Accelerating Search-Based Planning for Multi-Robot Manipulation by Leveraging Online-Generated Experiences

Yorai Shaoul*, Itamar Mishani*, Maxim Likhachev, Jiaoyang Li

Carnegie Mellon University
{yshaoul, imishani, maxim}@cs.cmu.edu, jiaoyangli@cmu.edu

Abstract

An exciting frontier in robotic manipulation is the use of multiple arms at once. However, planning concurrent motions is a challenging task using current methods. The high-dimensional composite state space renders many well-known motion planning algorithms intractable. Recently, Multi-Agent Path-Finding (MAPF) algorithms have shown promise in discrete 2D domains, providing rigorous guarantees. However, widely used conflict-based methods in MAPF assume an efficient single-agent motion planner. This poses challenges in adapting them to manipulation cases where this assumption does not hold, due to the high dimensionality of configuration spaces and the computational bottlenecks associated with collision checking. To this end, we propose an approach for accelerating conflict-based search algorithms by leveraging their repetitive and incremental nature – making them tractable for use in complex scenarios involving multi-arm coordination in obstacle-laden environments. We show that our method preserves completeness and bounded sub-optimality guarantees, and demonstrate its practical efficacy through a set of experiments with up to 10 robotic arms.

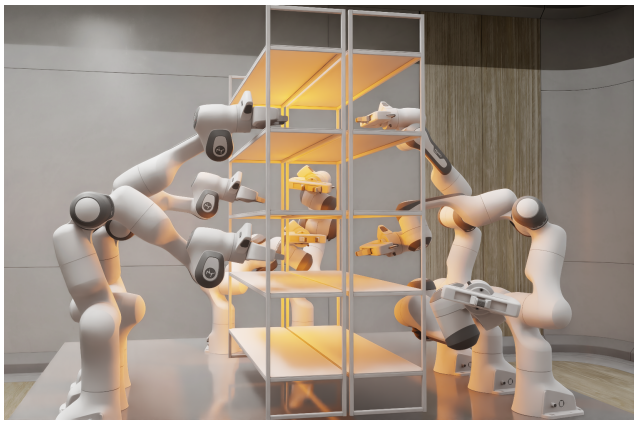


Figure 1: Eight robotic manipulators, each of 7-DoF, collaborating in a shelf-rearrangement pick and place task. Planning concurrent motions for all arms requires a motion planner capable of efficiently exploring a single arm's high-dimensional state space and reasoning about the motions of multiple robots operating in the shared task space.

Introduction

The synchronous use of multiple robotic arms may enable new application domains in robotics and enhance the efficiency of tasks traditionally carried out by a single arm. For example, in pick-and-place tasks (e.g., Fig. 1), multiple arms can potentially be more efficient than a single one, and in a manufacturing setting, multiple arms can be used to assemble a product collaboratively, unlocking the capability to perform tasks that are beyond the scope of a single arm. However, the inherent complexity of single-agent motion planning for robot manipulation (Canny 1988) makes it challenging to plan for multiple arms while ensuring collision-free paths, and thus has left Multi-Robot-Arm Motion Planning (M-RAMP) a relatively under-explored frontier in robotics.

To enable the use of multiple arms in more complex scenarios, we propose a method for accelerating M-RAMP. Our approach capitalizes on a key observation: widely-used Multi-Agent Path Finding (MAPF) algorithms exhibit a significant degree of repetitive planning. We exploit this repetitiveness by developing a method that leverages experiences gathered during the planning process. Unlike previous approaches that utilize incremental search techniques (Boyarski et al. 2021), we allow the use of bounded sub-optimal search techniques, which are crucial for exploring high-dimensional state spaces. To this end, we accelerate the single-agent planning process by reusing online-generated path experiences to speed up multi-agent search, ensuring both completeness and solution quality guarantees.

Our contributions in this paper are threefold. First, we introduce a novel method for M-RAMP that leverages the Conflict-Based Search framework and effectively re-uses intermediate search efforts to accelerate the search. Second, we provide a comprehensive theoretical analysis of our proposed framework, demonstrating its bounded sub-optimality guarantees. Third, we offer an empirical evaluation of our method and other algorithms in various multi-arm manipulation scenarios that include deadlock avoidance, cluttered environments, and closely interacting goals.

*These authors contributed equally.

Related Work

The literature has extensively examined path planning for both single and multiple agents. In the context of single-agent search, decades of research have yielded algorithms capable of scaling successfully to high-dimensional and computationally expensive search spaces. However, efforts in MAPF have generally been applied to domains such as 2D-grid worlds, resulting in algorithms that often rely on assumptions such as fast single-agent planning and informative heuristics. These assumptions may not always hold in other scenarios such as robotic manipulation. In this section, we discuss relevant work in MAPF, describe common approaches to M-RAMP in practice, and review the use of experiences in planning that inspired us to connect MAPF algorithms with multi-arm manipulation more effectively.

Multi-Agent Path Finding

MAPF is the problem of finding collision-free paths for a set of agents on a graph (e.g., on a grid world) (Stern et al. 2019). MAPF has been studied extensively, and optimal (e.g., CBS (Sharon et al. 2015)), bounded sub-optimal (e.g., ECBS (Barer et al. 2014)), and sub-optimal but effective (e.g., MAPF-LNS2 (Li et al. 2022)) algorithms have been proposed. Some work has also been done to generalize MAPF algorithms to non-point robots (Li et al. 2019), however, the most common domain is still in 2D. Arguably, the most influential family of algorithms is Conflict-Based Search (CBS) and its extensions (Sharon et al. 2015; Barer et al. 2014; Li, Ruml, and Koenig 2021). CBS is a two-level search algorithm, where at the low level, each agent is assigned a single-agent path-planning problem. At the high level, collisions between single-agent solutions are resolved by imposing constraints on the low-level planners.

CBS is known to provide completeness and optimality guarantees. However, CBS is also known to be computationally expensive as it requires repeated low-level searches upon additions of constraints. Given this inefficiency, CBS is often regarded as impractical for domains, such as manipulation, where planning for a single agent requires exploring a high-dimensional space and does not enjoy informed heuristics. In this work, we capitalize on this repetition and propose a method for accelerating CBS-based algorithms by reusing online-generated previous search solutions.

Planning for Multi-Arm Manipulation

In practice, planning for multi-arm manipulation is often done with coupled or prioritization methods. In coupled methods, the state of all arms is seen as a single composite state, and the search is performed in this space with algorithms such as A* (Nilsson 1982), Rapidly-exploring Random Trees (RRT) (Karaman and Frazzoli 2011), and their variants (e.g., weighted A*, RRT* (Karaman and Frazzoli 2011), and RRT-Connect (Kuffner and LaValle 2000)). With the addition of more arms, the search space grows exponentially, and in general, coupled methods may not scale to large numbers of arms.

In scenarios where coupled planning is rendered intractable due to the exponential growth of the search space, prioritization methods may be effective in reducing its dimensionality. In Prioritized Planning (PP) (Erdmann and Lozano-Perez 1987), each arm is assigned a priority, and the lower-priority arms treat higher-priority arms as moving obstacles. In the general case, solving the prioritized planning problem is more efficient than solving the coupled case, as the search space is reduced to the space of each single arm. However, the price paid for this dimensionality reduction is the loss of completeness. In scenarios requiring close coordination between arms, completeness may be important.

Recently, planning algorithms have been proposed specifically for teams of high-dimensional agents and applied to multi-arm settings. These methods explore the search space via pre-constructed single-agent roadmaps (e.g., probabilistic roadmaps (PRM) (Kavraki et al. 1996)), which may need to be arbitrarily resampled (Solis et al. 2021) to find collision-free paths in complex environments. Shome et al. (2020) present dRRT*, a method for exploring the composite state space of agents by traversing individual agents' roadmaps towards sampled configurations with goal bias. Solis et al. (2021) present CBS-MP, a variant of CBS that imposes new constraints on the search to resolve collisions. Specifically, to resolve a collision between two agents, CBS-MP requires one agent to avoid the colliding configuration of the other at the time of the collision.

Leveraging Experience in Planning

Streamlining motion planning from experience encompasses a wide range of motion-planning algorithms. These generally benefit from either utilizing offline-generated data (i.e., precomputation), leveraging online-generated data, or both.

Precomputation as Experience The utilization of offline computations to enhance online search efficiency is well exemplified by the PRM algorithm and its variants. Another novel approach is found in the Constant-Time Motion Planners (CTMP) family of algorithms, which operates on precomputed data structures to achieve constant-time path generation in online scenarios (Islam, Salzman, and Likhachev 2019; Islam et al. 2021; Mishani, Feddock, and Likhachev 2023). In recent research, a significant focus has been on the offline decomposition of the configuration space into collision-free convex sets (Dai et al. 2023). This decomposition enables planning smooth trajectories within these sets using optimization methods (Marcucci et al. 2022). Furthermore, various algorithms based on precomputed trajectories (Phillips et al. 2012), have been employed to expedite the search process. When extending these techniques to plan for multi-arm setups, it becomes essential to decompose the composite configuration space for computing collision-free trajectories. However, challenges arise when the environment or the robot undergoes changes, which can be as simple as rotating a bin or altering the robot's geometry by grasping an item. These changes may require resource-intensive operations like redoing precomputation or propagating changes, emphasizing a drawback inherent to using offline-generated experiences.

Online-Generated Experiences Online-generated experiences have been used in MAPF, but more commonly

in anytime incremental heuristic search and reinforcement learning (RL). In the heuristic search, experiences are used to directly guide the search, and in RL, as replay buffers to improve learning stability and sample efficiency (Fedus et al. 2020). In MAPF, one algorithm leveraging repetition in planning is Iterative-Deepening CBS (IDCBS) (Boyarski et al. 2021) that employs Lifelong Planning A* (LPA*) (Koenig, Likhachev, and Furcy 2004) for single-agent planning. However, this approach faces challenges in manipulation cases where bounded sub-optimal search is employed to navigate the high-dimensional search space (Likhachev and Koenig 2005).

Anytime algorithms, like Anytime Repairing A* (ARA*) (Likhachev et al. 2008), can be seen as methods that use experiences generated online to improve the quality of the solution over time. ARA* performs a sequence of searches that, given enough time, converge to the optimal solution. A recent anytime approach inspired by Likhachev et al. (2008); Phillips et al. (2012) and presented in Mishani, Feddock, and Likhachev (2023) employs both precomputation and online experience. Their algorithm computes an initial, potentially sub-optimal path within a (short) constant time and improves the quality of the path using the current best solution as an experience.

Drawing inspiration from the way Mishani, Feddock, and Likhachev (2023) capitalize on the flexibility seen in online-generated experiences, and with the observation that CBS-based algorithms inherently exhibit repetition in the form of nearly identical single-agent planning queries, we propose a method for accelerating MAPF algorithms by reusing online-generated experiences.

Preliminary

In this paper, we propose a method for solving the M-RAMP problem by extending the CBS algorithm and its variants to reuse search efforts. We first describe the problem formulation, explain how to cast it as a graph-search problem, and then detail how the CBS algorithm can solve M-RAMP.

M-RAMP: Problem Formulation

Consider $\mathcal{Q}^i \subseteq \mathbb{R}^d$ as the configuration space of a single robotic arm $\mathcal{R}_i$ with d degrees of freedom (DoF). A configuration $q^i \in \mathcal{Q}^i$ is defined by assigning values to all the DoF, i.e., defining the joint angles.

Let the composite configuration space of n robotic arms be $\mathcal{Q} = \mathcal{Q}^1 \times \mathcal{Q}^2 \times \cdots \mathcal{Q}^n$. With all manipulators operating within the same environment $\mathcal{W} \subset \mathbb{R}^3$, let $\mathcal{Q}_{\text{free}}$ be the set of all collision-free composite configurations (both with the environment and between robots):

$$\mathcal{Q}_{\text{free}} = \{q \in \mathcal{Q} \mid q \text{ is collision-free}\}$$

Given an initial composite configuration $q_{\text{start}} \in \mathcal{Q}_{\text{free}}$ and a composite goal configuration $q_{\text{goal}} \in \mathcal{Q}_{\text{free}}$, we want to find a shortest valid path $\bar{\Pi} : [0, T] \rightarrow \mathcal{Q}_{\text{free}}$, in terms of per-robot time in motion, where $\bar{\Pi}(0) = q_{\text{start}}$ and $\bar{\Pi}(T) = q_{\text{goal}}$.

M-RAMP as Graph Search

A discrete analog of the M-RAMP problem is to find a sequence of composite configurations $\{q_0, q_1, \cdots, q_T\}$ such that $\forall t \in \{0, \cdots T\},\ q_t \in \mathcal{Q}_{\text{free}}$, each interpolated configuration between q_t and q_{t+1} is collision-free, and $q_0 = q_{\text{start}}$ and $q_T = q_{\text{goal}}$. Instead of addressing the motion planning problem in the high-dimensional composite state space, it is possible to decompose the problem into a set of single-agent motion planning problems each computing a collision-free path π^i for $\mathcal{R}_i$ between q^i_{start} and q^i_{goal}.

One way to realize single-agent planning is by creating a lattice of allowable motions (often a small rotation in a single joint) and encoding those in an implicit graph for each robot $\mathcal{R}_i$ to represent its configuration space. Vertices correspond to configurations q^i and edges to unit-time transitions between configurations (Cohen et al. 2011). Between time steps, agents can traverse an edge or wait at a vertex.

When dealing with robot arms that have complex search spaces it is generally infeasible to enumerate all collision-free vertices and edges before the search or to compute informative heuristics. Each validity check requires querying a *collision-checker*, which determines the arms' occupancy in $\mathcal{W}$ via forwards kinematics and checks for collisions. This operation is usually computationally expensive. Therefore, it is common to determine edge and vertex validity during the search itself and,S to further reduce the number of collision checks, employ weighted A* for the single-agent search.

This approach allows us to find a path $\pi^i = \{q^i_0, q^i_1, \cdots, q^i_{T_i}\}$ for each robot that avoids collisions with obstacles, but potentially not with each other. Arms may collide during edge traversals (named *edge-conflict*), and similarly when at vertex configurations (named *vertex-conflicts*). To check for validity, the individual paths can be combined into a multi-agent solution $\Pi = \{\pi^1, \pi^2, \cdots, \pi^n\}$ or as a composite path by merging the individual robot configurations at each time step, i.e., $q_t = \{q^1_t, \cdots, q^n_t\}$. One way to find a valid (collision-free) solution of least sum of costs $|\Pi| = \sum_i |\pi^i| = \sum_i \sum_{t'=0}^{T_i - 1} \text{cost}(q^i_{t'}, q^i_{t'+1})$ is to use CBS.

Conflict-Based Search

CBS is a two-level best-first search algorithm designed to solve the MAPF problem. Although commonly applied to 2D grids, it can handle any discrete time problem where agent motion is on a graph. CBS identifies and resolves conflicts between agents by imposing motion constraints and replanning paths accordingly.

CBS begins by querying a path π_i for each agent $\mathcal{R}_i$ between its start and goal configurations without regard to other agents. This solution $\Pi = \{\pi^0, \cdots, \pi^n\}$ is a (possibly invalid) candidate solution for the problem, and it is stored in the OPEN list of the high-level search. Each high-level node N, known as a Constraint-Tree (CT) node, contains constraints $N.C$ on low-level planners and paths $N.\Pi$ that adhere to those constraints for all agents. The cost of a CT node N.cost is the sum of the costs of its stored paths $|N.\Pi|$.

CBS proceeds iteratively, selecting least-cost CT nodes N from OPEN and evaluating them for conflicts. If no conflicts are found in a solution, it is accepted as valid, and the algorithm terminates. Otherwise, CBS chooses a conflict and resolves it by splitting N into two child nodes that are added to OPEN. In each child node, one agent participating

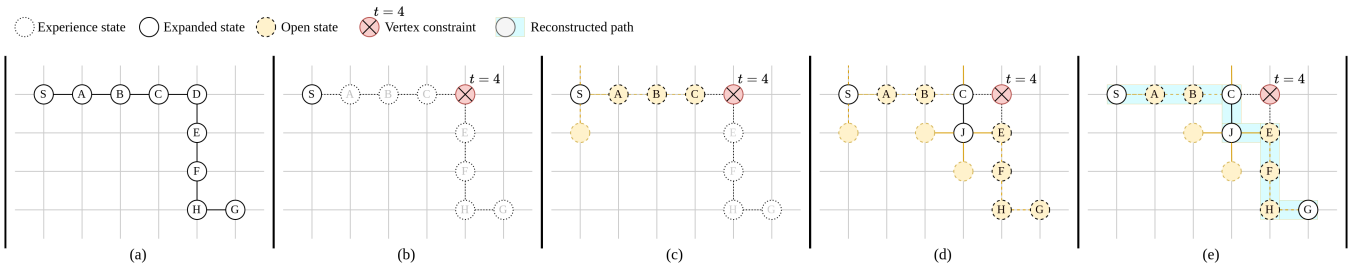


Figure 2: An illustration of our proposed algorithm accelerating a single agent search on a four-connected grid via reusing previous search efforts. (a) A single-agent path from S to G computed in a previous iteration. (b) Upon imposing a new constraint on the agent, shown in red, replanning is required. The previous path is drawn in light gray. (c) Upon expansion of node S, a prefix $\{A, B, C\}$ of the experience path is added to OPEN along with all other successors of S. (d) shows two steps: node C is selected for expansion from OPEN, and in the next iteration node J is expanded from OPEN. Upon expanding J, a segment of the experience is added to OPEN, since one of J's successors is equivalent to a node in the experience. (e) Finally, G is expanded from OPEN and the search terminates and recovers a path. In this example, the work done by xWA* (Alg. 2) is smaller than that of its previous iteration. By reusing experience, the intermediate nodes expanded are C and J.

in the conflict is prohibited from using the edge or vertex it used during the conflict. This is imposed via *edge-* or *vertex-constraints*. In M-RAMP, where each agent operates on their own graph, edge-conflicts take the form $\langle i, q_t^i, q_{t+1}^i, t\rangle$ or $\langle j, q_t^j, q_{t+1}^j, t\rangle$, forbidding $\mathcal{R}_i$ or $\mathcal{R}_j$ from moving on the edge between the configurations q_t^i and q_{t+1}^i or q_t^j and q_{t+1}^j at time t. *Vertex-constraints* $\langle i, q_t^i, t\rangle$, $\langle j, q_t^j, t\rangle$ forbid agents from visiting a configuration at time t.[1]

Bounded CBS (BCBS)

BCBS (Barer et al. 2014) is a bounded sub-optimal variant of CBS that trades off optimality in favor of efficiency. It employs *focal search*, an algorithm based on A_ϵ^* (Pearl and Kim 1982), in its low- and high-level searches. Focal search employs two priority queues: OPEN and FOCAL. OPEN mirrors the A* queue and is sorted by an admissible priority function f_1. FOCAL comprises a subset of OPEN defined as FOCAL $= \{N \in \text{OPEN} \mid f_1(N) \leq w \cdot \min_{N' \in \text{OPEN}} f_1(N')\}$ with $w \geq 1$ a sub-optimality bound. The next node expanded is chosen from FOCAL according to priority function f_2. Denoting search levels with a superscript, the BCBS high-level sets f_1^H to the sum of costs of CT nodes N and f_2^H to the number of conflicts in N. At the low-level, f_1^L is the original priority function of the search, and f_2^L prioritizes nodes whose current path on the search tree has fewer conflicts against the preexisting paths of other agents. BCBS guarantees $w^L w^H$-sub-optimal results.

[1] Assuming complete low-level searches, CBS-based algorithms are complete as long as, when they create constraints c_1 and c_2 for resolving a conflict, there won't exist any valid solution that invalidates both c_1 and c_2. Otherwise, valid solutions with respect to conflicts will be marked as invalid against constraints. Interestingly, by viewing the completeness of CBS in this way, it can be shown that some CBS variants, such as CBS-MP, gain efficiency by imposing stronger constraints but sacrifice completeness. We have discussed CBS-MP's theoretical guarantees with the authors and reached this conclusion.

Enhanced CBS (ECBS)

ECBS (Barer et al. 2014) is an improvement on BCBS, producing solutions bounded by a single factor w. To do so, the low-level in ECBS, when planning for $\mathcal{R}_i$, returns a path π^i and a lower bound on the optimal cost $C^{*,i}$ of this path $lb(i) = \min_{s \in \text{OPEN}} f(s) \leq C^{*,i}$. ECBS keeps track of a lower bound $LB(N)$ on the sum of costs for each CT node N defined as $LB(N) = \sum_{i=1}^{n} lb(i)$. Defining $LB = \min_{N' \in \text{OPEN}} LB(N') \leq C^*$ being a lower bound on the optimal sum of costs, and constructing the high-level FOCAL with FOCAL $= \{N \in \text{OPEN} \mid |N.\Pi| \leq w \cdot LB\}$, CT nodes in the high-level FOCAL satisfy $|N.\Pi| \leq w \cdot LB \leq w \cdot C^*$.

In CBS and its variants, consecutive low-level searches are nearly identical. For example, a low-level planner for agent $\mathcal{R}_i$ invoked with constraints $C_i = \{c \in C \mid c \text{ involves } \mathcal{R}_i\}$ may next be invoked with $C_i \cup \{\langle i, q_t^i, t\rangle\}$ after a single new constraint is added. This minor difference suggests potential benefits from reusing parts of the previous solution.

Algorithmic Approach

Our main contribution in this work is an experience-acceleration framework for CBS-based algorithms. We instantiate this framework in two incarnations, *xCBS* and *xECBS*, accelerating CBS and ECBS, respectively. In this section, we present the general form of our acceleration method in an intuitive manner grounded by Algorithm 1 and Algorithm 2 and then provide a theoretical analysis of its performance alongside its instantiations xCBS and xECBS.

Experience-Acceleration Framework

Our framework follows the CBS structure and informs new low-level planner calls with the experience generated in previous search efforts. In the high-level search (Alg. 1), each CT node contains a set of paths Π, one for each agent, and a set of constraints C. Upon obtaining a new node from a priority queue, it is checked for conflicts (line 14). If none exist, the node is a goal node, and the paths are returned (line

Algorithm 1: High-Level (HL) Planner

Input : $q_{\text{start}} = \{q^1_{\text{start}}, \ldots, q^n_{\text{start}}\}$
$q_{\text{goal}} = \{q^1_{\text{goal}}, \ldots, q^n_{\text{goal}}\}$
f_1^H, f_2^H: priority functions,
w^H: sub-optimality bound.
Output: $\Pi = \{\pi^1, \cdots, \pi^n\}$

1 **Procedure** `InitRootNode()`
2 $N_{\text{root}}.C \leftarrow \emptyset$
3 $N_{\text{root}}.\Pi \leftarrow$ invoke `LLPlanner` for each agent
4 $N_{\text{root}}.\text{cost} \leftarrow |N_{\text{root}}.\Pi|$
5 **return** N_{root}
6 **Procedure** `Plan` (q_{start}, q_{goal})
7 $N_{\text{root}} \leftarrow$ `InitRootNode()`
8 OPEN $\leftarrow \{N_{\text{root}}\}$
9 **while** *OPEN not empty* **do**
10 $B \leftarrow \min_{N' \in \text{OPEN}} f_1^H(N')$
11 FOCAL $\leftarrow \{N \in \text{OPEN} \mid N.\text{cost} \leq w^H \cdot B\}$
12 $N \leftarrow \operatorname{arg\,min}_{N' \in \text{FOCAL}} f_2^H(N')$
13 OPEN.remove(N)
14 **if** $N.conflicts = \emptyset$ **then**
15 **return** $N.\Pi$
16 constraints $\leftarrow$ `GetConstraints` (N.conflicts.first)
17 **for** $c \in$ *constraints* **do**
18 New CT node $N' \leftarrow$ `Copy` (N)
19 $N'.C \leftarrow N.C \cup$ c
20 $i \leftarrow c$.agent_id
21 *Experience* $\leftarrow N.\Pi[i]$
22 $N'.\Pi[i] \leftarrow$ `LLPlanner.Solve` (q^i_{start}, q^i_{goal}, $N'.C$, *Experience*) *// Invoke* `LLPlanner` *for each agent involved*
23 $N'.\text{cost} \leftarrow |N'.\Pi|$
24 $N'.\text{conflicts} \leftarrow$ `FindConflicts` ($N'.\Pi$)
25 OPEN.insert(N')
26 **return** $\emptyset$

15). Otherwise, a pair of constraints is derived from the first conflict (line 16). Usually, CBS proceeds by creating a new CT node, one with an added constraint from the constraint set (lines 17-19), and replans a single-agent path for the affected agent from scratch (line 22). However, we recognize that a considerable portion of the previously generated path remains valid and can be effectively reused. Thus, our high-level search caches a copy of the previously computed path as *experience* and passes it to the low-level motion planner (lines 21, 22). The experience path is a sequence of configurations (including waits and cycles). It is possible to construct the experience set for a replanned agent $\mathcal{R}_i$ in multiple ways. We have experimented with reusing its previous path stored in its parent CT node, paths for $\mathcal{R}_i$ from all previous searches on the CT branch, and all paths for $\mathcal{R}_i$ across the CT. Reusing the previous path performed the best.

The low-level of our acceleration framework, namely *xWA**, is detailed in Algorithm 2 and illustrated in Fig. 2. Let a *state* be a configuration with time. Each state expansion (lines 23-30) adds a set of successors to the OPEN list. Upon a choice of a state for expansion (line 23), the search terminates if it is a goal state (lines 25-26). Otherwise, we check if

Algorithm 2: xWA*: Low-Level (LL) Planner

Input : q_{start}: start ($q_{start} \in \mathcal{Q}^{free}$),
q_{goal}: goal ($q_{goal} \in \mathcal{Q}^{free}$),
C: constraints set,
$\tilde{\pi}$: experience sequence (without time),
w_1^L, w_2^L: sub-optimality bound in WA*, focal list,
$f_1^L := g(s) + w_1^L h(s)$, f_2^L: priority functions.
Output: Path π

1 **Procedure** `TryInsertOrUpdate` (s_1, s_2, *OPEN*)
2 **if** s_2 *was not visited before* **then**
3 $g(s_2) \leftarrow \infty$
4 **if** $g(s_2) > g(s_1) + cost(s_1, s_2)$ **then**
5 $g(s_2) \leftarrow g(s_1) + \text{cost}(s_1, s_2)$
6 OPEN.InsertOrUpdate(s_2)
7 **Procedure** `PushPartialExperience` ($\tilde{\pi}$,s,C,*OPEN*)
8 $(q_0, t_0) \leftarrow s$
9 $\hat{\pi} \leftarrow \tilde{\pi}$.`suffix`$(q_0)$; *// The experience configurations after* q_0.
10 **for** $\hat{q} \in \hat{\pi}$ **do**
11 $\hat{s} \leftarrow (\hat{q}, t_0 + 1)$
12 **if** `IsEdgeValid` $((q_0, t_0), \hat{s}, C) \wedge$ `IsStateValid` $(\hat{s}, C)$ **then**
13 `TryInsertOrUpdate` $((q_0, t_0), \hat{s}$, OPEN)
14 $(q_0, t_0) \leftarrow \hat{s}$
15 **else**
16 break
17 **Procedure** `Solve` (q_{start}, q_{goal}, C, $\tilde{\pi}$)
18 $s_{\text{root}} \leftarrow (q_{start}, 0)$ *// Adding time to state*
19 OPEN $\leftarrow \{s_{\text{root}}\}$
20 `PushPartialExperience` ($\tilde{\pi}$, s_{root}, C, *OPEN*)
21 **while** *OPEN* $\neq \emptyset$ **do**
22 FOCAL $\leftarrow \{s \in OPEN \mid f_1^L(s) \leq w_2^L \min_{s' \in \text{OPEN}} f_1^L(s')\}$
23 $s = (q, t) \leftarrow \operatorname{arg\,min}_{s' \in \text{FOCAL}} f_2^L(s')$
24 OPEN.remove(s)
25 **if** $q = q_{goal} \wedge$ *no future constraints at* q_{goal} **then**
26 **return** $\pi \leftarrow$ `ExtractPath`(s)
27 **if** $q \in \tilde{\pi}$ **then**
28 `PushPartialExperience` ($\tilde{\pi}$, s, C, *OPEN*)
29 **for** $s' \in$ `GetSuccessors`(s, C) **do**
30 `TryInsertOrUpdate` (s, s', *OPEN*)
31 **return** $\emptyset$

the expanded state belongs to the experience path (line 27). If the expanded state belongs to an experience, starting from that state, we aim to add as much of the experience as possible to the OPEN list (line 28). This process is also applied to the start state (line 20) and essentially provides a "warm start" to the search effort. Given an expanded state s that belongs to an experience, we attempt to add consecutive states from the experience while propagating their associated time (line 11) and setting or updating their associated cost (line 13). We continue this process until a *termination condition* is met (line 12). A simple condition terminates addition when it violates constraints in C. A more complex condition, for example, terminates the addition of states when transitions lead to a collision with another agent's path. This condition is effective for xECBS.

The effect of adding an experience path to the OPEN list of a bounded sub-optimal search algorithm, such as weighted A*, could be a rapid exploration of states that are closer to the end of the experience path (and, consequently, closer to the goal). Fig. 2 illustrates this effect. Such exploration results in the algorithm "jumping" over previously explored regions and avoiding redundant search efforts, directing its focus closer to the end of the experience.

Collision checking against the static environment, a significant factor in the slowness of planning for manipulation, can also be directly accelerated with experience. To this end, our acceleration framework also keeps track of the configurations (q_t^i, q_{t+1}^i) in all valid transitions (s_t, s_{t+1}) for each robot $\mathcal{R}_i$. With this information, the successors set (line 29) can be computed more rapidly by only checking the validity of edges previously unseen. Because one single-agent search can revisit the same configuration at different times, such experience reuse also speeds up the first search.

Theoretical Analysis

In this section, we discuss the theoretical foundation of our algorithm. We formally define the problem for both levels of CBS as focal search and introduce some of the properties of CBS and its bounded sub-optimal variants. We show that accelerating CBS variants by reusing experience retains completeness and bounded sub-optimality guarantees.

We commence by establishing the bounded sub-optimality of the low-level planner xWA* that leverages past experiences. To allow for the use of inflated heuristics using a weighted OPEN list (Veerapaneni, Kusnur, and Likhachev 2023), which is common in manipulation, we expand our analysis to low-level planners with w_1-*admissible* (Pearl and Kim 1982) priority function $f_1(s) = g(s) + w_1 h(s)$.

Lemma 1. *A focal search employing a w_1-admissible function $f_1(s)$ ($w_1 \geq 1$) and FOCAL $= \{s \in$ OPEN $\mid f_1(s) \leq w_2 \min_{s' \in OPEN} f_1(s')\}$ has a sub-optimality factor $w_1 \cdot w_2$.*

Proof. Let s_0 be a node on an optimal path that resides in OPEN. For every expanded node s':

$$f_1(s') \leq w_2 \min_{s \in \text{OPEN}} f_1(s) \leq w_2 f_1(s_0) =$$

$$w_2(g(s_0) + w_1 h(s_0)) \leq w_2 w_1 (g(s_0) + h(s_0)) \leq w_2 w_1 C^* \qquad \square$$

Next, we show that incorporating experiences in xWA* neither impacts its sub-optimality nor sacrifices completeness.

Lemma 2. *Consider a best-first search storing frontier states in an OPEN list. When systematically incorporating successors into OPEN, if additional nodes are introduced along with their associated priority function values, completeness and bounded sub-optimality persist.*

Proof. When introducing new nodes to OPEN, the original OPEN of weighted A* becomes a subset of the modified OPEN. Thus, the algorithm maintains its systematic nature, ensuring completeness. Furthermore, we also know that FOCAL will only be populated by nodes from OPEN that are within the specified sub-optimality bound. Consequently, when a goal state is expanded, the solution remains bounded sub-optimal. $\square$

Theorem 1. *xWA* is complete and bounded sub-optimal by a factor of $w^L = w_1^L w_2^L$.*

Proof. Since xWA* is a w_2^L-sub-optimal focal search, which employs a weighted OPEN (w_1^L-admissible $f_1 = f_1^L$), the proof follows directly from Lemmas 1 and 2. $\square$

We continue with analyzing the sub-optimality of the high-level planner, which is defined as a focal search. Let $f_1 = f_1^H$ be a priority function such that for every CT node N, $f_1^H(N) \leq N$.cost. Additionally, let the FOCAL queue be defined as $FOCAL = \{N \in \text{OPEN} \mid N.\text{cost} \leq w^H \cdot \min_{N' \in \text{OPEN}} f_1^H(N')\}$:

Lemma 3. *Let w^H, w^L be the sub-optimality factor of the high- and low-level focal searches, respectively. For any $w^H, w^L \geq 1$, the cost of a solution is at most $w^H w^L C^*$.*

Proof. Let N be a node in FOCAL of the high-level search. For each of the $\mathcal{R}_i$ of n agents, we denote the returned cost of a low-level plan as $cost(i)$ and its optimal cost as $C^{*,i}$.

$$N.\text{cost} \leq w^H \min_{N' \in \text{OPEN}} f_1^H(N') \leq w^H \min_{N' \in \text{OPEN}} \sum_{i=1}^{n} cost(i)$$

$$\leq w^H \sum_{i=1}^{n} w^L C^{*,i} = w^H w^L C^* \qquad \square$$

Theorem 2. *Our proposed acceleration framework is complete and bounded sub-optimal.*

Proof. Building on the work of Barer et al. (2014), we can establish that the high-level search is complete if the low-level planner is complete and the constraints are valid. Furthermore, it is bounded sub-optimal with a factor of w^H. As shown in Theorem 1, xWA* is both complete and bounded sub-optimal by w^L. By transitivity, our overall algorithm inherits completeness from both the high and low levels. Additionally, leveraging Lemma 3, we conclude that the sub-optimality upper bound of our approach is $w^H w^L C^*$. $\square$

Finally, we detail xCBS and xECBS as instances of our experience-acceleration framework and show their completeness and bounded sub-optimality.

xCBS At the low- and high-level, xCBS does not use focal lists (i.e., $f_2^L = f_1^L, f_2^H = f_1^H,\ w^H = w_2^L = 1$). Its CT node prioritization is identical to that of CBS ($f_1^H(N) = N$.cost), and so is its constraint generation function. Hence, its low- and high-levels are complete. By Theorem 1, it has a sub-optimality factor of $w^L = w_1^L$. Thus, xCBS maintains completeness and is bounded sub-optimal by factor w_1^L.

xECBS The low- and high-level focal lists are ordered similarly to ECBS, prioritizing nodes with fewer conflicts. At the high-level, xECBS uses $f_1^H(N) = LB(N)$, keeping the sub-optimality factor $w^H = 1$. At the low level, xWA* contributes a sub-optimality factor of $w^L = w_1^L \cdot w_2^L$. Thus,

its total sub-optimality factor is $w_1^L \cdot w_2^L$. Completeness is guaranteed for the same reasons as xCBS.

Experiments

To evaluate xECBS and xCBS, we created collaborative manipulation tasks with varying numbers of robots, obstacle density, and robot-robot interaction complexity, and evaluated them in simulation. We used MoveIt! (Coleman et al. 2014) for environment handling and Isaac Sim for rendering. This setup can directly control real robots. Each robot in our experiments is a Franka Panda manipulator with 7-DoF. The experiments were conducted on an Intel Core i9-12900H with 32GB RAM (5.2GHz).

Experiments Setup

Our experiments focus on testing the scalability of algorithms as well as their applicability for real-world use. We set up 7 scenes, each with 50 planning problems defined by starts $q_{\text{start}} \in \mathcal{Q}_{\text{free}}$ and goals $q_{\text{goal}} \in \mathcal{Q}_{\text{free}}$.

To test the applicability of algorithms for real-world scenarios, we evaluated algorithms in two sample tasks: shelf rearrangement with 8 arms and bin-picking with 4 arms (Fig. 3). For each scene, we randomly sampled 50 start and goal states from a set of task-specific configurations (e.g., pick and place configurations at different bins or positions in between shelves). Given that the robots operate within the same task space, these configurations require motion plans with substantial proximity between arms.

To shed light on how algorithms scale with the number of arms, we tested their performance in free or lightly cluttered scenes with 2, 4, 6, 8, and 10 arms as shown in Fig. 4. The starts and goals for each agent are in the shared workspace region. In each setup, robots were organized in a circle, and in the cases with 6, 8, and 10 robots, a thin obstacle was placed in the circle to encourage interaction.

Baselines

To show the efficacy of our method, we compare it to ubiquitous algorithms commonly used to solve the M-RAMP problem and other algorithms recently applied to M-RAMP.

Sampling-Based Methods We include PRM and RRT-Connect, which are arguably the most commonly used algorithms for planning in manipulation. For both, the search space is the composite state space $\mathcal{Q}$. We use their OMPL (Sucan, Moll, and Kavraki 2012) implementation. We include dRRT* (Shome et al. 2020), a more recent algorithm applied to M-RAMP that explores the composite state space via transitions on single-agent roadmaps. In our implementation, the single-agent roadmaps contain a minimum of 1,500 nodes, with increments of 1,000 added if the roadmap cannot be connected to the start or goal configurations.

Search-Based Methods We include PP, CBS, ECBS, and CBS-MP in our comparison, as well as our proposed methods xCBS and xECBS. For all, the single agent planners are weighted A* with a uniform cost for transition and an L_2 joint-angle distance as a heuristic. The heuristic inflation value [2] is $w_1^L = 50$, and in ECBS and xECBS, the sub-optimality bound is set to $w^H = w_2^L = 1.3$.

Our implementation of CBS-MP differs slightly from the original in that, here, agents plan on discretized implicit graphs and not on precomputed roadmaps. This has been done to compare all search algorithms on the same planning representation. All edge transitions on the implicit graphs are said to take one timestep.

Evaluation Metrics and Postprocessing

We are interested in the scalability and solution quality of algorithms. To this end, for each scene, we report the mean and standard deviation for planning time and solution cost across all segments, alongside the success rate of each algorithm in the scene. All algorithms were allocated 60 seconds for planning, after which a plan was considered a failure. The cost is the total motion (radians) carried out by the joints.

All solutions are post-processed with a simple incremental shortcutting algorithm (Choset et al. 2005). One by one, each agent's solution path is shortcutted without creating new conflicts. Starting from the beginning of the path, the algorithm attempts to replace path segments by linear interpolations while avoiding obstacles and other agents.

Experimental Results

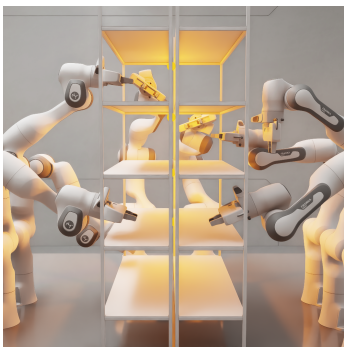
We observe that xECBS solves real-world multi-arm manipulation planning problems faster and with a higher success rate than other evaluated methods while keeping solution costs low. Both xCBS and xECBS improve on CBS and ECBS in general, however, xECBS offers a much larger boost in performance and is more suitable for M-RAMP.

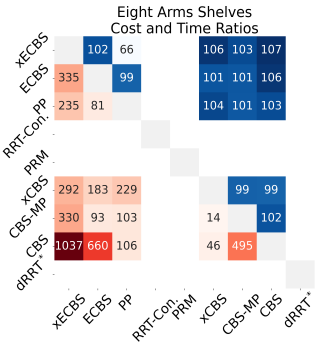
xECBS proved successful in our 4-robot bin-picking and 8-robot shelf rearrangement examples. Underscored by Fig. 3 (middle), xECBS demonstrates faster planning speed (red, above 100%) while delivering low-cost solutions comparable to those achieved by other CBS-based approaches (blue, values below or around 100%).

Our scalability analysis primarily focused on interactions between robots and minimized obstacle density. Here, xECBS scaled to scenes with more agents better than competing methods, consistently solving more problems. The experience reuse in xECBS allowed it to query a collision checker less than the other methods we implemented and aided in reducing its planning times.

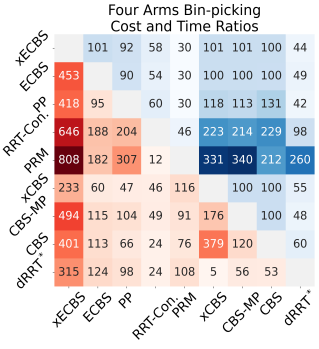
Across our experiments, the costs of all CBS-based methods, including xCBS and xECBS, were very similar. This stability sheds light on how experience reuse can maintain solution quality while also expediting the search. Sampling-based methods, however, faced notable challenges in scenarios with tight clutter and coordination, such as the 8-robot shelf rearrangement task. Not only did these methods struggle to find solutions, but the solutions they produced had high costs and variability.

[2] Our heuristic underestimates the cost to go in radians, and edges are unit cost. The weight w_1^L scales the heuristic value to match the cost of edge transitions and inflates it.





	Success	Planning Time (sec)	Cost (rad)
xECBS	**84%**	**13.6 ± 12.1**	**41.9 ± 8.3**
ECBS	40%	26.9 ± 17.9	37.1 ± 6.3
PP	40%	30.0 ± 19.7	58.1 ± 50.6
RRT-Con.	x	x	x
PRM	x	x	x
xCBS	4%	34.8 ± 25.4	41.5 ± 8.9
CBS	4%	39.8 ± 21.9	32.1 ± 4.9
CBS-MP	22%	25.7 ± 20.0	38.0 ± 4.6
dRRT*	x	x	x



	Success	Planning Time (sec)	Cost (rad)
xECBS	**96%**	**4.4 ± 3.4**	**24.1 ± 3.5**
ECBS	82%	18.4 ± 16.9	23.3 ± 3.3
PP	84%	14.5 ± 17.1	25.4 ± 14.1
RRT-Con.	42%	16.8 ± 14.2	42.0 ± 22.4
PRM	16%	9.3 ± 18.5	73.4 ± 46.2
xCBS	48%	8.6 ± 10.7	22.5 ± 2.9
CBS-MP	64%	14.4 ± 11.0	22.2 ± 2.6
CBS	28%	12.8 ± 11.2	21.8 ± 2.4
dRRT*	14%	6.1 ± 8.9	56.4 ± 20.8

Figure 3: Evaluating the real-world applicability of planning algorithms. Left: evaluation scenes, with 8-arm shelf rearrangement and 4-arm bin-picking. Middle: Comparing planning time and cost for each row-name planner relative to the column-name planner. The values offer a fair comparison by considering only successful runs in both planners. For instance, xECBS has shorter planning times (red, above 100%) and lower solution costs comparable to other CBS-based approaches (blue, around or below 100%). xECBS is faster and finds short paths. Right: Success rate and mean±standard deviation among successful runs.

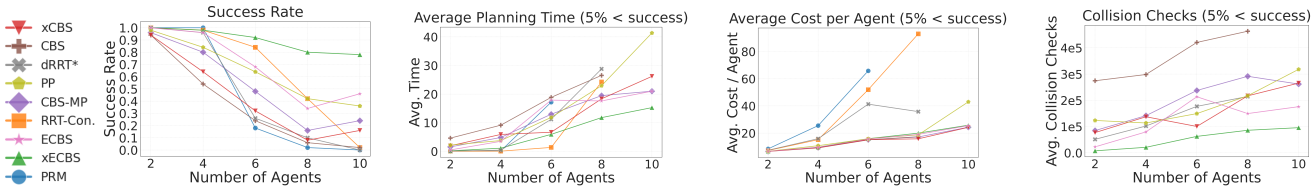


Figure 4: Scalability analysis. Top: our test scenes with 2, 4, 6, 8, and 10 robots. Bottom, from left to right: (a) success rate of methods in the 50 planning problems of each scene. xECBS scales better than competing methods. (b) Average planning time in successful runs. xECBS maintains a lower planning time as the number of robots increases. (c) Cost per agent in successful runs. All CBS-based methods maintain similar costs while PP and sampling-based methods eventually produce worse paths. (d) Average number of collision checks. (b,c,d) report on planners with at least 5% success to avoid unrepresentative data.

Conclusion

Popular multi-agent path finding algorithms like CBS and ECBS assume fast single-agent planners, which may not be available in multi-arm manipulation tasks. To address this, we propose to accelerate conflict-based algorithms by reusing online-generated path experiences and demonstrate their benefits in xCBS and xECBS. These adaptations improve performance in multi-arm manipulation tasks while ensuring bounded sub-optimality guarantees. Our experiments demonstrate the proposed method's effectiveness in various multi-arm manipulation tasks with up to 10 arms. We observe that xECBS is particularly effective in real-world scenarios such as pick and place and shelf rearrangement, achieving higher success rates and lower planning times than currently available methods.

Acknowledgements

The work was supported by the National Science Foundation under Grant 2328671 and the CMU Manufacturing Futures Institute, made possible by the Richard King Mellon Foundation.

References

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *International Symposium on Combinatorial Search*, 19–27.

Boyarski, E.; Felner, A.; Harabor, D.; Stuckey, P. J.; Cohen, L.; Li, J.; and Koenig, S. 2021. Iterative-Deepening Conflict-Based Search. In *International Joint Conferences on Artificial Intelligence*, 4084–4090.

Canny, J. 1988. *The complexity of robot motion planning*. MIT press.

Choset, H.; Lynch, K. M.; Hutchinson, S.; Kantor, G. A.; and Burgard, W. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT press.

Cohen, B. J.; Subramania, G.; Chitta, S.; and Likhachev, M. 2011. Planning for Manipulation With Adaptive Motion Primitives. In *Proceedings of IEEE International Conference on Robotics and Automation*, 5478–5485.

Coleman, D.; Sucan, I.; Chitta, S.; and Correll, N. 2014. Reducing the Barrier To Entry of Complex Robotic Software: a MoveIt! Case Study. *arXiv preprint arXiv:1404.3785*.

Dai, H.; Amice, A.; Werner, P.; Zhang, A.; and Tedrake, R. 2023. Certified Polyhedral Decompositions of Collision-Free Configuration Space. arXiv:2302.12219.

Erdmann, M.; and Lozano-Perez, T. 1987. On Multiple Moving Objects. *Algorithmica*, 477–521.

Fedus, W.; Ramachandran, P.; Agarwal, R.; Bengio, Y.; Larochelle, H.; Rowland, M.; and Dabney, W. 2020. Revisiting Fundamentals of Experience Replay. In *International Conference on Machine Learning*, 3061–3071.

Islam, F.; Salzman, O.; Agarwal, A.; and Likhachev, M. 2021. Provably Constant-Time Planning And Replanning For Real-Time Grasping Objects Off A Conveyor Belt. *The International Journal of Robotics Research*, 1370–1384.

Islam, F.; Salzman, O.; and Likhachev, M. 2019. Provable Indefinite-Horizon Real-Time Planning for Repetitive Tasks. In *International Conference on Automated Planning and Scheduling*, 716–724.

Karaman, S.; and Frazzoli, E. 2011. Sampling-Based Algorithms For Optimal Motion Planning. *The International Journal of Robotics Research*, 846–894.

Kavraki, L.; Svestka, P.; Latombe, J.-C.; and Overmars, M. 1996. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 566–580.

Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong Planning A*. *Artificial Intelligence*, 93–146.

Kuffner, J.; and LaValle, S. 2000. RRT-Connect: An Efficient Approach to Single-Query Path Planning. *IEEE International Conference on Robotics and Automation*.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing For Multi-agent Path Finding Via Large Neighborhood Search. In *AAAI Conference on Artificial Intelligence*, 10256–10265.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. In *AAAI Conference on Artificial Intelligence*, 12353–12362.

Li, J.; Surynek, P.; Felner, A.; Ma, H.; Kumar, T. S.; and Koenig, S. 2019. Multi-Agent Path Finding for Large Agents. In *AAAI Conference on Artificial Intelligence*, 7627–7634.

Likhachev, M.; Ferguson, D.; Gordon, G.; Stentz, A.; and Thrun, S. 2008. Anytime Search In Dynamic Graphs. *Artificial Intelligence*, 1613–1643.

Likhachev, M.; and Koenig, S. 2005. A Generalized Framework for Lifelong Planning A* Search. In *International Conference on Automated Planning and Scheduling*, 99–108.

Marcucci, T.; Petersen, M.; von Wrangel, D.; and Tedrake, R. 2022. Motion Planning Around Obstacles With Convex Optimization. *arXiv:2205.04422*.

Mishani, I.; Feddock, H.; and Likhachev, M. 2023. Constant-time Motion Planning with Anytime Refinement for Manipulation. arXiv:2311.00837.

Nilsson, N. J. 1982. *Principles of artificial intelligence*. Springer Science & Business Media.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 392–399.

Phillips, M.; Cohen, B.; Chitta, S.; and Likhachev, M. 2012. E-Graphs: Bootstrapping Planning with Experience Graphs. In *Robotics: Science and Systems*.

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

Shome, R.; Solovey, K.; Dobson, A.; Halperin, D.; and Bekris, K. E. 2020. dRRT*: Scalable and Informed Asymptotically-Optimal Multi-Robot Motion Planning. *Autonomous Robots*, 443–467.

Solis, I.; Motes, J.; Sandström, R.; and Amato, N. M. 2021. Representation-Optimal Multi-robot Motion Planning Using Conflict-based Search. *IEEE Robotics and Automation Letters*, 4608–4615.

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

Sucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics and Automation Magazine*, 72–82.

Veerapaneni, R.; Kusnur, T.; and Likhachev, M. 2023. Effective Integration of Weighted Cost-to-Go and Conflict Heuristic within Suboptimal CBS. *AAAI Conference on Artificial Intelligence*, 11691–11698.