the task there. These sets $f_A(v), v \in V_t \cup V_d$ are used to formulate the (agent-target) assignment constraints.

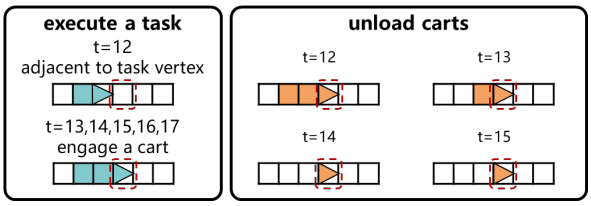

Fig. 2. Example for executing a task and unloading carts. As shown in Fig. 1 (c), the blue agent will execute a task from $t = 13$, and the orange agent will unload carts from $t = 12$.

B. Body Length and Occupation List

Let $b^i \in \mathbb{N}$ denote the *body length* of agent i, which is always equal to the number of carts that agent i is appended. The total length of agent i is $b^i + 1$, which includes the agent itself. Let $O^i(t) \subset V^W$ denote the occupation list, which is a list of $b^i + 1$ adjacent vertices that agent i occupies at time t (i.e., $|O^i(t)| = b^i + 1$). Let $O^i(t, k), k = 0, 1, \cdots, b^i$ denote the k-th vertex occupied by the agent, and the first vertex $O^i(t, k = 0)$ is occupied by the agent itself and is called the *head* vertex of agent i.

Let $V_{adj}(v) \subset V^W$ denote the subset of vertices that are adjacent to v in G^W. For an occupation list $O^i(t) = [v^i_0, v^i_1, \cdots, v^i_{b^i}]$, if the agent moves to an adjacent vertex $u \in V_{adj}(O^i(t, 0))$ at time $t + 1$, then each cart follows the motion of the former cart, i.e., at time $t + 1$ the k-th cart $O(t + 1, k)$ occupies the location $O(t, k - 1)$ of the $(k-1)$-th cart at time t, and the new occupation list becomes $O^i(t+1) = [u, v^i_0, v^i_1, \cdots, v^i_{b^i-1}]$ at time $t+1$. If the agent waits in place, then $O^i(t+1) = O^i(t)$. When the agent has at least one cart, i.e., $b^i \geq 1$, the agent cannot move backwards, since backward motion can incur self-collision.

It usually takes some time for an agent to engage a new cart (e.g. with the help of a human operator) at a target vertex, which is also called executing a task. Let $\tau^i(v) \in \mathbb{N}$ denote the amount of time that agent $i \in f_A(v)$ takes to execute the task at $v \in V_t$. To start executing the task at $v^i_k \in V_t$, the agent must be in a vertex such that the head vertex $O^i(t, 0)$ is adjacent to v^i_k (i.e., $O^i(t, 0) \in V_{adj}(v^i_k)$), and v^i_k is not occupied by any other agent. If agent i claims that it starts to execute the task at $v^i_k \in V_t$, agent i will then occupy $b^i + 2$ vertices within the time range $[t + 1, t + \tau^i(v)]$ and thereafter, where the occupation list changes from $O^i(t) = [v^i_0, v^i_1, ..., v^i_{b^i}]$ to $O^i(t') = [v^i_k, v^i_0, v^i_1, ..., v^i_{b^i}]$. There is an example in Fig. 2. Each target has only one task and we use terms task and target interchangeably hereafter.

When agent i has completed all tasks and arrived at its goal $O^i(t, 0) = v^i_d$, agent i starts to unload the carts, which decreases the body length of the agent by one unit per time step, and the agent eventually occupies only its goal vertex v^i_d thereafter. There is an example in Fig. 2.

C. Path and Conflicts

All agents share a global clock and start moving along their paths from time $t = 0$. Let $\pi^i(v^i_0, v^i_\ell) =$

$(v_0^i, v_1^i, v_2^i, \ldots, v_\ell^i)$ denote a path for agent i between vertices v_0^i and v_ℓ^i, where each vertex $v_k^i \in \pi^i(v_0^i, v_\ell^i)$ represents the head vertex of the agent at time $t = k$. If a path $\pi^i(v_o^i, v_d^i)$ connects the initial vertex and the goal vertex of agent i, we denote this path as π^i to simply the notation. Let $g(\pi^i)$ denote the cost of the path, which is the arrival time at v_d^i.

Any two agents $i, j \in I$ are in conflict if one of the following two cases happens. The first case is an edge conflict (i, j, e, t), where the heads of two agents $i, j \in I$ go through the same edge e from opposite directions between times t and $t+1$. The second case is a vertex conflict (i, j, v, t) where a vertex v is occupied by two agents $i, j \in I$ or their bodies at the same time t, i.e., $v \in O^i(t) \wedge v \in O^j(t)$.

The problem MCPF-TT seeks to find a set of conflict-free paths for the agents such that (1) the task at any $v \in V_t$ is executed by an eligible agent $i \in f_A(v)$, (2) the path for each agent $i \in I$ starts at its initial vertex and terminates at a unique goal vertex $u \in V_d$ such that $i \in f_A(u)$, and (3) the sum of costs of all agents' paths reaches the minimum.

Remark 1: When body length b^i remains unchanged for any agent at any target, MCPF-TT then becomes MCPF-D (D stands for task duration), which has been studied [4]. When the body length of any agent is always zero and the task duration $\tau^i(v) = 0$ for all $v \in V_t, i \in f_A(v)$, MCPF-TT becomes MCPF [6].

III. Preliminaries

A. Conflict-Based Search

Conflict-Based Search (CBS) [16] is a popular two-level search algorithm that finds an optimal joint path for MAPF. At the high-level, CBS starts with a root node $P_{root} = (\pi, g, \Omega)$, where $\pi = (\pi^1, \pi^2, \ldots, \pi^N)$ is the joint path that connects origins and destinations of all agents respectively; g is the sum of costs of all agents' paths; Ω is a set of constraints, and each constraint is of form (i, v, t) (or (i, e, t)), which indicates agent i can not occupy the vertex v (or edge e) at time t. The constraint set of the root node $\Omega_{root} = \emptyset$. CBS then selects a specific node $P = (\pi, g, \Omega)$ from that has the smallest g-value, and detects conflicts along the paths of any pair of agents. If no conflict is detected, π is returned as an optimal joint path. Otherwise, according to the detected conflict (i, j, v, t), two constraints (i, v, t) and (j, v, t) are generated, and each of them corresponds to a new constraint sets $\Omega \bigcup \{i, v, t\}$ and $\Omega \bigcup \{j, v, t\}$ (same to edge conflicts (i, j, e, t)). For each of those two constraints sets Ω', CBS runs a low-level search to find a new path satisfying all the constraints in Ω'. At the low-level, a single-agent planner is invoked to plan an optimal path π'^i that satisfies all constraints related to agent i in Ω'. Then a new joint path π' is formed by copying the paths of other agents from π and using π'^i as the path of agent i, and a corresponding node $P' = (\pi', g', \Omega')$ is generated for high-level search. The generated nodes are stored in an open list OPEN which is a priority queue that ranks the nodes based on their g-cost from the minimum to the maximum. In the next iteration, CBS pops the cheapest node from OPEN and repeats the above process. In CBS, the high-level search spans a binary tree, called constraint tree $\mathcal{T}$, and leaf nodes of $\mathcal{T}$ are the ones in OPEN for further expansion, until a node with collision-free paths is found. CBS guarantees finding a conflict-free joint path with the minimal cost.

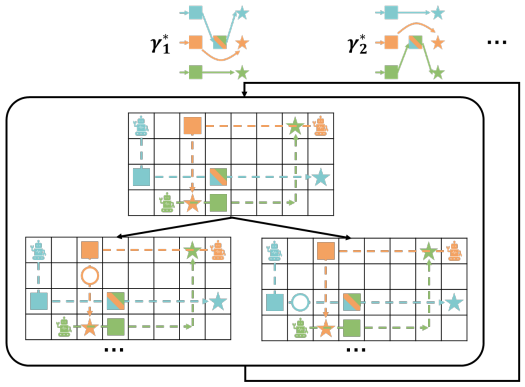

Fig. 3. An illustration of CBSS [6], which interleaves target sequencing and path planning.

Algorithm 1 Pseudocode for CBSS

INPUT: $G^W = (V^W, E^W, c^W)$
OUTPUT: a conflict-free joint path π in G^W.
$G_1 = (V_1, E_1, c_1) \leftarrow$ *GetTargetGraph*(G^W)
$\gamma_1^* \leftarrow$ *K-best-Sequencing*(G_1,f_A,$K = 1$)
$\Omega \leftarrow \emptyset$, $\pi, g \leftarrow$ *LowLevelPlan*(γ_1^*, Ω)
Add $P_{root,1} = (\pi, g, \Omega)$ to OPEN
while OPEN $\neq \emptyset$ **do**
  $P = (\pi, g, \Omega) \leftarrow$ OPEN.pop()
  $P' = (\pi', g', \Omega') \leftarrow$ *CheckNewRoot*(P, OPEN)
  $cft \leftarrow$ *DetectConflict*(π')
  if $cft = NULL$ **then return** π'
  $\Omega \leftarrow$ *GenerateConstraints*(cft)
  for all $\omega^i \in \Omega$ **do**
    $\Omega'' = \Omega' \cup \{\omega^i\}$
    $\pi'', g'' \leftarrow$ *LowLevelPlan*($\gamma(P')$, Ω'')
    Add $P'' = (\pi'', g'', \Omega'')$ to OPEN
return failure

B. Conflict-Based Steiner Search

CBSS [6] finds an optimal conflict-free joint path for MCPF, and is summarized in Fig. 3 and Alg. 1. CBSS alternates between target sequencing, allocate and order the targets for each agent by solving a multi-agent TSP (mTSP), and CBS-like path planning. For target sequencing, CBSS first creates a complete undirected metric graph called the target graph $G^T = (V^T, E^T, c^T)$, where the vertex set V^T includes all the start, targets and goals, i.e., $V^T = V_o \cup V_t \cup V_d$, and the cost of an edge $c^T(u, v)$ in G^T is the minimum path cost connecting u, v in G^W ignoring any agent-agent conflict. CBSS solves a K-Best mTSP on G^T (*K-best-Sequencing*) to obtain a set of joint sequences $\gamma_1^*, \gamma_2^*, \cdots, \gamma_K^*$, where each joint sequence $\gamma_k^* = (\gamma_k^{*1}, \gamma_k^{*2}, \cdots, \gamma_k^{*N})$ consists of N individual target sequence, and each individual sequence γ_k^{*i} is a path in G^T from v_o^i to v_d^i. As shown in Fig. 3, a joint sequence specifies the allocation and visiting order of targets among the agents. The cost of a joint sequence $c(\gamma_k^*)$ is the sum of costs of

individual sequences in γ_k^*, and the K-best joint sequences are the top K cheapest joint sequences in G^T with increasing costs, i.e., $c(\gamma_1^*) \leq c(\gamma_2^*) \leq \cdots \leq c(\gamma_K^*)$. CBSS solves K-Best mTSP in an incremental fashion by first setting $K = 1$ to only obtain γ_1^*, and then solves $K = 2$ for γ_2^* only when needed as explained next.

For each joint sequence γ_k^*, CBSS lets all agents visit the targets as specified by γ_k^* (*LowLevelPlan*) and uses CBS like search to resolve conflicts among the agents (*Generate-Constraints*), which creates a corresponding constraint tree $\mathcal{T}_k$. The open list of CBSS contains the leaf (open) nodes in all trees $\mathcal{T}_k$, and when the cost of a popped node is greater than a threshold value (*CheckNewRoot*), CBSS increases K to $K + 1$ and solves for the next-best joint sequence γ_{K+1}^* and creates a new constraint tree. CBSS terminates if a node with conflict-free joint path is popped from OPEN, which is optimal or bounded sub-optimal depending on threshold value that tunes when the next-best sequence is generated.

IV. Conflict Resolution

Existing conflict resolution strategies [14]–[16] focus mainly on MAPF with fixed body length. However, due to the variable body length and task duration of MCPF-TT, these strategies can be inapplicable or inefficient. This section introduces new conflict resolution methods. We begin with the case when there is no task duration at any target and then the general case with task duration.

A. Without Task Duration

Let $(i, b^i, k^i, j, b^j, k^j, v, t)$ denote a *body conflict*, which means that the k^i-th cart of agent i (with body length b^i) has a conflict with the k^j-th cart of agent j (with body length b^j) at vertex v at time t. When such a body conflict is first detected, the head vertex of either i or j must be at v, since otherwise, these two agents must have collided at earlier time steps which would be detected earlier. Without loss of generality, we assume that the head vertex of agent j is at the vertex v and $k^j = 0$ for the rest of this section. Let (i, b^i, k^i, v, t^i) denote a *body constraint*, meaning that when the body length of agent i is b^i, the k^i-th cart of agent i can not occupy v at time t^i. Then a body conflict can be resolved by the following branching rule, which generates two sets of constraints, where each set of constraint leads to a new high-level node in CBS search.

$$\Omega_i = \{(i, b^i, k^i, v, t^i) | t^i \in [t, t + b^j + k^i]\} \quad (1)$$

$$\Omega_j = \{(j, b^j, 0, v, t^j) | t^j \in [t, t + b^i - k^i]\} \quad (2)$$

The first set of constraints forbids the k^i-th cart of agent i (with body length b^i) to occupy vertex v in the time range $[t, t+b^j+k^i]$. The second set of constraints forbids the head vertex of agent j (with body length b^j) to occupy v in the time range $[t, t + b^i - k^i]$.

Intuitively, these constraints leverage the tractor-trailer model in MCPF-TT where each cart follows the path of the former cart. If v is occupied by the k-th cart of agent i at some time t, the agent must occupy v within time range $[t - k^i, t + b^i - k^i]$ ($v \in O^i(\tau), \tau \in [t - k^i, t + b^i - k^i]$), which is called the *occupation interval* of agent i at v and is denoted as $OI(i, v, t) = [t - k^i, t + b^i - k^i]$.

To prove CBS with this new branching rule returns an optimal solution, we need to show the two constraint sets are *mutually disjunctive* [15]: Two sets Ω_i and Ω_j are mutually disjunctive if a conflict-free joint path cannot simultaneously violate any pair of constraints $(\omega^i, \omega^j), \omega^i \in \Omega_i, \omega^j \in \Omega_j$. In other words, if a joint path π simultaneously violates any pair of such constraints, then π must have a conflict.

Theorem 1: In MCPF-TT, constraints (1) and (2) are mutually disjunctive.

Proof: We need to show that, within the time ranges as in constraints (1) and (2), for any pair of t^i, t^j, $t^i \in [t, t + b^j + k^i], t^j \in [t, t + b^i - k^i]$, the two corresponding occupation intervals $OI(i, v, t^i) = [t^i - k^i, t^i + b^i - k^i] = [\tau_a^i, \tau_b^i]$ and $OI(j, v, t^j) = [t^j, t^j + b^j] = [\tau_a^j, \tau_b^j]$ intersects, which indicates a conflict.[1] If $\tau_a^i \in [\tau_a^j, \tau_b^j]$ (or $\tau_a^j \in [\tau_a^i, \tau_b^i]$), then $OI(i, v, t^i) \cap OI(j, v, t^j)$ contains at least τ_a^i (or τ_b^i) and they intersect. Now we only need to consider the cases $\tau_a^i > \tau_b^j$ and $\tau_a^j > \tau_b^i$, and we show that these two cases cannot happen.

For the case $\tau_a^i > \tau_b^j$, due to $t^i \in [t, t + b^j + k^i]$ and $t^j \in [t, t+b^i-k^i]$, we know $\tau_a^i = t^i - k^i \leq t + b^j + k^i - k^i = t + b^j$ and $\tau_b^j = t^j + b^j \geq t + b^j$. Thus, $\tau_a^i \leq t + b^j \leq \tau_b^j$ which contradicts with $\tau_a^i > \tau_b^j$.

For the case $\tau_a^j > \tau_b^i$, due to $t^i \in [t, t + b^j + k^i]$ and $t^j \in [t, t + b^i - k^i]$, we know $\tau_a^j = t^j \leq t + b^i - k^i$ and $\tau_b^i = t^i + b^i - k^i \geq t + b^i - k^i$. Thus, $\tau_a^j \leq t + b^i - k^i \leq \tau_b^i$ which contradicts with $\tau_a^j > \tau_b^i$.

Therefore, if a joint path π simultaneously violates any pair of constraints $(\omega^i, \omega^j), \omega^i \in \Omega_i, \omega^j \in \Omega_j$, π must have a conflict, and constraints (1) and (2) are mutually disjunctive. ■

B. With Task Duration

When task duration are non-zero, we include the task duration into the branching rule. Let $(i, b^i, k^i, j, b^j, k^j, v, t, t_s, t_e)$ denote a *body conflict with task duration*, which means that the k^i-th cart of agent i (with body length b^i) collides with the k^j-th cart of agent j (with body length b^j) at vertex v at time t. Here, agents i and j cannot execute tasks at the same vertex when such a conflict is first detected, since otherwise, another conflict at an earlier time would be detected. Without loss of generality, we assume that agent i is executing a task and t_s and t_e represent the start and end time of agent i's task respectively.

To resolve the body conflict $(i, b^i, k^i, j, b^j, 0, v, t, t_s, t_e)$, for agent i that is executing a task, define a new form of constraint $(i, b^i, k^i, v, t^i, \Delta t)$, which means that when the body length of agent i is b^i, the k^i-th vertex of agent i can not occupy v at time t^i to start a task whose task duration is $\Delta t = t_e - t_s$. Then two sets of constraints can be generated:

$$\Omega_i = \{(i, b^i, k^i, v, t^i, \Delta t) | t^i \in [t_s, t + b^j + k^i]\} \quad (3)$$

$$\Omega_j = \{(j, b^j, 0, v, t^j) | t^j \in [t, t_e + b^i - k^i]\} \quad (4)$$

[1] Note that k^j disappears in $OI(j, v, t^j)$ as $k^j = 0$ (j's head is at v).

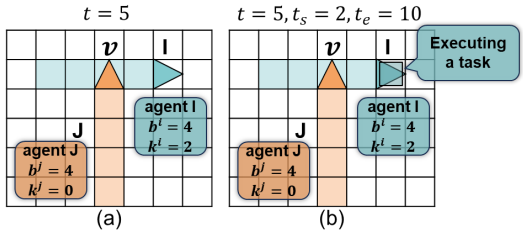

Fig. 4. Toy examples for conflict resolution. (a) shows a body conflict $(i, 4, 2, j, 4, 0, v, 5)$. (b) shows a body conflict $(i, 4, 2, j, 4, 0, v, 5, 2, 10)$, where agent i is executing a task.

The first set forbids the k^i-th cart of agent i (with body length b^i) to occupy vertex v to start a task whose task duration is Δt in the time range $[t_s, t + b^j + k^i]$. The second set forbids the head vertex of agent j (with body length b^j) to occupy v in the time range $[t, t_e + b^i - k^i]$.

It should be noted that if v is occupied by the k-th vertex of agent i at some time t and agent i starts to execute a task whose task duration is Δt at time t, since each cart follows the path of the former cart and agent i can not move during the task execution, the agent must occupy v within time range $[t - k^i, t + \Delta t + b^i - k^i]$ $(v \in O^i(\tau), \tau \in [t - k^i, t + \Delta t + b^i - k])$. So the *occupation interval* of agent i at v $OI(i, v, t)$ is $[t - k^i, t + \Delta t + b^i - k^i]$.

These two sets of constraints are mutually disjunctive, which can be shown in a similar way as aforementioned, and we thus omit the proof to save space.

Theorem 2: In MCPF-TT, constraints (3) and (4) are mutually disjunctive

The edge conflicts in MCPF-TT are similar to MCPF and only occur in the head. Therefore, our resolution strategy is same to CBS, which is mentioned in Sec. III-A

Example 1: For the example (a) in Fig. 4, the constraints for agent i is $\Omega_i = \{(i, 4, 2, v, t^i) | t^i \in [5, 11]\}$ and the constraints for agent j is $\Omega_j = \{(j, 4, 0, v, t^j) | t^j \in [5, 7]\}$. For the example (b) in Fig. 4, the constraints for agent i is $\Omega_i = \{(i, 4, 2, v, t^i, 8) | t^i \in [2, 11]\}$ and the constraints for agent j is $\Omega_j = \{(j, 4, 0, v, t^j) | t^j \in [5, 12]\}$.

C. Relation to the Existing Conflict Resolution Methods

Our method differs from the existing conflict resolution techniques in Multi-Train Path Finding (MTPF) as follows. Specifically, MT-CBS for MTPF [14] resolves a vertex conflict (i, j, v, t) by adding two sets of constraints $\Omega_i = \{v \notin O^i(t)\}$ and $\Omega_j = \{v \notin O^j(t)\}$, respectively, to prohibit any part of agent i and j from occupying vertex a common v at time t. This rule adds multiple constraints over many vertices at the same time step. In contrast, our new branching rules seek to add multiple constraints on the same vertex but over a time interval. We use the rule in [14] as a baseline in the experiments in Sec. VI.

V. Tractor-Trailer Combinatorial A*

For the low-level planning, a naive approach to plan a path for agent i given γ^i is running A* to find a minimum-cost path from one target to the next and concatenate these paths together. We call this approach as *sequential A**. Sequential A* does not guarantee completeness or solution optimality,

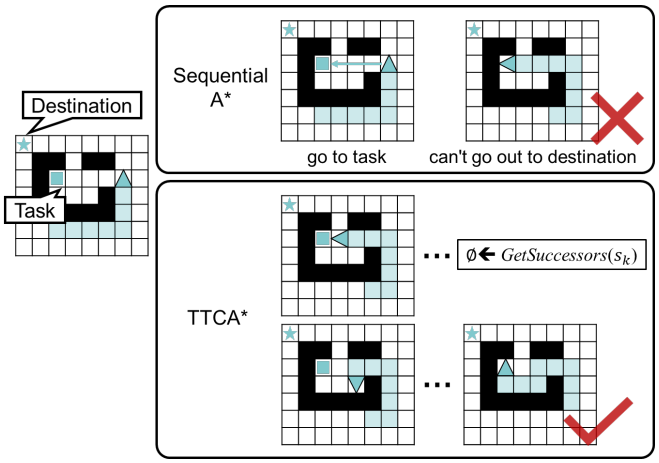

Fig. 5. An example that compares Sequential A* and TTCA*. Sequential A* fails to find any solution while TTCA* finds an optimal solution by searching in a larger state space while considering self-conflicts.

while our TTCA* guarantees completeness and solution optimality. We first illustrate the limitation of sequential A* with the following Example 2, and then present our TTCA*.

Example 2: In Fig. 5, an agent needs to enter a room-like area to for a target, and then exit the room to reach the goal. Sequential A* plans a path that the agent turns left and goes straight to the target, which is an optimal path from the current vertex to that target. However, the agent's body prevents the agent from exiting the room, and sequential A* thus returns no solution. In contrast, for TTCA*, when the agent goes straight to the task vertex, the search can find that the agent can not exit the room when the *GetSuccessor* returns $\emptyset$ for a certain state s. Then, TTCA* can find an alternate state, where the agent makes a detour so that it can exit the room after executing the task. Even if the detour is not an optimal path from the current vertex to the task vertex, it is part of a "global" optimal path for the agent.

A. Algorithm

1) Search State: To avoid self-collision and find an optimal individual path, TTCA* searches over a time-augmented state space that includes the position of the attached carts. Let $s = (g^i, O^i, a^i)$ denote the state of an agent i, which is a tuple of a time step g^i (i.e., the cost of the path), the occupation list O^i at that time, and $a^i \in \mathbb{Z}^+$, a non-negative integer, indicating the number of tasks that has been executed along the path. Let γ^i denote the ordered list of target vertices that are assigned to agent i and $\gamma^i(k)$ denote the k-th target vertices of agent i. If $a^i = 0$, the agent has not yet executed any task, and if $a^i = |\gamma^i|$, the agent has completed all tasks. Both O^i and a^i affects the set of successors of a state due to self-collision avoidance and task execution as explained in the next paragraph.

2) Generate Successors: *GetSuccessor* denotes the procedure in TTCA* that returns the valid successor states of any given state s. The *GetSuccessor* procedure returns the set of all reachable states from the given state s_k. Specifically, if $|O^i| = 1$, the agent has no cart and can move to any adjacent vertex of $v_h^i(O^i)$ or wait in place. Otherwise (i.e., $|O^i| > 1$),

Algorithm 2 Pseudocode for TTCA*

```
1: s_o = (g^i = 0, O^i = {v_o^i}, a^i = 0), f(s_o) ← 0 + MH(s_o)
2: add s_o into OPEN, CLOSED ← ∅
3: while OPEN not empty do                          ▷ Main search loop.
4:     s_k = (g_k^i, O_k^i, a_k^i) ← pop from OPEN
5:     add s_k to CLOSED
6:     if |O_k^i| = |γ^i| + 1 and v_h^i(O_k^i) = v_d^i then
7:         return Reconstruct(s_k)
8:     S_succ ← GetSuccessors(s_k)
9:     for all s' ∈ S_succ do                       ▷ State expansion.
10:        if ConstraintCheck(s') then
11:            continue
12:        if s' ∈ OPEN ∪ CLOSED then
13:            continue
14:        f(s') ← g(s') + MH(s')
15:        parent(s') ← s
16:        add s' to OPEN
17: return Failure
```

the agent can wait in place or move to all adjacent vertices that are not part of O^i, which avoids the self-collision. If the agent's head is next to the next target vertex $\gamma^i(|O^i|+1)$, it can choose to start the task or not (i.e., just pass by). If starting the task, the number of tasks a^i that have been executed is increased by one in the generated successor state. Finally, if a successor $s' = (g^{i'}, O^{i'}, a^{i'})$ is generated before, $i.e.$, $s' \in$ OPEN $\cup$ CLOSED, then s' will be skipped (Line 12 in Alg. 2). Here, CLOSED is a set that contains states that were popped from OPEN and were expanded.

3) Heuristics: To expedite the search, TTCA* uses a heuristic value $h(s)$ that considers the current position of the agent and the remaining tasks to be executed. To avoid confusion with the previous notations, let v_h^i denote the head vertex of the occupation list O^i in a state s in TTCA*, and let $c_\pi(u,v)$ denote the minimum path cost from u to v in the workspace graph G^W. For a state $s = (g^i, O^i, a^i)$, its heuristic is:

$$h(s) = c_\pi(v_h^i, \gamma^i(a^i+1)) + \sum_{k=a^i+1}^{|\gamma^i|-1} c_\pi(\gamma^i(k), \gamma^i(k+1)) \tag{5}$$

Here, the first term estimates the cost-to-go from the current vertex v_h^i to the immediate next target $\gamma^i(k)$ to be visited, the second term estimates the cost to visit all remaining targets in the same order as in γ^i and end at the goal vertex (i.e., the last vertex in γ^i). This heuristic is admissible.

4) Constraint Check: TTCA* plans a path subject to a set of constraints Ω that is added by the high-level search. These constraints include edge constraints (i,e,t) and body constraints as aforementioned. For each successor state $s' = (g^{i'}, O^{i'}, a^{i'})$ of state $s = (g^i, O^i, a^i)$, if $a^{i'} = a^i$, then agent i just move to the next vertex and do not execute a task. In this case, *ConstraintCheck* considers the body constraints of form (i, b^i, k^i, v, t^i) in Ω, and checks if Ω contains any constraint $(i, |O^{i'}|, k, O^{i'}(g^{i'}, k), g^{i'}), k \in [0, |O^{i'}| - 1]$. If so, s' violate a constraint and is thus pruned, and otherwise, s' is generated. If $a^{i'} = a^i + 1$, the agent executes a task. *ConstraintCheck* considers the body constraints with task duration, which are of form $(i, b^i, k^i, v, t^i, \Delta t)$, and checks if Ω contains any of the constraints $(i, |O^{i'}|, k, O^{i'}(g^{i'}, k), g^i, g^{i'} - g^i), k \in [0, |O'| - 1]$. If so, s' is pruned, and otherwise, s' is generated. Finally, edge constraints (i,e,t) are handled in a similar way as in CBS by checking if the transition from s to s' uses any edge e at any forbidden time t.

5) State Selection: When TTCA* plans a path for agent i, it also considers the paths of other agents. TTCA* plans an optimal path for agent i and then seeks to minimize the number of conflicts with other agents as the secondary objective, which can help reduce the number of conflicts to be resolved on the high-level search. Similar ideas can be found in other CBS variants [16], [17]. Specifically, TTCA* sorts its *OPEN* list by two values. The first value is the f-value $f(s) := g(s) + h(s)$, which is same to the regular A*. Using f-value to prioritize states in OPEN helps TTCA* find an optimal path. When two states have the same f-value, then the second value is used, which is the number of conflicts between the path of agent i that is being planned, and the paths of the other agents.

B. Relation to Multi-Label A*

TTCA* is related to the Multi-Label A* (MLA*) [18] that is a low-level single-agent planner for the Multi-Agent Pickup and Delivery (MAPD) problem, where each agent needs to visit both the pickup and delivery position. MLA* employs a binary variable to tell whether the agent is heading to the pickup or the delivery position. TTCA* differs from MLA* in the following aspects. First, TTCA* needs to visit multiple targets as opposed to only two targets (pickup and delivery), and thus uses an integer to indicate the number of visited targets. Second, TTCA* has to consider body length and body positions to avoid self-conflicts. Third, due to the high-level search CBSS-TT and the generated constraints, TTCA* needs to handle constraints that are defined on the agent's body.

VI. Experimental Results

We refer to the conflict resolution method in MTPF [14] as the "Old strategy" and we call our new body conflict resolution method as "New strategy". We compare the following four algorithms. The first one uses the Old strategy on the high-level of CBSS and Sequential A* on the low-level, which is denoted as OS. The second one uses the New strategy on the high-level of CBSS and Sequential A* on the low-level, which is denoted as NS. The third one uses the Old strategy on the high-level of CBSS and TTCA* on the low-level, which is denoted as OT. The fourth one is CBSS-TT, which uses the New strategy on the high-level and TTCA* on its low-level, which is denoted as NT. All algorithms use the same method for target sequencing as in the original CBSS [6].

We use three different maps from an online data set [19]: empty, random, and warehouse as shown on the top of Fig. 6. We test the algorithms by fixing the number of agents $N = 10$ and vary the number of tasks $M = 10, 20, 30, 40$, since

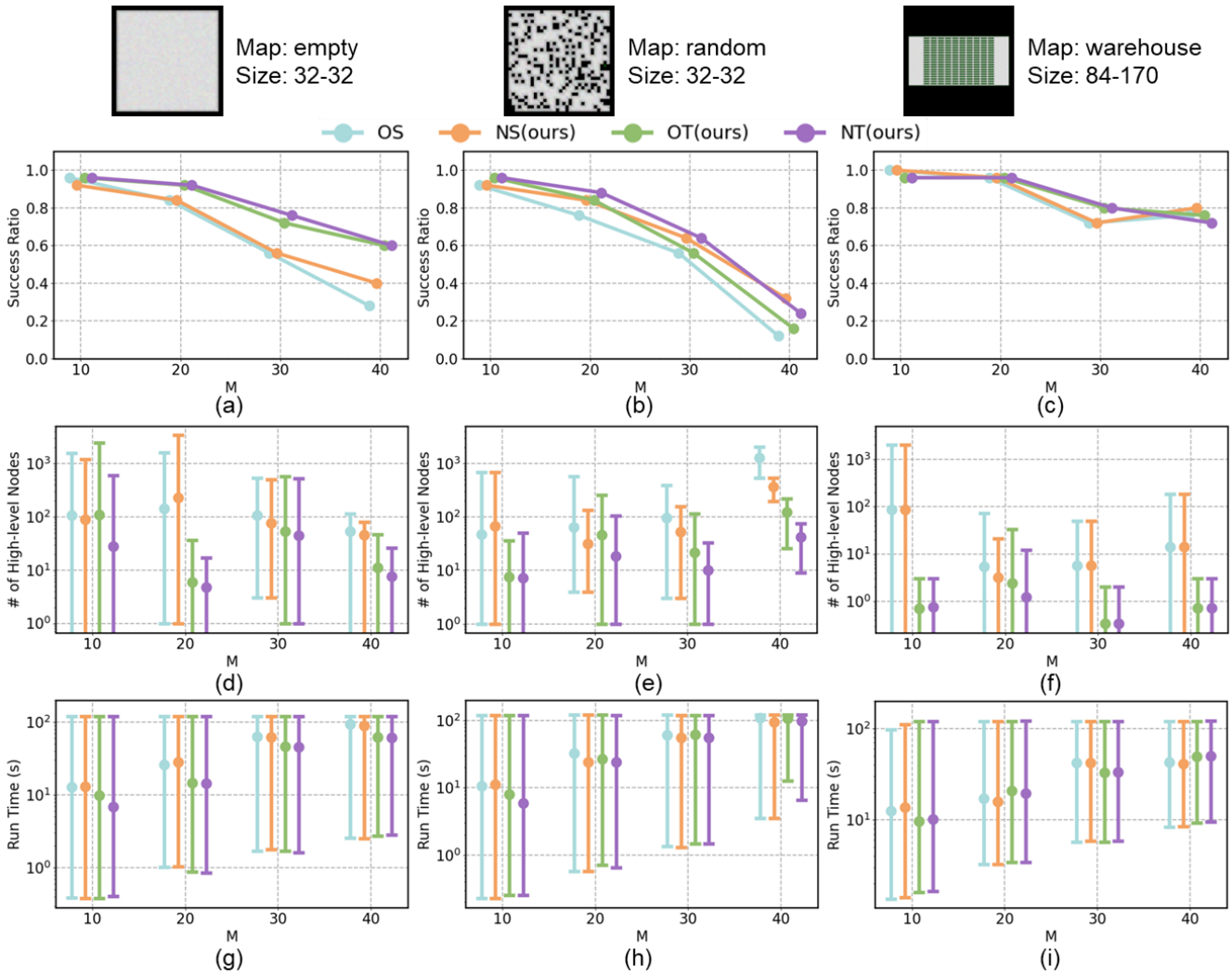

Fig. 6. (a)-(c): the success ratios of OS, NS, OT and NT when the number of agents $N = 10$ and the number of tasks $M = 10, 20, 30, 40$ at different maps. (d)-(f): the minimum, average and maximum number of high-level nodes expanded by OS, NS, OT and NT. (g)-(i): the minimum, average and maximum runtime of OS, NS, OT and NT.

the number of tasks affect the body lengths, which is the main focus of this paper. The runtime limit of each instance is 120 seconds. The task duration of all tasks is 10 time steps. We use a hyper-parameter $\epsilon = 1.0$ in CBSS [6], which tunes the solution sub-optimality bound, and lets the algorithms spend more time on conflict resolution as opposed to target sequencing. We implement all algorithms in Python and run all tests on a computer with an Intel Core i7-11800H CPU and 16G RAM.

A. Success Ratios

Fig. 6 (a)-(c) show the success ratios of the methods. We observe that, the use of TTCA* for the low-level search can often improve the success ratios. For example, in the empty-32-32, OT and NT (both with TTCA* as the low-level) enhances the success ratios up to around 20%. In the random map, due to the cluttered obstacles, some instances become infeasible due to self-conflicts, especially when agents have long body lengths, since there is no way for an agent to access some targets and then leave. As a result, the success rates in the random map are lower than the other two maps. In the large warehouse map, the success ratios fluctuate, and there are also cases where OS and NS (using sequential A* as the low-level) achieves better success rates than OT and NT. A possible reason is that, TTCA* needs to search a larger state space, and as the map size grows, each TTCA* call may take longer time, which thus lowers the overall success rates. This points out a future work direction on improving TTCA* to handle larger maps more efficiently.

B. Number of Expansions

Fig. 6 (d)-(f) shows the number of conflicts that are resolved, i.e., the number of high-level nodes that are expanded. First, the use of TTCA* as the low-level planner helps reduce the number of expansions, and a possible reason is that TTCA* is complete and can find solutions for cases that are otherwise failed to be solved by sequential A*. As a result, OT and NT find solution earlier while OS and NS (using sequential A*) have to expand more nodes before finding a solution. Second, the new branching rule sometimes

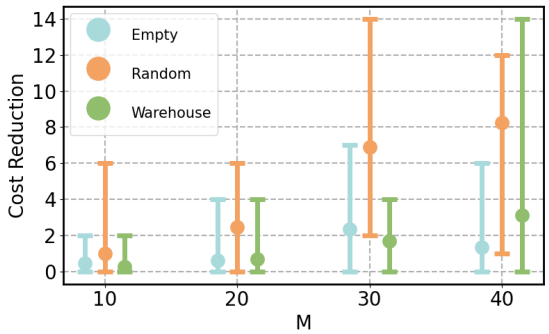

Fig. 7. The cost reduction about NS and NT when the number of agents $N = 10$ and the number of tasks $M = 10, 20, 30, 40$ at different maps. The cost of paths from NT is lower than that from NS. As the tasks increase, the cost reduction gradually increases.

expands fewer nodes than the old branching rule, especially in crowded and complex scenarios that may lead to more conflicts. For example, in the random map, when $M = 40$, comparing to OS and OT, NS and NT require up to 65.6% fewer expansions (reducing 122 expansions down to 42).

C. Runtime

We compare the runtime of all instances in Fig. 6 (g)-(i), and if an algorithm times out, its runtime is 120 seconds. In the empty map, OT and NT usually have shorter runtime than OS and NS, and the reason is that OT and NT reduces the number of high-level nodes expanded as verified in Fig. 6(d) and as discussed in the previous subsection. For the random map, the runtime of all four methods are similar while NT is slightly better than the other three in terms of the average runtime. For the warehouse map, there is no clear trend among the runtimes. The reason is that, although NT and OT have fewer number of expansions than OS and NS as shown in Fig. 6 (f), each expansion in NT and OT takes more time, because TTCA* needs longer runtime per call than sequential A*, especially in this large warehouse map.

D. Solution Quality

Although all methods have the same solution sub-optimality bounds, the actual solution costs differ. As shown in Fig. 7, we measure the cost difference $g(\pi_{NS}) - g(\pi_{NT})$ to compare their path costs The paths from NT is better than that of NS. The maximum reduction reaches 14 time steps. As the tasks increase and the map become more complex, the gap between NS and NT gradually increases. We also compare the cost from OT with the cost from NT. All costs from OT are same to that from NT. It is means that the new strategy can only reduce the number of expansions, while the low-level planenr TTCA* can help find a cheaper solution.

VII. Conclusion and Future Work

This paper investigates a new problem MCPF-TT, and develops new conflict resolution techniques for agents with long bodies and a new single-agent planner for agents with varying body lengths. Experimental results show the advantages of our new approaches in different settings. For future work, one can consider improving TTCA* for large maps, or developing sub-optimal planners for MCPF-TT to handle many agents with many tasks. One can also consider handling delays in task duration for agents with a variable body length.

References

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007.

[2] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[3] M. L. Americas. (2025) Move the world forward. Accessed: 2025-02-20. [Online]. Available: https://www.logisnextamericas.com/en/logisnext

[4] Y. Zhang, X. Wu, H. Wang, and Z. Ren, "Multi-agent combinatorial path finding with heterogeneous task duration," *arXiv preprint arXiv:2311.15330*, 2023.

[5] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5816–5823, 2021.

[6] Z. Ren, S. Rathinam, and H. Choset, "CBSS: A New Approach for Multiagent Combinatorial Path Finding," *IEEE Transactions on Robotics*, vol. 39, no. 4, pp. 2669–2683, 2023.

[7] M. Liu, H. Ma, J. Li, and 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.

[8] K. Brown, O. Peltzer, M. A. Sehr, M. Schwager, and M. J. Kochenderfer, "Optimal sequential task assignment and path finding for multi-agent robotic assembly planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 441–447.

[9] Z. Ren, S. Rathinam, and H. Choset, "A bounded sub-optimal approach for multi-agent combinatorial path finding," *IEEE Transactions on Automation Science and Engineering*, vol. 22, pp. 7590–7605, 2025.

[10] Z. Ren, A. Nandy, S. Rathinam, and H. Choset, "Dms*: Towards minimizing makespan for multi-agent combinatorial path finding," *IEEE Robotics and Automation Letters*, vol. 9, no. 9, pp. 7987–7994, 2024.

[11] Z. Ren, S. Rathinam, and H. Choset, "Ms*: A new exact algorithm for multi-agent simultaneous multi-goal sequencing and path finding," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 11 560–11 565.

[12] Y. Zhang, H. Wang, and Z. Ren, "A short summary of multi-agent combinatorial path finding with heterogeneous task duration," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 17, 2024, pp. 301–302.

[13] D. Atzmon, R. Stern, A. Felner, G. Wagner, R. Barták, and N.-F. Zhou, "Robust multi-agent path finding," in *Proceedings of the international symposium on combinatorial search*, vol. 9, no. 1, 2018, pp. 2–9.

[14] D. Atzmon, A. Diei, and D. Rave, "Multi-train path finding," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, no. 1, 2019, pp. 125–129.

[15] Z. Chen, J. Li, D. Harabor, P. J. Stuckey, and S. Koenig, "Multi-train path finding revisited," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 15, no. 1, 2022, pp. 38–46.

[16] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[17] J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Artificial Intelligence*, vol. 301, p. 103574, 2021.

[18] F. Grenouilleau, W.-J. Van Hoeve, and J. N. Hooker, "A multi-label a* algorithm for multi-agent pathfinding," in *Proceedings of the international conference on automated planning and scheduling*, vol. 29, 2019, pp. 181–185.

[19] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *arXiv preprint arXiv:1906.08291*, 2019.