From Agent Centric to Obstacle Centric Planning: A Makespan-Optimal Algorithm for the Multi-Agent Warehouse Rearrangement Problem

Yaakov Sherma, Eyal Weiss, Oren Salzman

Technion, Israel Institute of Technology
yaakovsherma@campus.technion.ac.il, eweiss@campus.technion.ac.il, osalzman@cs.technion.ac.il

Abstract

The Multi-Agent Warehouse Rearrangement (MAWR) problem calls for computing agents plans such that they collectively rearrange a warehouse environment from a given *layout* which species the location of every movable obstacle in the environment, to a goal layout. It is a natural variant of the well-studied Multi-Agent Path Finding (MAPF) and Multi-Agent Pickup and Delivery (MAPD) problems, which have numerous applications in warehouse automation. Similar to MAPF and MAPD, the problem is computationally hard and existing methods forgo any optimality guarantees while computing solutions to the problem. In contrast, in this work we present the first fully-coupled search-based *makespan-optimal* approach for solving the MAWR problem. This is done by shifting the algorithmic viewpoint from being *agent centric* to *obstacle centric*: Common MAPF and MAPD algorithms are agent-centric wherein tasks are assigned to agents, and the agents' paths are planned to fulfill these tasks. In contrast, the approach we present here is obstacle-centric: Our algorithm iterates between two phases: **Ph1** planning optimal paths for the movable obstacles using an adaptation of the celebrated CBS MAPF algorithm and **Ph2** attempting to realize the movable obstacles paths using a network-flow algorithm. We prove that our approach guarantees minimal-makespan solutions and empirically demonstrate that it achieves better plans than state-of-the-art approaches for MAWR.

1 Introduction

In the Multi-Agent Warehouse Rearrangement (MAWR) problem, also known as DD-MAPD (Li and Ma 2023) or MAT (Bachor, Bergdoll, and Nebel 2023), we are given an initial warehouse *layout* $\mathcal{L}_{\text{init}}$ which specifies the position of a set of *movable* obstacles in the warehouse, as well as the initial positions of agents capable of moving the obstacles. Given a goal layout $\mathcal{L}_{\text{goal}}$, the problem calls for planning collision-free paths for the agents tasked with relocating the obstacles from $\mathcal{L}_{\text{init}}$ to $\mathcal{L}_{\text{goal}}$. This is called a *rearrangement* and the MAWR problem calls for finding the rearrangement with the minimal *makespan*—the overall time required to complete the rearrangement (Fig. 1, top).

This problem is a natural variant of the well-studied Multi-Agent Path Finding (MAPF) (Sharon et al. 2015; Wagner and Choset 2015; Stern et al. 2019) and Multi-Agent Pickup and Delivery (MAPD) (Ma et al. 2017, 2019; Liu et al. 2019; Salzman and Stern 2020) problems, all of which have numerous applications in warehouse automation (Wurman, D'Andrea, and Mountz 2008). Generally speaking, the former calls for computing collision-free paths for a group of agents from their start to goal location while the latter calls for agents to complete a stream of tasks, where each task is composed of a pick-up location and a delivery location.

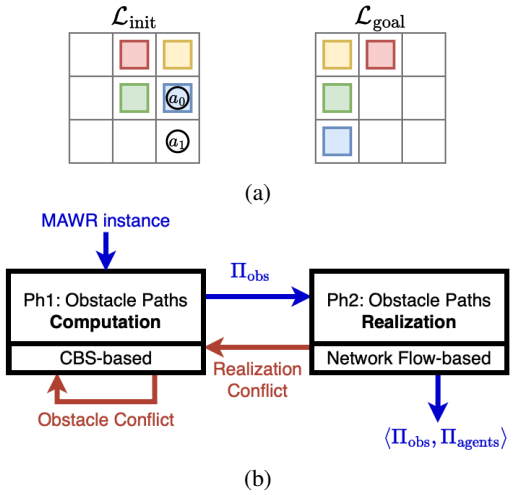


Figure 1: (a) Initial and goal warehouse layouts. (b) NAT-CBS algorithmic framework.

MAWR was recently shown by Li and Ma (2023) to generalize MAPF, and thus to be NP-hard. To tackle the computational complexity of MAWR, they suggested MAPF-DECOMP, a decoupled approach wherein first plans for the obstacles are computed. Then, plans for the agents to move the obstacles according to their plans are computed. This decoupled approach is extremely fast in practice but offers no guarantees on solution quality. Importantly, it can easily be shown that any algorithm that follows the decoupled approach can not be optimal. Bachor, Bergdoll, and Nebel (2023) showed that MAWR is NP-complete and suggested a makespan-optimal, reduction-based algorithmic solution to solve the problem. In contrast, we present the first fully-coupled search-based approach approach for solving

the MAWR problem which is *makespan-optimal*.

Our key insight, motivating this work, is shifting the algorithmic viewpoint from being *agent centric* to *obstacle centric*: Common MAPF and MAPD algorithms (see, e.g., Hönig et al. 2018; Tang et al. 2023; Madar, Solovey, and Salzman 2022; Kottinger et al. 2024, for a very partial list) are agent-centric: tasks are assigned to agents, and the agents' paths are planned to fulfill these tasks. Each task is carried out by a single agent, a property we dub as *atomic*. In contrast, the approach we present here is obstacle-centric: Our algorithm, called Non-Atomic Task Conflict-Based Search (NAT-CBS), first plans optimal paths for the obstacles and then assigns agents to realize these paths. As a side effect, a task can be completed by multiple agents moving an obstacle along its path, a property we refer to as *non-atomic*. If an obstacle's path cannot be realized, the algorithm identifies the specific movement causing the issue and replans the obstacle's path accordingly.

More specifically and as visualized in Fig. 1, NAT-CBS optimally solves the MAWR problem by building upon the optimal CBS for MAPF (Sharon et al. 2015). NAT-CBS treats obstacles as agents and computes conflict-free plans for them. Subsequently, it computes paths for agents to execute the obstacles' movements by solving a network-flow problem on a time-expanded graph, similar to Ma and Koenig (2016). If the plan is realizable, the algorithm returns paths for both agents and obstacles. If not, it identifies at least one obstacle movement that cannot be executed by an agent. Algorithmically, this is modeled as a new CBS-type conflict which enforces the search algorithm to plan paths that account for this conflict. We term this a *realization conflict*, representing a mismatch between the planned obstacles' movements and the agents' ability to execute them. As will be explained, this conflict is resolved by adding constraints on the obstacle's potential paths. The algorithm iterates until a fully realizable plan is found.

We formally prove that NAT-CBS is guaranteed to return a plan with the minimal makespan for all solvable instances. Finally, we conduct experiments that empirically validate that NAT-CBS achieves optimality, and we evaluate the scalability of our algorithm on randomly generated warehouse rearrangement instances, with varying numbers of agents and obstacles.

2 Related Work

Recall that MAWR, formally defined in Section 3, is a natural variant of the MAPF and MAPD problems. However, it is important to emphasize unique properties of MAWR when compared to MAPF and MAPD as well as key differences between these problems: (i) In MAWR, the number of obstacles that need to be moved can be smaller, equal or larger than the number of agents, in contrast to typical MAPF formulations where they are equal; (ii) in MAWR, the initial and final layouts are known in advance, in contrast to typical MAPD formulations which deal with solving a sequence, or stream, of task-assignment and MAPF instances given in an online fashion; and (iii) the start and goal positions of the obstacles can be arbitrary, in contrast to typical MAPD formulations in which either the pickup or the delivery locations are chosen from a small subset of packing stations. Finally, a unique MAWR trait is that agents actively manipulate the layout, rather than simply navigating in it as in MAPF.

There are various algorithmic approaches to solving MAPF and MAPD, which primarily include search-based methods (e.g., (Wagner and Choset 2015; Li et al. 2021; Li, Ruml, and Koenig 2021)), and reduction-based methods (e.g., (Lam et al. 2022; Gómez, Hernández, and Baier 2020)). In this work we build on the CBS framework (which is a search-based method) that has proved very effective, and extend it to our setup to solve MAWR.

The closest line of research to ours is a recent paper (Li and Ma 2023) that introduced DD-MAPD. While considering the same problem, their algorithmic approach is different: they present MAPF-DECOMP which works in two steps. First, it calls an off-the-shelf MAPF planner to find a collision-free plan (a set of trajectories) for the obstacles. Second, it constructs a dependency graph that represents the obstacles plan, and partitions each trajectory into segments which can be executed by agents. Lastly, it solves a MAPD instance with dependencies – reflected by the dependency graph – by assigning obstacle segments to agents and computing collision-free paths for the agents that do not carry obstacles (free agents). In contrast, our approach computes an obstacles plan and an agents plan in a coupled manner. As mentioned before, this allows to provide quality guarantees. From an algorithmic standpoint, this is advantageous due to more degrees of freedom to co-optimize the plans. However, this renders the problem harder to solve. Indeed our empirical results reflect significantly higher runtime.

In the rest of this section we briefly mention MAPF and MAPD variants which, similar to our work, require coordination between agents that goes beyond mere collision avoidance. Anonymous MAPF (A-MAPF) is a variant of MAPF where goal locations are not preassigned to agents, with the number of agents matching the number of goals. Any agent can reach any goal as long as all goals are occupied. Unlike standard MAPF, A-MAPF can be solved efficiently in polynomial time through a reduction to a network-flow problem (Yu and LaValle 2012). In A-MAPF, all agents are interchangeable, meaning any agent can reach any goal. This mirrors the nature of MAWR, where agents are interchangeable, and any agent can move any obstacle.

The Multi-Agent Meeting problem (Atzmon et al. 2023) calls for finding a common, meeting location for all agents as well as a set of conflict-free paths for them to reach it, while minimizing the plan's total cost. The Cooperative MAPF (Co-MAPF) problem (Greshler et al. 2021) is an extension of MAPF where tasks need to be completed by pairs of agents each with its own capability—an initiator agent needs to pick up an obstacle while an executer agents needs to drop said object. In order to complete the task the object needs to be transferred between the agents. While both of these problems are inherently cooperative, their coordination is more structured and predefined. In contrast, MAWR offers a more flexible form of cooperation. All agents have the same capabilities, and the extent of their collaboration is not predetermined. A task may be completed by a single agent, by all agents working together, or through any level

of cooperation in between.

A MAPD variant that allows for non-trivial coordination is Terraforming MAPD (tMAPD) (Vainshtein et al. 2023) which extends MAPD by permitting environment manipulation. In tMAPD agents that are not carrying an obstacle can modify the environment while agents that do carry obstacles can only drop off obstacles at their designated goal locations. Thus, in contrast to our setting, tasks remain atomic, i.e., each must be completed by a single agent.

3 Problem Formulation

We define an MAWR problem instance to be a tuple $P = \langle G, A, O, s, \mathcal{L}_{\textbf{init}}, \mathcal{L}_{\textbf{goal}} \rangle$, where $G = (V, E)$ represents the environment (undirected) graph; $A = \{a_1, \ldots, a_n\}$ denotes a set of n agents; $O = \{o_1, \ldots, o_m\}$ denotes a set of m movable obstacles; and $s : A \to V$ an injective function defining the initial agent locations. Given a graph $G = (V, E)$, we define a layout $\mathcal{L} : O \to V$ as an injective function specifying the obstacles' locations in the environment and $\mathcal{L}_{\textbf{init}}$ and $\mathcal{L}_{\textbf{goal}}$ to be initial and goal layouts, respectively.

Time is discretized and at each timestep, an agent can stay at its current location or move along an edge to an adjacent location. An agent can share its location with an obstacle as it moves underneath it. An agent can also pick up an obstacle if it is at the obstacle's current location; subsequently, the obstacle moves with the agent until the agent sets it down. Both the pick-up and set-down actions are instantaneous and take no time. In other words, obstacles can move along an edge to an adjacent location only if accompanied by an agent; otherwise, they remain in place.

A path $\pi = (v_0, v_1, \ldots, v_k)$ is a sequence of locations $v_i \in V$, representing agent or obstacle movements. We define the length $|\pi|$ of the path as the number of time steps in π (i.e, $|\pi| := k$). For any path to be *valid*, we require that for every i either $v_{i-1} = v_i$ or $(v_{i-1}, v_i) \in E$. For an agent path $\pi_{a_i} = (v_0, v_1, \ldots, v_k)$ of agent $a_i \in A$ to be valid, we also require that the agent starts at its initial location (i.e., $v_0 = s(a_i)$). Similarly, for an obstacle path $\pi_{o_i} = (v_0, v_1, \ldots, v_k)$ of obstacle $o_i \in O$ to be valid, we also require that the path obstacle starts at its location in the initial layout and ends at its location in the goal layout (i.e., $v_0 = \mathcal{L}_{\textbf{init}}(o_i)$ and $v_k = \mathcal{L}_{\textbf{goal}}(o_i)$).

Since agents can move under obstacles, there are two conceptual height levels: an "obstacle level" for the obstacles and a "agent level" for the agents. For each height level, we can define what happens when agent paths or obstacle paths overlap at the same timestep. An event defined as a *conflict*. Specifically, given two paths (either of agents or of obstacles), a conflict occurs if they overlap at the same timestep. Conflicts can be categorized into two types: (i) a *vertex conflict* $\langle a_i, a_j, v, t \rangle$ where agents a_i and a_j occupy the same location v at timestep t (defined analogously for obstacles), and (ii) an *edge conflict* $\langle a_i, a_j, u, v, t \rangle$ where agents a_i and a_j traverse the same edge (u, v) in opposite directions at timestep t (defined analogously for obstacles).

An obstacles plan $\Pi_{\text{obs}} = \langle \pi_{o_1}, \ldots, \pi_{o_m} \rangle$ is a vector of m obstacle paths, and an agents plan $\Pi_{\text{agents}} = \langle \pi_{a_1}, \ldots, \pi_{a_n} \rangle$ is a vector of n agent paths. An obstacles plan Π_{obs} and an agents plan Π_{agents} are said to be *collision free* if there are no conflicts between obstacles and agents in Π_{obs} and Π_{agents}, respectively.

A plan is defined as $\Pi = \langle \Pi_{\text{obs}}, \Pi_{\text{agents}} \rangle$. Given a plan Π, we denote by $\pi_{a_i}(t)$ and $\pi_{o_j}(t)$ the locations of agent a_i and obstacle o_j at time t, respectively and say that Π_{agents} *realizes* Π_{obs} if for every movement of an obstacle along an edge there is an agent that goes along the same edge at the same time. I.e., $\forall o_j \in O, t \in \mathbb{N}, \pi_{o_j}(t) = \pi_{o_j}(t+1)$ or $\exists a_i \in A$, s.t. $\big(\pi_{a_i}(t) = \pi_{o_j}(t) \wedge \pi_{a_i}(t+1) = \pi_{o_j}(t+1)\big)$. A plan Π is said to be valid if (i) all obstacle and agent paths are valid, (ii) Π_{obs} and Π_{agents} are collision-free, and (iii) Π_{obs} is realized by Π_{agents}. A plan's makespan is the maximum of all agent- and obstacle-paths: $\text{MKSP}(\Pi) := \max_{\pi \in \Pi} |\pi|$. A *solution* for the MAWR problem is a valid plan Π. An *optimal solution* to an MAWR problem is a valid plan with a minimal makespan.

4 Algorithmic Background

4.1 Conflict-Based Search

Conflict-Based Search (CBS) (Sharon et al. 2015; Li et al. 2019; Gordon, Filmus, and Salzman 2021) is a well-known optimal MAPF solver that executes a best-first search on a so-called *Constraint Tree* (CT). Each node in the CT contains a set of *constraints* and paths for all agents that adhere to those constraints. Most common instantiations of CBS define positive and negative constraints that can be associated with either a vertex or and edge. Specifically, a positive or negative vertex constraint $(\pm, a, v, t)$ specifies that agent a is either constrained to or prohibited from occupying vertex v at timestep t, respectively. Similarly, a positive or negative edge constraint $(\pm, a, u, v, t)$ specifies that agent a is either constrained to or prohibited from moving along edge $e = (u, v)$ from vertex u to v at timestep t.

The root node of the CT is associated with an empty set of constraints, thus initially each agent takes the shortest path to its goal, ignoring other agents. The root node is inserted to OPEN, a priority queue ordered by the plan's total cost. CBS then repeatedly takes a node $\mathcal{N}$ from OPEN, if there is a conflict between two agents, two child nodes are created, inheriting all constraints from node $\mathcal{N}$, and in one of them imposing a positive constraint and on the other a negative constraint, effectively resolving the conflict. After replanning the necessary paths, new child nodes are inserted into OPEN. If we reach a node where there are no conflicts, a solution is found and returned.

4.2 Network Flow

The Min-Cost Max-Flow (MCMF) problem is a classic optimization problem in network-flow theory. Given a directed graph with capacities and costs on each edge, the goal is to find the maximum flow from a designated source to a sink while minimizing the total cost of the flow. The cost of a flow is determined by the sum of the products of the flow value on each edge and its corresponding cost. MCMF extends the maximum flow problem by introducing edge costs, and has been widely studied, with numerous algorithms proposed for its solution. Fundamental approaches include the

Successive Shortest Path, Cycle-Canceling, and Cost Scaling algorithms, each with many variations and adaptations. For an overview of network-flow problems and algorithms, see, e.g., (Ahuja, Magnanti, and Orlin 1993).

A variant of MCMF, relevant to our work, introduces lower bounds on edges, requiring each edge to carry a minimum flow in addition to an upper bound capacity. This formulation can model scenarios where a guaranteed level of service or commitment is required. To handle lower bounds, the problem can be transformed by adjusting capacities and introducing so-called "demand nodes" or constructing auxiliary networks. These adjustments ensure that any feasible solution respects the lower bounds while maintaining the standard max-flow constraints. Solving this variant typically involves modifying the flow network and applying traditional MCMF algorithms to the adjusted network.

5 Algorithmic Framework

To solve the MAWR problem we take an *obstacle centric* approach where we continuously iterate between two phases which we call obstacle-path computation (phase **Ph1**) and obstacle-path realization (phase **Ph2**), respectively.

In **Ph1** we use the CBS framework to plan optimal paths for the obstacles by introducing obstacle constraints making sure that there are no obstacle-obstacle conflicts. Once such a path Π_{obs} is computed, we move to **Ph2** to plan agent paths in order to realize Π_{obs} using a network flow-based approach. As a side effect of this approach, a task can be completed by multiple agents moving an obstacle along its path, a property we refer to as *non-atomic*. If Π_{obs} can be realized using agent paths Π_{agents}, then Π_{obs} and Π_{agents} constitute an optimal solution the MAWR query. If not, a new type of conflict which we term a *realization conflict* is added to the CT of the CBS-like algorithm which constrains obstacle movements. See Fig. 1 for a visualization.

Importantly, **Ph1** is concerned with where *each* obstacle moves while **Ph2** is concerned that *an* agent realizes each obstacle motion. Roughly speaking, these correspond to the notions of classical MAPF in which agents must be relocated to a specific, preassigned location and anonymous MAPF in which agents must be relocated to any location among a set of goals. Thus, as algorithmic building blocks, we adapt tools developed for standard and anonymous MAPF (specifically, CBS and network flow, respectively).

We now continue to detail in Sec. 5.1 and 5.2 phase **Ph1** and **Ph2**, respectively.

5.1 Phase Ph1—Obstacle-Path Computation

Similar to CBS, our algorithm performs a best-first search over a so-called constraint tree (CT). Each node $\mathcal{N}$ of the CT contains (i) a set of *obstacle constraints*, (ii) a set of *obstacle paths*, and (iii) the *cost* of the (possibly invalid) solution associated with $\mathcal{N}$.

An obstacle constraint can be introduced due to an *obstacle conflict* or due to a *realization conflict*. Specifically, an obstacle conflict $\mathcal{C}_{\text{obs}}$ is a vertex or edge conflict due to the obstacles' paths associated with a node. This is identical to the conflicts used by CBS (Sec. 4.1). A realization conflict $\mathcal{C}_{\text{real}}$ is a new concept which is a tuple $\langle o_i, u, v, t\rangle$, representing an obstacle $o_i \in O$ moving along edge $(u, v) \in E$ from u to v at timestep t, without an agent moving it. Realization conflict is our key mechanism to adjust an obstacles plan in order to make it realizable by agents. While we begin the search with a root node that contains an optimal obstacles plan (in terms of makespan), this plan may not be directly realizable by the agents. In such a case, when detected by phase **Ph2** that the current obstacles plan cannot be realized, it returns a realization conflict of one obstacle action that could not be realized.

Algorithm 1: High-level of NAT-CBS

```
   Input: MAWR instance P
   Output: Makespan-optimal plan Π for P
1  Π_obs ← Optimal obs. paths          ▷ no obs. coordination
2  R ← ⟨∅, Π_obs, MKSP(Π_obs)⟩         ▷ no obs. constraints
3  OPEN ← {R}
4  while OPEN is not empty do
5      N ← OPEN.pop()                  ▷ CT node with min cost
6      if N.Π_obs has obs conflict C_obs then
7          N_children ← split(N, C_obs)
8          foreach N' ∈ N_children do OPEN.insert(N')
9          continue
10     Π_agents ← realize_obs_paths(N)
11     if Π_agents = ∅ then
12         continue                    ▷ Can't satisfy N's constraints
13     if (N.Π_obs, Π_agents) has real. conflict C_real then
14         N_children ← split(N, C_real)
15         foreach N' ∈ N_children do OPEN.insert(N')
16     else
17         return Π = ⟨N.Π_obs, Π_agents⟩
18 return not found
```

We are now ready to describe NAT-CBS (our adaptation of CBS to solve MAWR) whose pseudocode is outlined in Alg. 1. The algorithm starts by creating the root node $\mathcal{R}$, in which the set of constraints is empty and each obstacle takes the shortest path from its location in $\mathcal{L}_{\textbf{init}}$ to its location in $\mathcal{L}_{\textbf{goal}}$. Node $\mathcal{R}$ is then inserted to a priority queue OPEN, ordered by nodes' costs. (Lines 1-3).

At each iteration, the algorithm takes a node with the lowest cost out of OPEN (Line 5) and checks if it is conflict-free, i.e., if it contains no obstacle conflicts. If there is an obstacle conflict, we *split* the node into two child nodes—one with a corresponding positive constraint and the other with a corresponding negative constraint (Lines 6-8).

If no obstacle conflicts remain, i.e., a conflict-free obstacles plan Π_{obs} is found, we proceed to find paths for agents that realize the plan (Line 10). Specifically, for the agents to realize Π_{obs} for a CT node $\mathcal{N}$, an agents plan Π_{agents} must satisfy two requirements (**R1**) Π_{agents} should be a valid plan that satisfies $\mathcal{N}$'s positive edge constraints and (**R2**) Π_{agents} should realize obstacle move actions in Π_{obs}. Importantly, if requirement **R1** can not be satisfied, then the node $\mathcal{N}$, is dis-

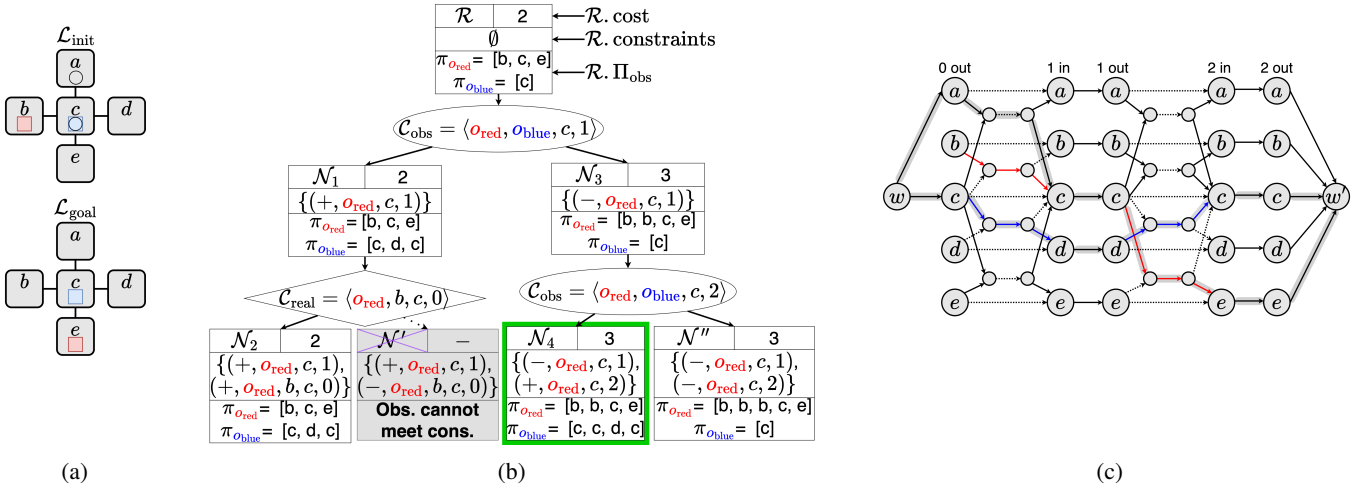


Figure 2: (a) A toy MAWR instance with two obstacles o_{red} and o_{blue} and two agents a_0, a_1 with the initial and goal layouts at the top and bottom, respectively. (b) The CT that NAT-CBS constructs while solving the problem with the solution CT node is marked in green (c): The 2-step time-expanded graph of node $\mathcal{N}_1$ of the CT, where the move actions of o_{red} and o_{blue} are marked in red and blue, respectively; edges depicted with solid lines have cost 0, while those with dotted lines have cost 1; and the resulting flow is illustrated by a gray background, consisting of two paths from w to w'.

carded (Lines 11-12). If requirement **R2** can not be satisfied, this is due to some realization conflict. Consequently $\mathcal{N}$ is split into two child nodes—one with a corresponding positive constraint, ensuring future agents plans are obligated to realize the move, and the other with a corresponding negative constraint, ensuring that future obstacles plans will avoid the move (Lines 13-15). If a valid agents plan Π_{agents} that realizes Π_{obs} is found, we terminate (Line 17).

Toy example. Consider the MAWR instance given by the initial and goal layouts in Fig. 2a with the corresponding CT expanded during the run of NAT-CBS shown in Fig. 2b. The root node $\mathcal{R}$ contains an obstacle vertex conflict (paths $\pi_{o_{\text{red}}}$ and $\pi_{o_{\text{blue}}}$ of obstacle o_{red} and o_{blue} intersect at vertex c at timestep 1 inducing a corresponding obstacle conflict). Consequently, $\mathcal{R}$ is split into nodes $\mathcal{N}_1$ with a positive vertex constraint and $\mathcal{N}_3$ with a negative vertex constraint. Node $\mathcal{N}_1$ contains no obstacle conflicts, so phase **Ph2** is invoked (to be explained shortly), returning a realization conflict. As a result, $\mathcal{N}_1$ is split into two child nodes, $\mathcal{N}_2$ and $\mathcal{N}'$. Node $\mathcal{N}_2$ is inserted to OPEN. For node $\mathcal{N}'$, however, there is no possible path for obstacle o_{red}, and thus it is not inserted to OPEN. Next, when node $\mathcal{N}_2$ is taken out of OPEN, obstacle path realization fails because the agents cannot satisfy the positive edge constraint (again, we elaborate on this shortly), and the node is discarded. Next, node $\mathcal{N}_3$ is taken out of OPEN, and since it contains an obstacle conflict (paths $\pi_{o_{\text{red}}}$ and $\pi_{o_{\text{blue}}}$ intersect at vertex c at timestep 0), it is split into nodes $\mathcal{N}_4$ and $\mathcal{N}''$ which are inserted to OPEN. In the next iteration, node $\mathcal{N}_4$ is taken out of OPEN, and after finding no obstacle conflicts, we invoke obstacle path realization that returns valid Π_{agents} that realizes $\mathcal{N}_4.\Pi_{\text{obs}}$. Hence, we found a solution.

5.2 Phase Ph2—Obstacle-Path Realization

Recall that when NAT-CBS reaches a node $\mathcal{N}$ with collision-free obstacles plan $\mathcal{N}.\Pi_{\text{obs}}$, phase **Ph2** is invoked with $\mathcal{N}$ as an input. Specifically, phase **Ph2** is tasked with computing agents plan Π_{agents} that realizes $\mathcal{N}.\Pi_{\text{obs}}$ (Line 10 in Alg. 1) by satisfying requirements **R1** (plan validity satisfying positive edge constraints of $\mathcal{N}$) and **R2** (realizing Π_{obs}). Thus, to compute Π_{agents}, we use a reduction to a network-flow problem which is similar (though not identical) to network-flow problems used in previous work on MAPF (Yu and LaValle 2012; Ma and Koenig 2016).

T-step time-expanded graph. Let T be the makespan of $\mathcal{N}.\Pi_{\text{obs}}$. Now, given T and the undirected environment graph $G = (V, E)$, we construct a T-step time-expanded directed graph $\mathcal{G}^T = (\mathcal{V}^T, \mathcal{E}^T)$. The vertices $\mathcal{V}^T$ of $\mathcal{G}^T$ are defined to be $\mathcal{V}^T := \{w, w'\} \cup \mathcal{V}^T_{\text{out}} \cup \mathcal{V}^T_{\text{in}} \cup \mathcal{V}^T_{\text{edges}}$. Here, w and w' are new source and sink vertices, respectively, and the sets $\mathcal{V}^T_{\text{out}}$, $\mathcal{V}^T_{\text{in}}$ and $\mathcal{V}^T_{\text{edges}}$ are defined as:

$$\mathcal{V}^T_{\text{out}} = \{v^{\text{out}}_t \mid \forall v \in V, 0 \le t \le T\},$$

$$\mathcal{V}^T_{\text{in}} = \{v^{\text{in}}_t \mid \forall v \in V, 1 \le t \le T\},$$

$$\mathcal{V}^T_{\text{edges}} = \bigcup_{\substack{e \in E \\ 0 \le t \le T-1}} \{e^{\text{in}}_t, e^{\text{out}}_t\}.$$

For each undirected edge $e = (u, v) \in E$ we add the following set of directed edges to $\mathcal{E}^T$, that represents a move action along the edge e at timestep t, and which ensures that there are no edge conflicts:

$$\{(u^{\text{out}}_t, e^{\text{in}}_t), (v^{\text{out}}_t, e^{\text{in}}_t), (e^{\text{in}}_t, e^{\text{out}}_t), (e^{\text{out}}_t, v^{\text{out}}_{t+1}), (e^{\text{out}}_t, u^{\text{out}}_t)\}.$$

Moreover, for each vertex $v \in V$ we add the directed edges: $(v^{\text{out}}_t, v^{\text{in}}_{t+1})$ which represents a wait action at timestep t, and

$(v_t^{\text{in}}, v_t^{\text{out}})$ which ensures there are no vertex conflicts. Finally, we add edges (w, v_0^{out}) from the source to every $v \in V$ which is a starting point of an agent, and edges (v_T^{out}, w') from every location at timestep T to the sink.

Network-flow problem. We construct a Min-Cost Max-Flow with Lower-Bounds problem on the graph $\mathcal{G}^T$, using the obstacles plan $\mathcal{N}.\Pi_{\text{obs}}$ and the set of obstacles' positive edge constraints $\mathcal{N}$.constraints. We set the capacity (upper bound) of each edge to 1. Initially, all edges lower bounds are set to 0. For each positive edge constraint $\langle o_i, u, v, t\rangle$ in $\mathcal{N}$.constraints, we update the lower bound of edges $(u_t^{\text{out}}, e_t^{\text{in}}), (e_t^{\text{in}}, e_t^{\text{out}}), (e_t^{\text{out}}, v_{t+1}^{\text{in}})$ to 1. This ensures that one of the agents' paths would go through u to v at timestep t. Finally, we initially set the cost of the edges $(v_t^{\text{out}}, v_{t+1}^{\text{in}}), (e_t^{\text{in}}, e_t^{\text{out}})$ to 1, and the cost of the rest of the edges to 0. Then, for each move action in $\mathcal{N}.\Pi_{\text{obs}}$ of an obstacle from u to v at timestep t, along edge $e = (u, v)$, we update the cost of the edge $(e_t^{\text{in}}, e_t^{\text{out}})$ to 0, and the costs of $(u_t^{\text{out}}, e_t^{\text{in}}), (e_t^{\text{out}}, v_{t+1}^{\text{in}})$ to 1. We set a supply of n at w and demand of n and w'. A feasible solution of this MCMF problem consists of n disjoint paths from w to w' that represent the n agents' paths.

We run an MCMF algorithm to solve the problem, specifically, we use a cost-scaling push-relabel algorithm (Goldberg and Tarjan 1987; Goldberg and Kharitonov 1991). We extract from the solution the agents plan Π_{agents}. If Π_{agents} realizes Π_{obs}, we found a solution to the MAWR problem. If not, then we have a realization conflict (possibly more than one) and we return to phase **Ph1**. Note that edge costs are leveraged to guide the agents in order to realize as many obstacles' move actions as possible. Further, as in anonymous MAPF, note that the specific identity of obstacles is not used in the MCMF problem construction. This ensures the realization of obstacle move actions without explicitly identifying the moving obstacle, allowing, in theory, any agent to realize any move.

Lastly, the MCMF problem might not have a feasible solution at all, due to the positive obstacles edge constraints, which are translated to flow lower bound constraints. In this case we return that no agent paths found. Note that edge costs alone are insufficient to ensure that agents realize the positive edge constraints. Therefore, we leverage flow lower bounds to enforce that these specific actions are realized and to detect infeasibility when realization is not possible.

Toy example continued. Fig. 2c shows the 2-step time-expanded graph of node $\mathcal{N}_1$ in Fig. 2b. There is a move action of o_{red} at timestep 0 that is not realized, and thus the realization conflict $\mathcal{C}_{\text{real}} = \langle o_{\text{red}}, b, c, 0\rangle$ is returned.

As previously explained, node $\mathcal{N}_2$ results in an obstacle path realization failure. This corresponds to the relevant MCMF problem not having a feasible solution. Node $\mathcal{N}_2$ is created after resolving the above realization conflict by adding the positive edge constraint $\langle +, o_{\text{red}}, b, c, 0\rangle$. Here, the constructed MCMF problem is almost the same as the one of $\mathcal{N}_1$ (depicted in Fig. 2c). The difference lies in edges $(b_0^{\text{out}}, e_0^{\text{in}}), (e_0^{\text{in}}, e_0^{\text{out}}), (e_0^{\text{out}}, c_1^{\text{in}})$ for $e = (b, c)$. For these edges, we set the flow lower bound to 1, to make sure an agent realizes the move action. However, now the flow problem does not have a feasible solution.

Moreover, in phase **Ph2** we actually construct a ***reduced*** T-step time-expanded graph. The construction follows the same structure as described earlier, except that vertices unreachable from the source w are omitted from the network, as they represent locations at timesteps where no agent can be present and no obstacle move action is possible. This reduces the network size, thus improving runtime efficiency.

5.3 NAT-CBS and MAPF-DECOMP Solutions Compared

After fully describing the algorithmic details of NAT-CBS we present an example solution for an MAWR instance. Figure 3 shows the solutions plans for NAT-CBS and MAPF-DECOMP for the same instance. The optimal solution from NAT-CBS with a makespan of 8 (top) and the MAPF-DECOMP solution with a makespan of 11 (bottom). In the NAT-CBS solution, the blue obstacle undergoes a non-atomic task, moved by agent a_1 at timestep 1 and later by agent a_0 at timesteps 3-4. Additionally, in both solutions, the red obstacle, which starts at the same location as required in $\mathcal{L}_{\textbf{goal}}$, is temporarily moved to allow another obstacle to pass. It is then moved back to its location in $\mathcal{L}_{\textbf{goal}}$. In the NAT-CBS solution, this repositioning is done in a non-atomic manner, with agent a_1 moving it at timestep 1 and agent a_0 moving it at timestep 7.

5.4 Optimality of NAT-CBS

Lemma 1. *Given a realizable obstacles plan Π_{obs} with makespan T, there exists an agents plan Π_{agents} that realizes Π_{obs} with the same makespan T.*

Proof. By definition, if Π_{obs} is realizable then there exists an agents plan Π_{agents} that realizes Π_{obs}. Furthermore, the makespan of any realizing plan Π_{agents} is greater or equal than the makespan of Π_{obs}, since any obstacle movement is carried out by an agent. Finally, if a realizing agents plan Π_{agents} has a makespan greater than T, then truncating Π_{agents} to its first T timesteps still realizes Π_{obs}. □

Thus, the cost of the solution to an MAWR instance is exclusively determined by the cost of its obstacles plan.

Lemma 2. *Let $\mathcal{N}$ be a CT node with corresponding obstacles plan Π_{obs}, and consider the MCMF problem of node $\mathcal{N}$. There is a bijective mapping from each feasible flow to a valid agents plan Π_{agents} that is consistent with the set of positive edge constraints in $\mathcal{N}$.*

Proof Sketch. The bijection holds directly from the construction of the MCMF problem on the T-step time-expanded graph, where T is the makespan of $\mathcal{N}.\Pi_{\text{obs}}$. Given a feasible flow, it can be divided into n disjoint unit-flows where each represents a path from w to w'. These n paths form a valid agents plan due to the network construction: 1. Each path is a valid path on G by flow conservation. 2. For each agent a_i, there is a unique path that starts at its start location $s(a_i)$. 3. The paths are collision-free by capacity constraints. Moreover, the flow is constrained to pass through

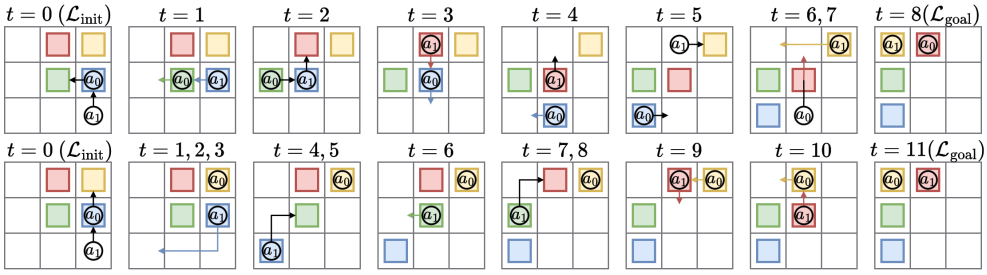


Figure 3: Two solutions of an MAWR instance: (top) NAT-CBS, makespan of 8; (bottom) MAPF-DECOMP, makespan of 11.

all edges with a positive constraint, by the lower bound constraints. Hence, Π_{agents} is consistent with the positive edge constraints.

Conversely, let Π_{agents} be a valid agents plan that is consistent with the set of positive edge constraints. Each agent path $\pi_{a_i} \in \Pi_{\text{agents}}$ induces a unit-flow that includes edges: 1. (w, v_0^{out}), where $v = s(a_i)$ is the start location of a_i. 2. (u_T^{out}, w'), where $u = \pi_{a_i}(T)$ is the location of a_i at timestep T. 3. The set of edges associated with the path on $\mathcal{G}^T$, such that if it finishes at a timestep earlier than T, continue with wait actions until timestep T. Each of the paths is a valid agent path and Π_{agents} is collision-free. Therefore, these n unit flows are edge disjoint, and combining them results in a flow of n units that satisfies flow conservation and capacity constraints. Lastly, Π_{agents} adheres to all positive edge constraints, therefore, all lower-bound constraints are satisfied by the flow and the flow is feasible. □

Definition 1. *A candidate goal node is a CT node that contains a valid obstacles plan.*

Definition 2. *A goal node is a CT node that contains a realizable and valid obstacles plan.*

Lemma 3. *Given a candidate goal node $\mathcal{N}$ with cost T, there exists a valid agents plan that realizes $\mathcal{N}.\Pi_{obs}$ if and only if there exists a feasible minimum-cost flow with total cost $(n \cdot T - R)$, where R is the total number of obstacle move actions in $\mathcal{N}.\Pi_{obs}$.*

Proof Sketch. Let $\mathcal{N}$ be a candidate goal node with cost T, and let R be the total number of obstacle move actions in $\mathcal{N}.\Pi_{\text{obs}}$. By Lemma 2 each valid agents plan with a makespan of T that obeys positive edge constraints corresponds to a feasible flow. Each agent action in an agents plan incurs a network cost of 0 for realizing an obstacle move, of 2 for reversing an obstacle move, and of 1 for all other actions. Thus, the cost of the feasible flow is $n \cdot T - S_1 + S_2$, where S_1 is the number of realized obstacle move actions and S_2 is the number of obstacle move actions that agents reverse in the plan. A lower-bound is given for the maximal S_1 and minimal S_2, that is, when all obstacle move actions are realized, then $S_1 = R$ and $S_2 = 0$. □

Theorem 1. *For any MAWR instance P for which a solution exists NAT-CBS returns a makespan-optimal solution for P.*

Proof Sketch. Following the optimality of CBS (Sharon et al. 2015; Li et al. 2019) and the fact that in phase **Ph1** NAT-CBS preserves CBS's high-level structure, NAT-CBS inherits the property of makespan-optimality w.r.t. the obstacles plans. Furthermore (and again, following the partial completeness of CBS), if a solution exists, NAT-CBS is promised to reach a goal node. By Lemma 3, upon reaching a candidate goal node, NAT-CBS finds a solution and returns it, if and only if it is realizable, i.e., it is a goal node. Since NAT-CBS is a best-first search algorithm, it guarantees that the first reached goal node is with an optimal solution cost, as determined by the cost of the obstacles plans. Due to Lemma 1, the cost of the solution to an MAWR instance is the cost of the obstacles plan, and thus the solution returned by NAT-CBS is makespan-optimal. □

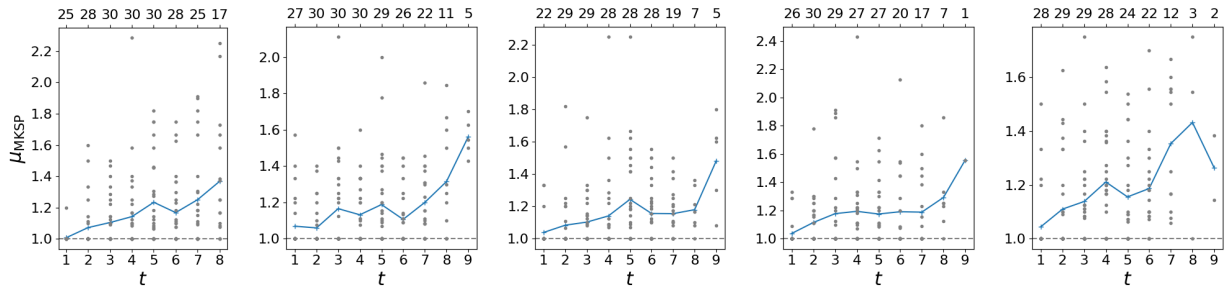
6 Experiments

We empirically compare NAT-CBS with MAPF-DECOMP. Both algorithms were implemented in C++[1] and executed on a 12th Gen Intel Core i7-12700 PC running Ubuntu, with 64 GB RAM. For NAT-CBS, we use Google OR-Tools' implementation for the MCMF algorithm[2]. For MAPF-DECOMP, we use the implementation provided by the authors of (Li and Ma 2023). MAPF-DECOMP computes obstacle trajectories before assigning agents to fulfill them and uses a bounded-suboptimal algorithm, EECBS (Li, Ruml, and Koenig 2021), for this task. To ensure a fair comparison with NAT-CBS, we fixed the suboptimality factor w to 1, ensuring the solution is as close to optimal as possible.

We consider different MAWR instances on grid maps. Each instance is characterized by three parameters: (i) the number of agents n, (ii) the number of movable obstacles m, and (iii) the number of tasks t, defined as the number of obstacles that change location between initial and goal layouts. For each combination of these parameters, we generate 30 random instances with agent and obstacle locations sampled uniformly. For a given number of agents n, the number of obstacles m varies from n to $2n$, while the number of tasks t ranges from 1 to $\min(m, n+1)$.

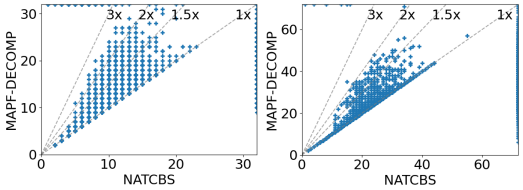
Fig. 5 compares the solution costs of the two algorithms, on two maps of sizes 8×8 and 15×20. Each point in the graph represents an instance, with the x and y-axes correspond to solution cost of NAT-CBS and MAPF-DECOMP, respectively. For each instance, both algorithms were given a time limit of five minutes. Notably, the average runtime of

[1] github.com/CRL-Technion/wh-rearrangement
[2] github.com/google/or-tools



(a) $m = 8$ (b) $m = 10$ (c) $m = 12$ (d) $m = 14$ (e) $m = 16$

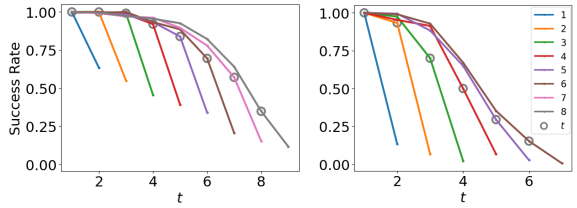
Figure 4: Solution cost ratio μ_{MKSP} as a function of the number of tasks t and the numbers of obstacles m on 8×8 maps with $n = 8$ agents. The number of solved instances (out of 30 generated) of each point is written on the top of each figure. The solution cost ratio of individual instance are marked in gray dots while their average is denoted by the blue line.



(a) 8×8 map. (b) 15×20 map.

Figure 5: Solution costs of MAPF-DECOMP and NAT-CBS. Points on the perimeter indicate the algorithm failed to solve an instance. For MAPF-DECOMP, this happens when no solution was found. For NAT-CBS, this happens due to either time limit reached or running out of memory.

NAT-CBS is approximately four orders of magnitude slower than that of MAPF-DECOMP. As expected, solution costs of NAT-CBS serve as a lower bound for those of MAPF-DECOMP as the latter does not guarantee to output optimal solutions. Interestingly, in some instances, solution cost of NAT-CBS can be as low as half of MAPF-DECOMP.



(a) 8×8 map. (b) 15×20 map.

Figure 6: Success rate of NAT-CBS as a function of t (number of tasks) for varying values of n (number of agents).

We now continue to compare how solution costs change between MAPF-DECOMP and NAT-CBS as we increase the number of tasks t and the number of obstacles m. We define the *solution cost ratio* μ_{MKSP} of an instance to be the ratio between the solution cost of MAPF-DECOMP and NAT-CBS and consider 8×8 maps with $n = 8$ agents. Results, depicted in Fig. 4, indicate that average values of μ_{MKSP} typically increase as the number of tasks t increases. Intuitively, this is because as there are more tasks, more obstacles need to be moved and the problem complexity increases. MAPF-DECOMP computes obstacle trajectories only once and then assigns them to agents without further refinement. This decoupled approach fails to capture settings for which the initial obstacle paths do not align with the optimal solution. In contrast, the systematic approach of NAT-CBS where obstacle paths are continuously adjusted through the realization conflict mechanism allows the algorithm to better coordinate obstacle movements with agent paths, leading to higher-quality solutions.

We continue to consider the success rate of NAT-CBS as a function of the number of tasks t (Fig. 6). As expected, the success rate decreases as t increases with the most substantial drop in success rate occurring when the number of tasks increases from $t = n$ (highlighted as a circle in the figure) to $t = n + 1$. To understand why, note that when an obstacles plan moves more obstacles than agents (a phenomenon that is exacerbated when $t > n$), the algorithm *must* introduce a realization conflict. To guarantee that obstacles plan in child CT nodes will will move only n obstacles may require introducing *multiple* such realization conflicts. This may require extremely long running times, which in turn, may cause the running time to surpass our time limit.

7 Summary and Future Work

In this paper, we studied the MAWR problem and introduced NAT-CBS, an algorithm that takes a novel obstacle-centric approach. We proved that NAT-CBS is optimal and demonstrated that it outperforms MAPF-DECOMP in terms of solution cost. Unfortunately, guaranteeing optimality comes with high runtime. Thus, a natural extension would be to develop a variant of NAT-CBS with bounded suboptimality guarantees, balancing solution quality and computational efficiency. A possible approach is to replace optimal obstacle path computation with bounded-suboptimal algorithms such as EECBS (Li, Ruml, and Koenig 2021). Additionally, obstacle path realization could leverage bounded-suboptimal network flow algorithms to efficiently assign agents while maintaining theoretical guarantees on solution quality.

Acknowledgments

This research was supported in part by the Technion Autonomous Systems Program.

References

Ahuja, R. K.; Magnanti, T. L.; and Orlin, J. B. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.

Atzmon, D.; Felner, A.; Li, J.; Shperberg, S. S.; Sturtevant, N. R.; and Koenig, S. 2023. Conflict-Tolerant and Conflict-Free Multi-Agent Meeting. *Artificial intelligence*, 322: 103950.

Bachor, P.; Bergdoll, R.-D.; and Nebel, B. 2023. The Multi-Agent Transportation Problem. *Association for the Advancement of Artificial Intelligence (AAAI)*, 37(10): 11525–11532.

Goldberg, A. V.; and Kharitonov, M. 1991. On Implementing Scaling Push-Relabel Algorithms for the Minimum-Cost Flow Problem. In *Network Flows And Matching, DIMACS Workshop*, volume 12 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 157–198.

Goldberg, A. V.; and Tarjan, R. E. 1987. Solving Minimum-Cost Flow Problems by Successive Approximation. In *ACM Symposium on Theory of Computing (STOC)*, 7–18.

Gómez, R. N.; Hernández, C.; and Baier, J. A. 2020. Solving Sum-of-Costs Multi-Agent Pathfinding with Answer-Set Programming. In *Association for the Advancement of Artificial Intelligence (AAAI)*, volume 34, 9867–9874.

Gordon, O.; Filmus, Y.; and Salzman, O. 2021. Revisiting the Complexity Analysis of Conflict-Based Search: New Computational Techniques and Improved Bounds. In *Symposium on Combinatorial Search (SoCS)*, 64–72.

Greshler, N.; Gordon, O.; Salzman, O.; and Shimkin, N. 2021. Cooperative Multi-Agent Path Finding: Beyond Path Planning and Collision Avoidance. In *International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 20–28.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2018. Conflict-Based Search with Optimal Task Assignment. In *Autonomous Agents and MultiAgent Systems (AAMAS)*, 757–765.

Kottinger, J.; Geft, T.; Almagor, S.; Salzman, O.; and Lahijanian, M. 2024. Introducing Delays in Multi Agent Path Finding. In *Symposium on Combinatorial Search (SoCS)*, 37–45.

Lam, E.; Le Bodic, P.; Harabor, D.; and Stuckey, P. J. 2022. Branch-and-Cut-and-Price for Multi-Agent Path Finding. *Computers & Operations Research*, 144: 105809.

Li, B.; and Ma, H. 2023. Double-Deck Multi-Agent Pickup and Delivery: Multi-Robot Rearrangement in Large-Scale Warehouses. *IEEE Robotics and Automation Letters*, 8(6): 3701–3708.

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. 2019. Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 279–283.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. In *Association for the Advancement of Artificial Intelligence (AAAI)*, volume 35, 12353–12362.

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and Path Planning for Multi-Agent Pickup and Delivery. In *Autonomous Agents and MultiAgent Systems (AAMAS)*, 1152–1160.

Ma, H.; Hönig, W.; Kumar, T. K. S.; Ayanian, N.; and Koenig, S. 2019. Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 7651–7658.

Ma, H.; and Koenig, S. 2016. Optimal Target Assignment and Path Finding for Teams of Agents. In *Autonomous Agents and MultiAgent Systems (AAMAS)*, 1144–1152.

Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Autonomous Agents and MultiAgent Systems (AAMAS)*, 837–845.

Madar, N.; Solovey, K.; and Salzman, O. 2022. Leveraging Experience in Lifelong Multi-Agent Pathfinding. In *Symposium on Combinatorial Search (SoCS)*, 118–126.

Salzman, O.; and Stern, R. 2020. Research Challenges and Opportunities in Multi-Agent Path Finding and Multi-Agent Pickup and Delivery Problems. In *Autonomous Agents and MultiAgent Systems (AAMAS)*, 1711–1715.

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

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

Tang, Y.; Ren, Z.; Li, J.; and Sycara, K. P. 2023. Solving Multi-Agent Target Assignment and Path Finding with a Single Constraint Tree. In *International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 8–14.

Vainshtein, D.; Sherma, Y.; Solovey, K.; and Salzman, O. 2023. Terraforming - Environment Manipulation during Disruptions for Multi-Agent Pickup and Delivery. In *Symposium on Combinatorial Search (SoCS)*, 92–100.

Wagner, G.; and Choset, H. 2015. Subdimensional Expansion for Multirobot Path Planning. *Artificial intelligence*, 219: 1–24.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1): 9–20.

Yu, J.; and LaValle, S. M. 2012. Multi-agent Path Planning and Network Flow. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, volume 86, 157–173.