Autonomous Robots
https://doi.org/10.1007/s10514-023-10148-y

Multi-robot geometric task-and-motion planning for collaborative manipulation tasks

Hejia Zhang[1] · Shao-Hung Chan[1] · Jie Zhong[1] · Jiaoyang Li[2] · Peter Kolapo[3] · Sven Koenig[1] · Zach Agioutantis[3] · Steven Schafrik[3] · Stefanos Nikolaidis[1]

Received: 4 March 2023 / Accepted: 26 September 2023

Abstract
We address multi-robot geometric task-and-motion planning (MR-GTAMP) problems in *synchronous*, *monotone* setups. The goal of the MR-GTAMP problem is to move objects with multiple robots to goal regions in the presence of other movable objects. We focus on collaborative manipulation tasks where the robots have to adopt intelligent collaboration strategies to be successful and effective, i.e., decide which robot should move which objects to which positions, and perform collaborative actions, such as handovers. To endow robots with these collaboration capabilities, we propose to first collect occlusion and reachability information for each robot by calling motion-planning algorithms. We then propose a method that uses the collected information to build a graph structure which captures the precedence of the manipulations of different objects and supports the implementation of a mixed-integer program to guide the search for highly effective collaborative task-and-motion plans. The search process for collaborative task-and-motion plans is based on a Monte-Carlo Tree Search (MCTS) exploration strategy to achieve exploration-exploitation balance. We evaluate our framework in two challenging MR-GTAMP domains and show that it outperforms two state-of-the-art baselines with respect to the planning time, the resulting plan length and the number of objects moved. We also show that our framework can be applied to underground mining operations where a robotic arm needs to coordinate with an autonomous roof bolter. We demonstrate plan execution in two roof-bolting scenarios both in simulation and on robots.

Keywords Task-and-motion planning · Multi-robot collaboration · Collaborative manipulation · Mining robotics

1 Introduction

Task-and-motion planning (TAMP) is the problem of combining task and motion planning to divide an objective, such as assembling a table, into a series of robot-executable motion trajectories (Garrett et al. 2021). Task planning is used to generate a sequence of discrete actions, such as picking up a screwdriver and driving a screw, while motion planning is used to compute the actual trajectories the robot should execute.

Geometric task-and-motion planning (GTAMP) is an important subclass of TAMP where the robot has to move

✉ Hejia Zhang
hejiazha@usc.edu

Shao-Hung Chan
shaohung@usc.edu

Jie Zhong
jzhong54@usc.edu

Jiaoyang Li
jiaoyangli@cmu.edu

Peter Kolapo
peter.kolapo@uky.edu

Sven Koenig
skoenig@usc.edu

Zach Agioutantis
zach.agioutantis@uky.edu

Steven Schafrik
steven.schafrik@uky.edu

Stefanos Nikolaidis
nikolaid@usc.edu

[1] Thomas Lord Department of Computer Science, University of Southern California, Los Angeles, USA

[2] The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA

[3] Department of Mining Engineering, University of Kentucky, Lexington, USA

Published online: 30 October 2023

several objects to regions in the presence of other movable objects (Kim et al. 2019). GTAMP has been addressed efficiently in single-robot domains (Kim et al. 2019; Kim and Shimanuki 2020; Kim et al. 2022). We focus on *multi-robot geometric task-and-motion planning* (MR-GTAMP), where several robots have to collaboratively move several objects to regions in the presence of other movable obstacles.

MR-GTAMP naturally arises in many multi-robot manipulation domains, such as multi-robot construction, multi-robot assembly and autonomous warehousing (Chen et al. 2022; Hartmann et al. 2021). MR-GTAMP is interesting as multi-robot systems can perform manipulation tasks more effectively than single-robot systems and can also perform manipulation tasks that are beyond the capabilities of single-robot systems (Shome et al. 2021). For example, in a product-packaging task, a single robot may have to move a lot of objects to clear a path to grasp an object, while a two-robot system can easily perform a handover action to increase the effectiveness of task execution.

Examples of MR-GTAMP problem instances are shown in Fig. 1. The example task shown in Fig. 1 (left) requires multiple robotic arms to sort colored objects into boxes of corresponding colors in a confined workspace. The example task shown in Fig. 1 (right) requires multiple mobile manipulators to move green objects to the green region. In both tasks, white objects are movable obstacles and are only allowed to be relocated within their current regions. These example tasks embody the key challenges that MR-GTAMP aims to address. First, they are in a hybrid discrete-continuous planning space which is extremely large when multiple robots are involved (Pan et al. 2021; Kim et al. 2022). This involves high-level task planning, which decides which robot should move which objects and in what sequence, and low-level motion planning, which decides the positions to which objects should be relocated and the motion trajectories robots should follow. Second, in both scenarios, robots work in a confined workspace and have to consider geometric constraints imposed by the environments and the tasks carefully. Finally, robots must collaborate intelligently to perform tasks effectively. For example, robots can achieve their targets more quickly by concurrently manipulating multiple objects, and they can avoid relocating too many objects by performing handover actions.

We address the following research question: How can we enable multiple robots to perform GTAMP tasks effectively and efficiently?

Determining effective collaborative action sequences for multiple robots is difficult as manipulation planning in the presence of movable obstacles has been shown to be NP-hard for single-robots (Stilman et al. 2007; Hun Cheong et al. 2020). MR-GTAMP is even harder since one needs to decide which robot should move which objects to which positions.

Our key insight to solving MR-GTAMP efficiently is that *we can compute information about the manipulation capabilities of individual robots and their potential collaborative relationships by calling motion-planning algorithms* and then use it to prune the search space and guide the search process. For example, based on the information that a robot cannot reach an object, we can eliminate all task plans that involve the action where the robot has to reach the object. Moreover, the computed information can be used to generate collaborative plans where each robot performs the tasks that it excels at.

We propose a two-phase framework. In the first phase, we compute the collaborative manipulation information, i.e., the occlusion and reachability information for individual robots and the potential collaborative relationships between them (Sec. 4.1). In the second phase, we search for collaborative task-and-motion plans using a Monte-Carlo Tree Search (MCTS) exploration strategy due to its good exploration-exploitation balance (Sec. 4.2). Our search algorithm is based on two key components: (i) The first key component uses the collected information from the first phase to generate promising task skeletons for moving a specified set of objects by formulating a series of mixed-integer linear programs (MIPs), that can be solved efficiently by leveraging recent developments in MIP solvers (Cplex 2009) (Sec. 4.2.1). The term *task skeleton* represents a sequence of actions that are missing continuous parameters required for execution. The missing continuous parameters include the intended positions for objects that need to be relocated, and the motion trajectories that the robots should follow to relocate these objects. The formal definition of *task skeleton* can be found in Sec. 3. (ii) The second key component efficiently finds feasible continuous parameters for the generated task skeletons, such as the locations to which to relocate objects (Sec. 4.2.2).

Fig. 1 Left: Sorting colored objects into boxes of corresponding colors. Right: Moving the colored boxes to the green region. In both scenarios, white objects are only allowed to be relocated within their current regions (red). We use PyBullet (Coumans and Bai 2016) as our simulator (Color figure online)

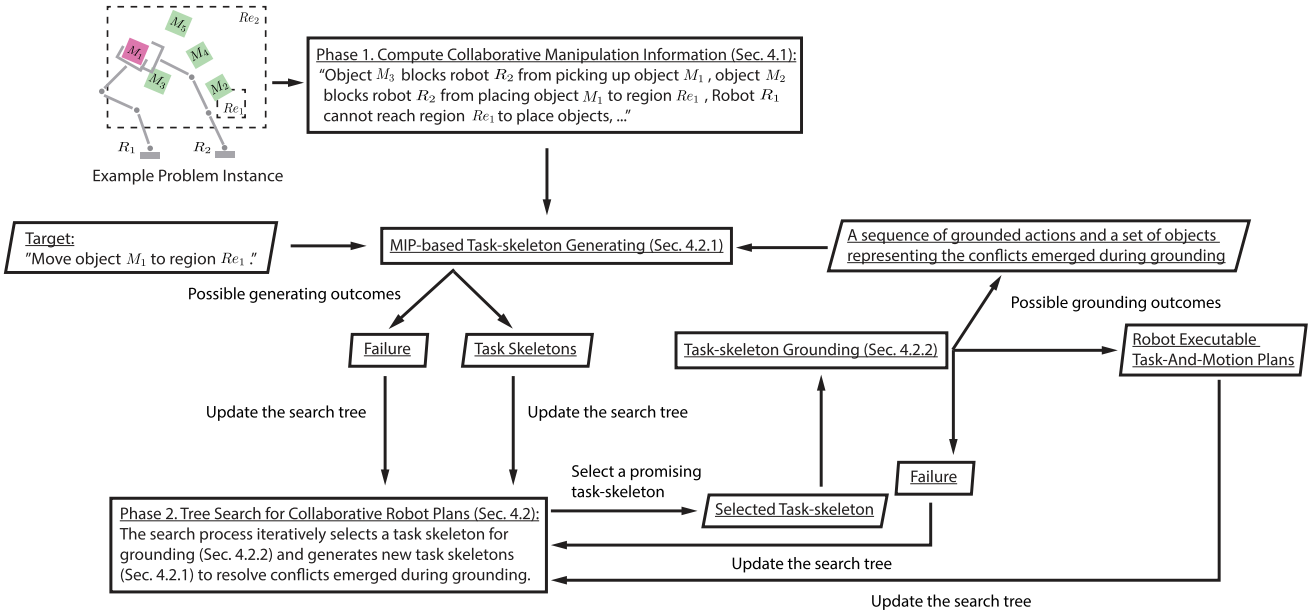


Fig. 2 Overview of the proposed framework. Fig. 3 provides a more detailed visualization and description of Phase 2.

We denote the process of finding continuous parameters to make a task skeleton executable as *grounding*. Fig. 2 presents an overview of our framework.

We compare our framework with two state-of-the-art baselines, namely, a general MR-TAMP framework (Pan et al. 2021) and a multi-robot extension of the ResolveSpatialConstraints (RSC) algorithm (Stilman et al. 2007). We evaluate our framework in two challenging MR-GTAMP domains and show that it outperforms two state-of-the-art baselines with respect to the planning time, the resulting plan length and the number of objects moved (Sec. 5).

We also conducted an application study and show that our framework can be used to coordinate a robotic arm with an autonomous roof bolter for underground mining operations. We demonstrate the execution of the computed plans in two example roof-bolting scenarios both in simulation and on robots.

Our work makes the following assumptions, which are common in MR-TAMP (Shome et al. 2021; Pan et al. 2021): (i) It considers only *monotone* instances of the MR-GTAMP problem, where each object is moved only once. The monotone problems are common in less constrained environments such as home environments and relate to a range of warehouse applications such as packing and stowing (Shome et al. 2021). (ii) It assumes the robots *synchronously* start and stop the executions of actions. We plan to relax these assumptions in future work.

This work is an extended version of our prior paper (Zhang et al. 2022). We make the following additional contributions.

- We conduct an application study on the roof-bolting task, which is an essential operation within the underground mining cycle. We show that the roof-bolting task can be formulated as MR-GTAMP problems and addressed efficiently with the proposed planning framework. We demonstrate plan execution in two roof-bolting scenarios both in simulation and on real robots.
- We conduct additional scalability evaluation experiments to study the performance change of our framework when more robots are involved.
- We substantially expand the description of the task-skeleton grounding component and the tree search algorithm.

2 Related work

There has been much work on solving general TAMP problems efficiently. TAMP problems are challenging because they require search in a large hybrid space that consists of task-level search and motion-level search. Different approaches for TAMP problems focus on different strategies to combine task-level search and motion-level search. In Lagriffoul et al. (2014); Bidot et al. (2017), efficient geometric backtracking algorithms are proposed to systematically consider all the combinations of geometric instances of a given symbolic task plan such that the symbolic task plan can be efficiently rejected if there is no way to instantiate it geometrically. In Dantam et al. (2018), task-level search is modeled as a constraint satisfaction problem and failures on motion-level search are efficiently encoded as new con-

straints to inform task-level search. In Srivastava et al. (2014), an extensible planner-independent interface layer is proposed to combine task and motion planning. In Garrett et al. (2020), motion-level facts are encoded in task-level planning and modern task planners (Hoffmann 2001) are leveraged to efficiently search for task-and-motion plans. Recently, more and more work has been focused on utilizing learning to guide TAMP by ranking task plans (Khodeir et al. 2023), predicting feasibility of task plans (Yang et al. 2022), and ranking object importance in problem instances (Silver et al. 2020). More comprehensive surveys on TAMP can be found in Garrett et al. (2021); Mansouri et al. (2021).

In this work, we focus on GTAMP which is an important subclass of TAMP. The goal of the GTAMP is to move several objects to regions in the presence of other movable objects.

There has been much work on solving single-robot GTAMP (SR-GTAMP) problems efficiently (Kim et al. 2022, 2019; Kim and Shimanuki 2020) by utilizing learning to guide planning. However, these approaches cannot be directly applied to multi-robot domains. Several problem types in the literature can also be seen as versions of the GTAMP problem. In Stilman et al. (2007), the "manipulation among movable obstacles" (MAMO) problem is addressed, in which a robot has to move objects out of the way to move a specified object to its goal location. Although this approach can be extended to multi-robot settings straightforwardly, it would require searching through a large space of all possible combinations of multi-agent actions. Moreover, the focus of this approach is on feasibility of the task-and-motion plans, rather than on the plan length and number of objects moved. In Hun Cheong et al. (2020) and Nam et al. (2020); Danielczuk et al. (2019), the object retrieval problem is addressed, in which a robot has to retrieve a target object from clutter by relocating the surrounding objects. In King et al. (2016); Krontiris and Bekris (2016), the rearrangement planning problem is addressed, in which a robot has to move objects into given goal configurations. However, these methods do not plan collaboration strategies in multi-robot domains.

There has been work on solving general TAMP with several robots efficiently (Pan et al. 2021; Toussaint and Lopes 2017; Mansouri et al. 2021). We focus on a subclass of these problems, where a robot has to move objects in the presence of movable obstacles. In Pan et al. (2021), a novel task scheduling layer, positioned between task planning and motion planning, is proposed to prune task planning search space. However, since this approach does not focus on geometric aspects of the TAMP problem, it does not include guidance for finding continuous parameters such as feasible positions for object relocation. In Rodríguez and Suárez (2016); Ahn et al. (2021), efficient approaches are proposed for the multi-robot object retrieval problem, assuming permanent object removal and considering one target object at a time, while our planner relocates the obstacles within the workspace and considers several target objects at the same time. Multi-robot rearrangement planning problems (Shome et al. 2021; Hartmann et al. 2021; Chen et al. 2022) are also closely related to MR-GTAMP. However, the rearrangement planning problems assume that the goal configurations of all the movable objects are given, while MR-GTAMP requires the planners to decide which objects to move and to which positions. There is also work that focuses on task allocation and scheduling for multiple robots, assuming that a sequence of discrete actions to be executed is given (Behrens et al. 2020). However, MR-GTAMP requires the planners to decide which discrete actions to execute, e.g., which objects to move.

There has been work on optimization-based TAMP, where TAMP problems are modeled as mixed-integer non-linear programs (Toussaint, 2015; Toussaint & Lopes, 2017), mixed-integer linear programs (Kogo et al., 2021) and continuous nonlinear programs (Takano et al., 2021). However, these frameworks do not focus on scenarios where obstacle avoidance is the major challenge and objects can be moved to enable the manipulation of other objects.

3 Problem formulation

In an MR-GTAMP problem, we have a set of $n_{\mathbf{R}}$ robots $\mathbf{R} = \{R_i\}_{i=1}^{n_{\mathbf{R}}}$, a set of fixed rigid objects $\mathbf{F}$, a set of $n_{\mathbf{M}}$ movable rigid objects $\mathbf{M} = \{M_i\}_{i=1}^{n_{\mathbf{M}}}$ and a set of $n_{\mathbf{Re}}$ regions $\mathbf{Re} = \{Re_i\}_{i=1}^{n_{\mathbf{Re}}}$. We assume that all objects and regions have known and fixed shapes. The focus of our work is not on grasp planning (Quispe et al. 2016). So, for simplicity, we assume a fixed set of grasps $\mathbf{Gr}_{M,R}$ for each object $M \in \mathbf{M}$ and robot $R \in \mathbf{R}$ pair. $\mathbf{Gr}$ is the union of the sets of grasps for all object and robot pairs.

Each object has a configuration, which includes its position and orientation. Each robot has a configuration defined in its base pose space and joint space. We are given the initial configurations of all robots, objects and a goal specification $\mathcal{G}$ in form of a conjunction of statements of the form $\textsc{InRegion}(M, Re)$, which is true *iff* object $M \in \mathbf{M}$ is contained entirely in region $Re \in \mathbf{Re}$. An example goal specification is $(\textsc{InRegion}(M_1, Re_1) \wedge \textsc{InRegion}(M_2, Re_1))$ which indicates the target that we want to move objects M_1 and M_2 to region Re_1.

We define a grounded joint action as a set of $n_{\mathbf{R}}$ actions and motions performed by all the robots at one time step, i.e., the grounded joint action at time step j is an $n_{\mathbf{R}}$-tuple $s_j = \langle (a^j_{R_1}, \xi^j_{R_1}), (a^j_{R_2}, \xi^j_{R_2}), \ldots, (a^j_{R_{n_{\mathbf{R}}}}, \xi^j_{R_{n_{\mathbf{R}}}}) \rangle$, where each action a is a pick-and-place action or a wait[1] action that the corresponding robot executes and motion ξ is a trajectory that the corresponding robot executes, specified as a sequence of

[1] As in Pan et al. (2021), a robot with a wait action does not have to do anything but can move to avoid other robots.

robot configurations. In this work, we focus on pick-and-place actions because of their importance in robotic manipulation in cluttered space. Each pick-and-place action is a tuple of the form $\langle M, Re, R^{pick}, R^{place}, g^{pick}, g^{place}, P_M^{place}\rangle$, where M represents the object to move; Re represents the target region for M; R^{pick} and R^{place} represent the robots that pick and place M, respectively; g^{pick} and g^{place} represent the grasps used by R^{pick} and R^{place}, respectively, and P_M^{place} represents the configuration at which to place M. Moreover, we call a pick-and-place action whose R^{pick} is different from R^{place} a handover action. Each grounded joint action maps the configurations of the movable objects to new configurations and the unaffected objects remain at their old configurations.

We define a partially grounded joint action as an $n_{\mathbf{R}}$-tuple of the form $\langle \bar{a}_{R_1}, \ldots, \bar{a}_{R_{n_{\mathbf{R}}}}\rangle$, where $\bar{a}$ is a wait action or a pick-and-place action without the placement information P_M^{place}. We refer to a pick-and-place action without the placement information as a partially grounded pick-and-place action since it has only the information about the grasps that will be used.

We define a task skeleton $\bar{\mathbf{S}}$ as a sequence of partially grounded joint actions. We want to find a task-and-motion plan, i.e., a sequence of grounded joint actions $\mathbf{S}$ that changes the configurations of the objects to satisfy $\mathcal{G}$.

We denote the process of finding feasible object placements and motion trajectories for a task skeleton as *grounding*.

A task-and-motion plan is valid *iff*, at each time step j: (i) the corresponding multi-robot trajectory $\Xi^j = \langle \xi_{R_1}^j, \xi_{R_2}^j, \ldots, \xi_{R_{n_{\mathbf{R}}}}^j\rangle$ is collision-free; (ii) the robots can use the corresponding motion trajectories and grasp poses to grasp the target objects and place them at their target configurations without collisions; and (iii) all handover actions can be performed without inducing collisions. The considered collisions include collisions between robots, collisions between an object and a robot and collisions between objects.

4 Our approach

We present our two-phase MR-GTAMP framework (Fig. 2) in this section. In the first phase, we compute the collaborative manipulation information, i.e., the occlusion and reachability information for individual robots and whether two robots can perform a handover action for an object (Sec. 4.1). In the second phase, we use a Monte-Carlo Tree Search exploration strategy to search for task-and-motion plans (Sec. 4.2). The search process depends on a key component that generates promising task skeletons (Sec. 4.2.1) and a key component that finds collision-free object placements and trajectories for the task skeletons to construct valid task-and-motion plans (Sec. 4.2.2).

4.1 Computing collaborative manipulation information

Given an MR-GTAMP problem instance and the initial configurations of all objects and robots, our framework first computes the occlusion and reachability information for individual robots, e.g., whether an object blocks a robot from manipulating another object and whether a robot can reach a region to place an object there. We also compute whether two robots can perform a handover action for an object by computing whether they can both reach a predefined handover point to transfer the object. In this work, we consider only handover actions for objects that are named in goal specification $\mathcal{G}$ for computational simplicity. We assume that all robots return to their initial configurations after each time step. Inspired by Kim et al. (2022), we use the conjunction of all true instances of a set of predicates to represent the computed information. To define these predicates, we define two volumes of the workspace similar to Stilman et al. (2007); Kim et al. (2022). The first volume $V_{pick}(M, g, R, \xi)$ is the volume swept by robot R to grasp object M with grasp g following trajectory ξ. The second volume $V_{place}(M, g, R, P_M^{place}, \xi)$ is the volume swept by robot R and object M to transfer the object to configuration P_M^{place} following trajectory ξ. Our predicates are as follows:

- OccludesPick(M_1, M_2, g, R) is true *iff* object M_1 overlaps with the swept volume $V_{pick}(M_2, g, R, \xi)$, where ξ is chosen to be collision-free with all the objects except M_2, if possible;
- OccludesGoalPlace(M_1, M_2, Re, g, R) is true *iff* M_1 is an object that overlaps with the swept volume $V_{place}(M_2, g, R, P_{M_2}^{place}, \xi)$, where $P_{M_2}^{place}$ and ξ are chosen to be collision-free with all the objects except M_2, if possible, and the pair $\langle M_2, Re\rangle$ is named in goal specification $\mathcal{G}$;
- ReachablePick(M, g, R) is true *iff* there exists a trajectory for robot R to pick object M with grasp g;
- ReachablePlace(M, Re, g, R) is true *iff* there exists a trajectory for robot R to place object M into region Re with grasp g; and
- EnableGoalHandover(M, g_1, g_2, R_1, R_2) is true *iff* robots R_1 and R_2 can both reach a predefined handover point for object M with grasps g_1 and g_2, respectively, and the object M is named in goal specification $\mathcal{G}$.

For a predicate instance to be true, the corresponding trajectories are required to be collision-free with respect to all fixed objects. For a predicate instance of EnableGoalHandover to be true, the two robots are required to not collide with each other.

The values of all predicate instances can be computed with existing inverse-kinematics solvers (Diankov 2010) and motion planners (LaValle 2006). Ideally, we wish to find trajectories for the robots that have the minimum number of collisions with all objects, i.e., the *minimum constraint removal* (Hauser 2013) trajectories. However, this is known to be very time consuming. Thus, we follow previous work (Kim et al. 2022) and first attempt to find a collision-free trajectory with respect to all movable and fixed objects. If we fail, we attempt to find a collision-free trajectory with respect to only the fixed objects.

In our implementation, we efficiently compute the predicates – with the exception of EnableGoalHandover – in parallel for all robots by creating an identical simulation environment for each robot.

4.2 Searching for task-and-motion plans

We now describe our search process (Fig. 3) for efficiently finding effective collaborative task-and-motion plans. Our search process is initialized with a set of task skeletons, that is generated for moving the set of objects named in the goal specification, utilizing the computed collaborative manipulation information (Sec. 4.1). We will describe our key component for generating task skeletons in detail in Sec. 4.2.1. We then generate a search tree with a root node, denoted as D_0 as shown in Fig. 3 (left). We associate an empty sequence of grounded joint actions with node D_0, denoted as $D_0.\mathbf{S} = \emptyset$. We use the "." operator to denote the association relationship. This implies that at node D_0, we do not have any grounded joint actions. We then create edges originating from node D_0, with each edge storing a distinct initial task skeleton.

Throughout our search process, at each search iteration, we select an edge that has not been evaluated yet, and we evaluate it by trying to ground the task skeleton associated with it. As previously defined, the term *grounding* refers to the process of finding feasible object placements and motion trajectories for a task skeleton to be executable. After each evaluation, we compute a reward based on the evaluation result. The reward will then be propagated back up the search tree, with each edge in the path from the root node to the selected edge having its value updated based on the reward. We use a Monte-Carlo Tree Search (MCTS) exploration strategy to balance exploration (exploring different unevaluated edges) and exploitation (biasing the search towards the branches that have received high rewards).

We use a reverse search algorithm inspired by Stilman et al. (2007) to ground task skeletons. We will describe our key component for task-skeleton grounding in detail in Sec. 4.2.2. The insight behind the reverse search algorithm is to use the grounded future joint actions as the artificial constraints to guide the grounding for the current actions. Therefore, throughout our search process, we save the grounding results and use them as artificial constraints for subsequent ground-

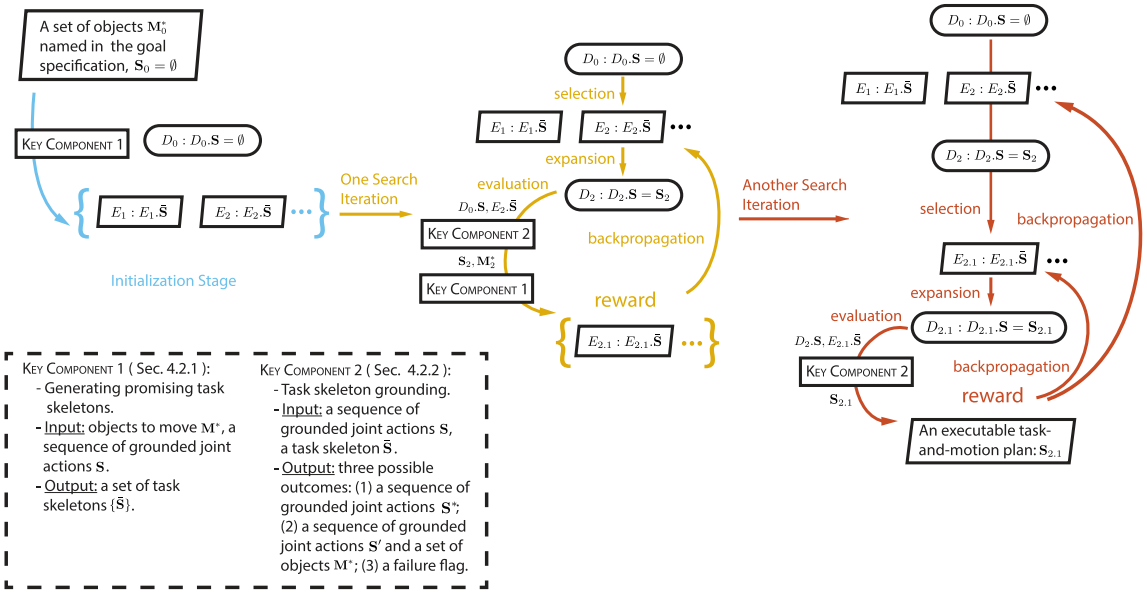


Fig. 3 Visualization of the search process in the second phase of our framework. We show the initialization stage of the search process (left) and two example search iterations (middle, right) that lead to different evaluation outcomes. Left: Blue arrows represent the workflow for initializing the search tree. Middle: Yellow arrows represent a search iteration that results in an updated set of objects to be moved and thus a new set of task skeletons to be grounded. Right: Red arrows represent a search iteration that results in an executable task-and-motion plan (Color figure online)

ing tasks. We use two examples, as shown in Fig. 3 (middle, right), to illustrate the idea.

In the first example (Fig. 3 (middle)), we select edge E_2 for evaluation. We create a new node, denoted as D_2, to serve as the head node of edge E_2. The tail node of edge E_2 is the root node D_0 whose associated sequence of grounded joint actions is empty. This means that we can attempt to ground the task skeleton associated with E_2, denoted as $E_2.\bar{\mathbf{S}}$, without any artificial constraints. Ideally, if we manage to ground task skeleton $E_2.\bar{\mathbf{S}}$ successfully, we would get an executable task-and-motion plan to perform the task. However, in many situations, we can only ground the task skeleton partially. This implies that there are conflicts that emerge during task-skeleton grounding. For example, there would not be enough space to place objects unless we relocate some objects that were not planned to be moved initially. Such situations can arise as we cannot account for all geometric specifics during task-skeleton generation. In such situations, we generate new task-skeletons to address the emerged conflicts, and we expand the tree by creating new edges, with each edge storing a distinct new task skeleton. In our first example, we create new edges originating from node D_2. Moreover, we store the sequence of joint actions that have been grounded to this point in node D_2, denoted as $D_2.\mathbf{S}$. It should be noted that $D_2.\mathbf{S}$ contains $D_0.\mathbf{S}$ and the grounded part of $E_2.\bar{\mathbf{S}}$.

In the second example (Fig. 3 (right)), we select edge $E_{2.1}$ for evaluation. The grounding of the task skeleton associated with edge $E_{2.1}$, denoted as $E_{2.1}.\bar{\mathbf{S}}$, should consider $D_2.\mathbf{S}$ as artificial constraints which is the sequence of joint actions that have been grounded to this point. If we successfully ground $E_{2.1}.\bar{\mathbf{S}}$, we can get an executable task-and-motion plan by concatenating the grounded task-skeleton with $D_2.\mathbf{S}$.

At each search iteration, we have four phases: *selection*, *expansion*, *evaluation* and *backpropagation*.

Notation. We use $|\mathbf{S}|$ and $|\bar{\mathbf{S}}|$ to denote the number of objects intended to be moved in sequences of grounded joint actions $\mathbf{S}$ and task skeletons $\bar{\mathbf{S}}$, respectively.

***Selection* phase.** In the *selection* phase, we start at the root node and recursively select the edge with the highest Upper Confidence Bound (UCB) value until we reach an edge E_i with a task skeleton that has not been grounded yet. We denote the tail node of edge E_i as D_j. We follow the UCB value formula used in Silver et al. (2017). The UCB value of the pair of node D_j and edge E_i is: $Q(D_j, E_i) = \frac{E_i.value}{E_i.visits+1} + c \times E_i.prior \times \frac{\sqrt{D_j.visits}}{E_i.visits+1}$, where $E_i.value$ is the cumulative reward edge E_i has received so far, $D_j.visits$ and $E_i.visits$ are the number of times D_j and E_i have been selected, c is a constant to balance exploration and exploitation, and $E_i.prior$ is used to bias the search with domain knowledge (Silver et al. 2017). In our implementation, we set $E_i.prior$ to $\frac{1}{|E_i.\bar{\mathbf{S}}|}$ to prioritize grounding task skeletons with fewer objects to move. The value $E_i.value$ of an edge is initialized to 0.

Assume that we select edge E_i from node D_j in the *selection* phase.

***Expansion* phase.** In the *expansion* phase, we create a new node D_i as the head node of edge E_i.

***Evaluation* phase.** In the *evaluation* phase, we use the task-skeleton grounding component (Sec. 4.2.2) to ground task skeleton $E_i.\bar{\mathbf{S}}$ associated with E_i to compute reward r for selecting edge E_i. Note that node D_j is the tail node of edge E_i and the grounded sequence of joint actions stored in node D_j is denoted as $D_j.\mathbf{S}$. There are three possible outcomes: (i) If we fail at grounding, we set r to 0. (ii) If we obtain a sequence of grounded joint actions $\mathbf{S}^*$, then we found a valid task-and-motion plan. In this case, we set r to $1+\alpha\frac{1}{|\mathbf{S}^*|}$, where α is a constant hyperparameter used to balance the two terms of the reward that is set to 1 in our experiments (Sec. 5). The first term of the reward incentivizes the search algorithm to select edges where more actions have been grounded, and the second term incentivizes the search algorithm to select edges that move fewer objects. (iii) In the third case, task skeleton $E_i.\bar{\mathbf{S}}$ cannot be fully grounded without relocating some objects that are not planned to be moved in $E_i.\bar{\mathbf{S}}$. In this case, we obtain a sequence of grounded joint actions $\mathbf{S}'$ and a set of objects $\mathbf{M}^*$ from the grounding process. Here, $\mathbf{S}'$ consists of $D_j.\mathbf{S}$ and the grounded part of task skeleton $E_i.\bar{\mathbf{S}}$. We use $\mathbf{M}^*$ to represent the set of objects for which we need to find a sequence of grounded joint actions, denoted as $\mathbf{S}_{\mathbf{M}^*}$, to relocate so that we can construct a final task-and-motion plan for the problem by concatenating $\mathbf{S}_{\mathbf{M}^*}$ with $\mathbf{S}'$. We then call the task-skeleton generating component (Sec. 4.2.1) to move $\mathbf{M}^*$. If we cannot find any task skeleton to move $\mathbf{M}^*$, then we set r to 0. However, if we find a set of task skeletons $\{\bar{\mathbf{S}}\}$, then we set r to $\frac{\mathbf{S}'.length}{\mathbf{S}'.length+\bar{\mathbf{S}}^*.length} + \alpha\frac{1}{|\mathbf{S}'|+|\bar{\mathbf{S}}^*|}$, where $\bar{\mathbf{S}}^*$ is the task skeleton with the minimum number of time steps among all task skeletons $\{\bar{\mathbf{S}}\}$ and $\mathbf{S}'.length$ and $\bar{\mathbf{S}}^*.length$ represent the number of time steps of $\mathbf{S}'$ and $\bar{\mathbf{S}}^*$, respectively.

We would like to point out that the reward in the second possible outcome represents a special case of the reward in the third possible outcome. Both rewards use their first terms to incentivize the search algorithm to select edges where more actions have been grounded, and their second terms to incentivize the search algorithm to select edges that move fewer objects.

We use node D_i to store the returned grounded joint actions $\mathbf{S}'$ as $D_i.\mathbf{S}$. In the third scenario, if we find new task skeletons, then we create new edges to store them for node D_i. If no new edge is created, then we mark node D_i as a terminal node.

***Backpropagation* phase.** In the *backpropagation* phase, we update the cumulative reward of the selected edges $\{E^{sel}\}$ with the computed reward r according to $E^{sel}.value =$

$E^{sel}.value + r$. We also increment the number of visits of the selected edges and nodes by 1.

In our implementation, we track the grounding failures for different task skeletons similarly to Ren et al. (2021), so that we can skip over those branches where grounding their task skeletons is known to be infeasible.

4.2.1 Key component 1: generating promising task skeletons

One key component in the second phase of our framework is to generate promising task skeletons $\{\bar{\mathbf{S}}\}$ for moving a set of objects $\mathbf{M}^*$ given a sequence of already grounded joint actions $\mathbf{S}'$. As previously defined, the term *task-skeleton* refers to a sequence of actions without the placement and trajectory information. This key component will be used in two situations. It is firstly called at the initialization stage of the search process (Fig. 3 (left)). In this situation, we set $\mathbf{S}'$ as empty and set $\mathbf{M}^*$ as the set of objects named in the goal specification of the problem instance. We will use the generated task skeletons to initialize the search tree as shown in Fig. 3 (left). The second scenario where this component is called is when we can only ground part of a task skeleton in the *evaluation* phase during the search process. Figure 3 (middle) depicts one example search iteration where this situation happens. In this example search iteration, we set $\mathbf{S}'$ as $\mathbf{S}_2$ and set $\mathbf{M}^*$ as $\mathbf{M}_2^*$. We use this key component to generate task skeletons to relocate $\mathbf{M}_2^*$. We take $\mathbf{S}'$ as input because we should exclude objects from our task-skeleton generation that are already planned to be moved in $\mathbf{S}'$. The task-skeleton generation algorithm is designed to utilize the computed collaborative manipulation information from the first phase (Sec. 4.1) to eliminate task skeletons that include infeasible actions and to prioritize motion planning for effective task plans that have fewer time steps and fewer objects to be moved.

Notation. Assume that we want to generate task skeletons to move objects $\mathbf{M}^*$ given a sequence of grounded joint actions $\mathbf{S}'$. The set of objects included in $\mathbf{S}'$ cannot be moved again because of the *monotone* assumption. For simplicity of presentation, we slightly abuse $\mathbf{M}$ to denote the movable objects not included in $\mathbf{S}'$.

Building the collaborative manipulation task graph.

To reason about the collaborative manipulation capabilities of the individual robots, we encode the computed information as a graph. We build a *collaborative manipulation task graph* (CMTG) to capture the precedence of the manipulations of different objects, i.e., we can only move an object after we move the obstacles that block the pick-and-place action we are going to execute, based on the computed information from the first phase (Sec. 4.1). Since we only compute occlusion information for placing objects named in the goal specification, the precedence encoded in the CMTG lack occlusion information for relocating objects that are not named in the goal specification. Instead, we assume that we will always find the feasible places to relocate these objects. We determine the exact object placements during task-skeleton grounding (Sec. 4.2.2).

A CMTG (Fig. 4) has two types of nodes: An *object node* represents an object $M \in \mathbf{M}$; and an *action node* represents a partially grounded pick-and-place action $\bar{a}$, i.e. a pick-and-place action without placement information. A CMTG has three types of edges: An *action edge* is an edge from an object node to an action node. It represents moving the object represented by the object node with the action represented by the action node. A *block-pick edge* is an edge from an action node to an object node. It represents that the object represented by the object node obstructs the pick action of the action represented by the action node. A *block-place edge* is an edge from an action node to an object node. It represents that the object represented by the object node obstructs the place action of the action represented by the action node. All *block-place edges* are connected to the action nodes that move the objects named in the goal specification. A CMTG has a set of object nodes that represents the input objects $\mathbf{M}^*$ that must be moved.

Given the computed collaborative manipulation information and a set of objects $\mathbf{M}^*$ to move, we incrementally construct a CMTG by iteratively adding object $M \in \mathbf{M}^*$ to the CMTG with Alg. 1. Given the CMTG $\mathbf{C}$ built so far and an object M to add, we first add an object node representing M to $\mathbf{C}$ (Alg. 1, line 4). Then, for each pair of a robot $R \in \mathbf{R}$ and its grasp $g_{M,R} \in \mathbf{Gr}_{M,R}$, we find all partially grounded pick-and-place actions $\mathbf{A}$ that move object M to its target region Re_M with R as the pick robot (Alg. 1, line 5-20). For each partially grounded pick-and-place action $\bar{a}$, we find all movable objects that block the pick action of $\bar{a}$ and add the corresponding block-pick edges (Alg. 1, line 28-31). If M is named in goal specification $\mathcal{G}$, then we also find all movable objects that block the place action of $\bar{a}$ and add the corresponding block-place edges (Alg. 1, line 32-36). We recursively add the blocking objects in a similar way (Alg. 1, lines 30 and 35).

Mixed-integer linear program formulation and solving. Given a CMTG $\mathbf{C}$, we find a set of task skeletons that specify which robot will move which object at each time step. We assume that each object will be moved at most once, i.e., we assume that the problem instances are *monotone*. Given a time step limit T, we cast the problem of finding a task skeleton that has a minimum number of objects to be moved as a mixed-integer linear program (MIP). We encode the precedence of manipulating different objects as formal constraints in the MIP such that we can generate task skeletons that are promising to be successfully grounded. We incrementally increase the time step limit T. In our implementation, the maximum time step limit is a hyperparameter.

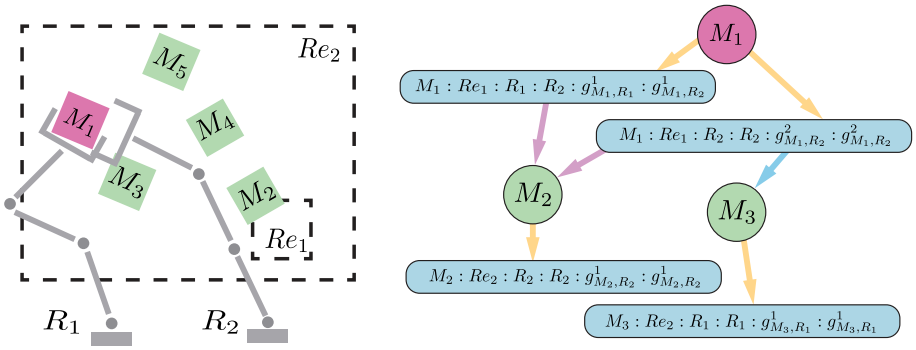


Fig. 4 (Left) An example scenario where we want to generate task skeletons to move object $\mathbf{M}_1$ given an empty sequence of grounded joint actions. (Right) The corresponding *collaborative manipulation task graph* for moving object M_1. The rounded rectangular nodes are *action nodes*. The circular nodes are *object nodes*. The red circular nodes represent objects that are specified to be moved. The yellow arrows represent *action edges*. The purple arrows represent *block-place edges*, and the blue arrow represents a *block-pick edge* (Color figure online)

Algorithm 1 AddObject(M, $\mathbf{C}$)

input: an object M; the collaborative manipulation task graph built so far, denoted as $\mathbf{C}$.
if $M \in \mathbf{C}.object\_nodes$ **then**
  return
$\mathbf{C}.object\_nodes.add(M)$
if M is named in goal specification $\mathcal{G}$ **then**
  Re_M = GetGoalRegion(M)
else
  Re_M = GetCurrentRegion(M)
for $R^{pick} \in \mathbf{R}$ **do**
  for $g_{M,R^{pick}} \in \mathbf{Gr}_{M,R^{pick}}$ **do**
    $\bar{\mathbf{a}} = \{\}$
    if ReachablePick(M, $g_{M,R^{pick}}$, R^{pick}) **then**
      if ReachablePlace(M, Re_M, $g_{M,R^{pick}}$, R^{pick}) **then**
        $\bar{\mathbf{a}}.add((M, Re_M, R^{pick}, R^{pick}, g_{M,R^{pick}}, g_{M,R^{pick}}))$
      if M is named in goal specification $\mathcal{G}$ **then**
        for $R^{place} \in \mathbf{R} \setminus \{R^{pick}\}$ **do**
          for $g_{M,R^{place}} \in \mathbf{Gr}_{M,R^{place}}$ **do**
            if EnableGoalHandover(M, $g_{M,R^{pick}}$, $g_{M,R^{place}}$, R^{pick}, R^{place}) and
              ReachablePlace(M, Re_M, $g_{M,R^{place}}$, R^{place}) **then**
              $\bar{\mathbf{a}}$.add($(M, Re_M, R^{pick}, R^{place}, g_{M,R^{pick}}, g_{M,R^{place}})$)
      for $\bar{a} \in \bar{\mathbf{a}}$ **do**
        $R^{pick}_{\bar{a}}$ is the robot to pick M in $\bar{a}$
        $g^{pick}_{\bar{a}}$ is the grasp used by $R^{pick}_{\bar{a}}$ in $\bar{a}$
        $R^{place}_{\bar{a}}$ is the robot to place M in $\bar{a}$
        $g^{place}_{\bar{a}}$ is the grasp used by $R^{place}_{\bar{a}}$ in $\bar{a}$
        $\mathbf{C}.action\_nodes.add(\bar{a})$
        $\mathbf{C}.action\_edges.add(M, \bar{a})$
        for $M_j \in \mathbf{M}$ **do**
          if OccludesPick(M_j, M, $g^{pick}_{\bar{a}}$, $R^{pick}_{\bar{a}}$) **then**
            AddObject(M_j, $\mathbf{C}$)
            $\mathbf{C}.block\_pick\_edges.add(\bar{a}, M_j)$
        if M is named in goal specification $\mathcal{G}$ **then**
          for $M_j \in \mathbf{M}$ **do**
            if OccludesGoalPlace(M_j, M, Re_M, $g^{place}_{\bar{a}}$, $R^{place}_{\bar{a}}$) **then**
              AddObject(M_j, $\mathbf{C}$)
              $\mathbf{C}.block\_place\_edges.add(\bar{a}, M_j)$

For simplicity of presentation, we slightly abuse $\mathbf{M}$ again to denote the objects in $\mathbf{C}$. We use $\mathbf{M}^* \subseteq \mathbf{M}$ to denote the objects that are intended to be moved. We slightly abuse $\bar{\mathbf{a}}$ to denote the set of partially grounded pick-and-place actions in

$\mathbf{C}$. We use $E_{\bar{\mathbf{a}}} = \{(M, \bar{a})\}$ to denote the set of action edges in $\mathbf{C}$. We use $E_B^{pick} = \{(\bar{a}, M)\}$ to denote the set of block-pick edges and $E_B^{place} = \{(\bar{a}, M)\}$ to denote the set of block-place edges in $\mathbf{C}$, $E_B = E_B^{pick} \cup E_B^{place}$, where $M \in \mathbf{M}$ and $\bar{a} \in \bar{\mathbf{a}}$. We define the binary variables $X^t_{M,\bar{a}}$ and $X^t_{\bar{a},M}$, where $t \in [1, \ldots, T]$, $(M, \bar{a}) \in E_{\bar{\mathbf{a}}}$ and $(\bar{a}, M) \in E_B$. $X^t_{M,\bar{a}} = 1$ implies that action $\bar{a}$ is executed at time step t' s.t. $t' \geq t$. $X^t_{\bar{a},M} = 1$ implies that object M can be considered for being moved at time step t since it blocks action $\bar{a}$ which is executed at or after time step t.

Our MIP model is shown in the following. The implications in constraint (11) and constraint (12) are compiled to linear constraints using the big-M method (Griva et al. 2009):

$$\text{minimize} \sum_{(M,\bar{a}) \in E_{\bar{a}}} X^1_{M,\bar{a}} \tag{1}$$

$$X^t_{M,\bar{a}} \geq X^{t+1}_{M,\bar{a}}, \forall (M, \bar{a}) \in E_{\bar{a}}, t \in [1, T-1]$$

$$X^t_{M,\bar{a}} = X^t_{\bar{a},M'}, \forall (M, \bar{a}) \in E_{\bar{\mathbf{a}}}, (\bar{a}, M') \in E_B, t \in [1, T] \tag{2}$$

$$X^t_{M,\bar{a}'} \leq \sum_{(\bar{a},M) \in E_B} X^t_{\bar{a},M}, \forall M \in \mathbf{M} \setminus \mathbf{M}^*, (M, \bar{a}') \in E_{\bar{\mathbf{a}}}, t \in [1, T] \tag{3}$$

$$\sum_{(M,\bar{a}) \in E_{\bar{\mathbf{a}}} \text{ s.t. } R \text{ in } \bar{a}} X^T_{M,\bar{a}} \leq 1, \forall R \in \mathbf{R} \tag{4}$$

$$\sum_{(M,\bar{a}) \in E_{\bar{\mathbf{a}}}} X^T_{M,\bar{a}} \geq 1 \tag{5}$$

$$\sum_{(M,\bar{a}) \in E_{\bar{\mathbf{a}}} \text{ s.t. } R \text{ in } \bar{a}} X^t_{M,\bar{a}} \leq 1 + \sum_{(M,\bar{a}) \in E_{\bar{\mathbf{a}}} \text{ s.t. } R \text{ in } \bar{a}} X^{t+1}_{M,\bar{a}}, \forall R \in \mathbf{R}, t \in [1, T-1] \tag{6}$$

$$\sum_{(M,\bar{a}) \in E_{\bar{\mathbf{a}}}} X^t_{M,\bar{a}} \geq 1 + \sum_{(M,\bar{a}) \in E_{\bar{\mathbf{a}}}} X^{t+1}_{M,\bar{a}}, t \in [1, T-1] \tag{7}$$

$$\sum_{(M,\bar{a}) \in E_{\bar{\mathbf{a}}}} X^1_{M,\bar{a}} = 1, \forall M \in \mathbf{M}^* \tag{8}$$

$$\sum_{(M,\bar{a}') \in E_{\bar{\mathbf{a}}}} X^1_{M,\bar{a}'} \geq X^1_{\bar{a},M}, \forall (\bar{a}, M) \in E_B \tag{9}$$

$$\sum_{(M,\bar{a}) \in E_{\bar{\mathbf{a}}}} X^1_{M,\bar{a}} \leq 1, \forall M \in \mathbf{M} \tag{10}$$

$$X^1_{\bar{a},M} = 1 \implies \sum_{t \in [1,\ldots,T]} X^t_{\bar{a},M} \geq (\sum_{(M,\bar{a}') \in E_{\bar{\mathbf{a}}}} \sum_{t \in [1,\ldots,T]} X^t_{M,\bar{a}'}) + 1, \forall (\bar{a}, M) \in E_B^{pick} \tag{11}$$

$$X^1_{\bar{a},M} = 1 \implies \sum_{t \in [1,\ldots,T]} X^t_{\bar{a},M} \geq (\sum_{(M,\bar{a}') \in E_{\bar{\mathbf{a}}}} \sum_{t \in [1,\ldots,T]} X^t_{M,\bar{a}'}), \forall (\bar{a}, M) \in E_B^{place} \tag{12}$$

Constraint (1) enforces that $X^t_{M,\bar{a}}$ indicates whether we have selected $\bar{a}$ at or after time step t. Constraint (2) enforces that, if an action is selected, then the objects that obstruct it are also moved. Constraint (3) enforces that, besides the objects in $\mathbf{M}^*$, we only move objects that obstruct the actions we have selected. Constraints $(4-7)$ enforce that, at each time step, we select at least one action, while each robot executes at most one action. Constraint (8) enforces that the objects in $\mathbf{M}^*$ are moved. Constraint (9) enforces that all obstacles for the selected actions are moved, while constraint (10) enforces that each object is moved only once. Constraint (11) enforces that each object is moved after the obstacles for its pick action have been moved. Constraint (12) enforces that each object is moved after the obstacles for its place action have been moved. The objective function represents the number of moved objects.

From a MIP solution, we construct a task skeleton which is grounded later. Moreover, we want to construct multiple task skeletons since some task skeletons may be impossible to ground. Every time we obtain a solution, we add a constraint to the MIP model to enforce that we find a different solution from the existing ones until we collect enough task skeletons (Danna et al. 2007). In our implementation, the maximum number of task skeletons is a hyperparameter that varies for different problem instances.

4.2.2 Key component 2: task-skeleton grounding

The second key component in the search phase (Sec. 4.2) is to ground the task skeletons, i.e., to find the object placements and motion trajectories for the partially grounded pick-and-place actions. We use a reverse search algorithm inspired by Stilman et al. (2007) since forward search for continuous parameters of long-horizon task skeletons without any guidance is very challenging (Kim et al. 2019). The insight behind the reverse search strategy is to use the grounded future joint actions as the artificial constraints to guide the grounding for the present time step.

The input to this component is a task skeleton $\bar{\mathbf{S}}$ of T time steps and a sequence $\mathbf{S}_{fut}$ of future grounded joint actions. We use $\mathbf{S}_{fut}$ as artificial constraints to guide the grounding for the current actions, so that we can efficiently find geometrically feasible long-horizon plans (Stilman et al. 2007). Ideally, if we manage to ground task skeleton $\bar{\mathbf{S}}$ successfully, we will get a fully executable task-and-motion plan. However, in many situations, since we cannot account for all geometric specifics during task-skeleton generation, we can only ground the task skeleton partially. In such cases, we will get a set of objects, denoted as $\mathbf{M}^*$, for which we have to generate new task skeletons to relocate. We will then return the sequence of grounded joint actions together with objects $\mathbf{M}^*$. Furthermore, in certain situations, the grounding may totally fail. In such cases, we will simply return a failure flag.

We denote the volume of work space occupied by grounded joint actions $\mathbf{S}_{fut}$ as V_{fut}. We denote the set of

Algorithm 2 TASK- SKELETON GROUNDING($\bar{\mathbf{S}}, \mathbf{S}_{fut}, \mathbf{M}_{fut}, \mathbf{V}_{fut}, \mathbf{M}_{out}$)

1: **input**: a task skeleton $\bar{\mathbf{S}}$; a sequence of grounded joint actions $\mathbf{S}_{fut}$; the set of objects that are planned to be moved in $\mathbf{S}_{fut}$, denoted as $\mathbf{M}_{fut}$; the volume of work space that is occupied by $\mathbf{S}_{fut}$, denoted as V_{fut}; the set of movable objects that are not planned to be moved in $\mathbf{S}_{fut}$ and $\bar{\mathbf{S}}$, denoted as $\mathbf{M}_{out}$.
2: **result:** three possible returns: (i) a sequence of grounded joint actions $\mathbf{S}^*$, indicating that we successfully find an executable task-and-motion plan; (ii) a sequence of grounded joint actions $\mathbf{S}'$ and a set of objects $\mathbf{M}^*$, indicating that we can only partially ground task skeleton $\bar{\mathbf{S}}$ and we have to relocate objects $\mathbf{M}^*$; (iii) a failure flag.
3: **notation:** We denote the sequence concatenating operation as $\oplus$.
4: $\mathcal{G}$ = goal specification of the MR-GTAMP problem instance
5: $\mathbf{M}$ = the set of movable objects of the MR-GTAMP problem instance
6: **for** $t \in [T, \ldots, 1]$ **do**
7: $\quad\bar{\mathbf{S}}[t]$ = PARTIALLYGROUNDEDJOINTACTIONAT($\bar{\mathbf{S}}, t$)
8: $\quad\mathbf{M}^t, \mathbf{R}^t$ = OBJECTSANDROBOTSTOMOVE($\bar{\mathbf{S}}[t]$)
9: $\quad\mathbf{P}$ = FINDPLACEMENTS($\mathbf{M}^t, \mathbf{M}_{out} \cup \mathbf{M}_{fut} \cup \mathbf{F} \cup V_{fut}, \bar{\mathbf{S}}[t]$)
10: $\quad$**if** $\mathbf{P}$ is None **then**
11: $\quad\quad\mathbf{P}$ = FINDPLACEMENTS($\mathbf{M}^t, \mathbf{M}_{fut} \cup \mathbf{F} \cup V_{fut}, \bar{\mathbf{S}}[t]$)
12: $\quad\quad$**if** $\mathbf{P}$ is None **then**
13: $\quad\quad\quad$**return** failure flag
14: $\quad\quad\Xi$ = FINDTRAJECTORIES($\mathbf{M}^t, \mathbf{R}^t, \mathbf{P}, \mathbf{M}_{fut} \cup \mathbf{F}, \bar{\mathbf{S}}[t]$)
15: $\quad\quad$**if** Ξ is None **then**
16: $\quad\quad\quad$**return** failure flag
17: $\quad\quad\mathbf{S}_{fut}$ = CREATEGROUNDEDJOINTACTION($\bar{\mathbf{S}}[t], \Xi, \mathbf{P}$) $\oplus \mathbf{S}_{fut}$
18: $\quad\quad\mathbf{M}^*$ = HAVENOTBEENMOVED($\mathcal{G}, \mathbf{S}_{fut}$) $\cup$ MOVABLESOCCLUDE($\mathbf{M}, \mathbf{S}_{fut}$)
19: $\quad\quad\mathbf{S}' = \mathbf{S}_{fut}$
20: $\quad\quad$**return** $\mathbf{S}', \mathbf{M}^*$
21: $\quad\Xi$ = FINDTRAJECTORIES($\mathbf{M}^t, \mathbf{R}^t, \mathbf{P}, \mathbf{M}_{out} \cup \mathbf{M}_{fut} \cup \mathbf{F}, \bar{\mathbf{S}}[t]$)
22: $\quad$**if** Ξ is None **then**
23: $\quad\quad\Xi$ = FINDTRAJECTORIES($\mathbf{M}^t, \mathbf{R}^t, \mathbf{P}, \mathbf{M}_{fut} \cup \mathbf{F}, \bar{\mathbf{S}}[t]$)
24: $\quad\quad$**if** Ξ is None **then**
25: $\quad\quad\quad$**return** failure flag
26: $\quad\quad\mathbf{S}_{fut}$ = CREATEGROUNDEDJOINTACTION($\bar{\mathbf{S}}, t, \Xi, \mathbf{P}$) $\oplus \mathbf{S}_{fut}$
27: $\quad\quad\mathbf{M}^*$ = HAVENOTBEENMOVED($\mathcal{G}, \mathbf{S}_{fut}$) $\cup$ MOVABLESOCCLUDE($\mathbf{M}, \mathbf{S}_{fut}$)
28: $\quad\quad\mathbf{S}' = \mathbf{S}_{fut}$
29: $\quad\quad$**return** $\mathbf{S}', \mathbf{M}^*$
30: $\quad\mathbf{M}_{fut} = \mathbf{M}_{fut} \cup \mathbf{M}^t$
31: $\quad\mathbf{S}_{fut}$ = CREATEGROUNDEDJOINTACTION($\bar{\mathbf{S}}, t, \Xi, \mathbf{P}$) $\oplus \mathbf{S}_{fut}$
32: $\quad V_{fut} = V_{fut}$.append(SWEPTVOLUME($\Xi, \mathbf{M}^t, \mathbf{R}^t$))
33: $\mathbf{S}^* = \mathbf{S}_{fut}$
34: **return** $\mathbf{S}^*$

movable objects that will be moved by grounded joint actions $\mathbf{S}_{fut}$ as $\mathbf{M}_{fut}$. We denote the set of movable objects that will not be moved by task skeleton $\bar{\mathbf{S}}$ and grounded joint actions $\mathbf{S}_{fut}$ as $\mathbf{M}_{out}$. For time step $t \in [1, \ldots, T]$, we denote the set of objects that are planned to be moved as $\mathbf{M}^t$ and the set of robots that are planned to move them as $\mathbf{R}^t$. Recall that we denote the goal specification and the set of movable objects as $\mathcal{G}$ and $\mathbf{M}$, respectively.

The detailed grounding algorithm is as follows (Alg. 2). The grounding starts at the last time step T. For time step t, we first sample placements for objects $\mathbf{M}^t$ that are collision-free with respect to objects $\mathbf{M}_{out} \cup \mathbf{M}_{fut}$ at their initial poses, fixed objects $\mathbf{F}$ and volume V_{fut} (Alg. 2, line 9). The sampled placements should not collide with volume V_{fut}, because, otherwise, they will prevent the execution of future grounded joint actions that occupy V_{fut}.

Given the placements, we plan pick trajectories, place trajectories and handover trajectories for objects $\mathbf{M}^t$ and robots $\mathbf{R}^t$ that are collision-free with respect to objects $\mathbf{F} \cup \mathbf{M}_{fut} \cup \mathbf{M}_{out}$ at their initial poses (Alg. 2, line 21). We note that, in addition to the fixed objects $\mathbf{F}$ and the objects $\mathbf{M}_{out}$, the planned trajectories should not collide with the objects $\mathbf{M}_{fut}$ that are moved in future grounded joint actions.

Since we may move multiple robots and objects concurrently, we do not allow collisions between the robots, collisions between the moved objects and collisions between a robot and a moved object that is not intended to be manipulated by that robot.

If we succeed in grounding the joint action at time step t, then we expand volume V_{fut} with the volume occupied by the newly planned robot and object trajectories, expand the set $\mathbf{M}_{fut}$ with the moved objects $\mathbf{M}^t$ and expand the grounded joint actions $\mathbf{S}_{fut}$ with the newly grounded joint action (Alg. 2, line 30-32). We then start to ground the joint action at time step $t-1$. If we succeed in grounding the joint actions at every time step, we return an executable task-and-

motion plan $\mathbf{S}^* = \mathbf{S}_{fut}$. However, if we fail at grounding the joint action at time step t, we relax the collision constraints by allowing the sampled placements and trajectories to collide with the objects $\mathbf{M}_{out}$ since we can generate new skeletons to move them later (Alg. 2, line 10-20 and line 22-29). If we succeed after relaxing the constraints, then we terminate the grounding and return the sequence of the grounded joint actions $\mathbf{S}' = \mathbf{S}_{fut}$ and a set of objects $\mathbf{M}^*$. The set of objects $\mathbf{M}^*$ consists of the objects that are named in the goal specification $\mathcal{G}$ but have not yet been moved and the movable objects in the environment that occlude the grounded joint actions $\mathbf{S}'$ (Alg. 2, line 18 and line 27). During the search process (Sec. 4.2), the returned $\mathbf{S}'$ and $\mathbf{M}^*$ are then used as input to the first key component (Sec. 4.2.1) to generate new task skeletons. If, after relaxing the collision constraints, we still cannot find feasible placements and paths, then we simply return failure.

5 Experiments

We empirically evaluate our framework in two challenging domains and show that it can generate effective collaborative task-and-motion plans more efficiently than two baselines.

5.1 Baselines

We compare our framework with two state-of-the-art TAMP frameworks. We provide both baseline planners with information about the reachable regions of each robot.

Ap1 is a multi-robot extension of the RSC algorithm (Stilman et al. 2007) by assuming that the robots form a single composite robot. The action space includes all possible combinations of the single-robot actions and collaboration actions. Unlike our framework, which eliminates infeasible task plans using computed information about the manipulation capabilities of individual robots (Sec. 4.1), thereby pruning the search space, Ap1 would require searching through a large space of all possible combinations of multi-agent actions. Moreover, the focus of Ap1 is on feasibility of the task-and-motion plans, rather than on the plan length and number of objects moved. In contrast, our framework uses the intermediate grounding results (Sec. 4.2) to guide the search towards more effective task-and-motion plans, considering the resulting plan length and the number of objects moved.

Ap2 is a general MR-TAMP framework (Pan et al. 2021) that is efficient in searching for promising task plans based on the constraints incurred during motion planning. We implement the planner in a way such that geometric constraints can be utilized efficiently, e.g., the planner can identify that it needs to move the blocking objects away before it can manipulate the blocked objects. Unlike our framework, which guides the search for feasible positions for object relocation using sampled future actions (Sec. 4.2.2), Ap2 does not include guidance for finding feasible positions for object relocation, which can facilitate finding feasible plans in confined settings.

5.2 Benchmark domains

We evaluate the efficiency and effectiveness of our method and the two baselines in the **packaging** domain shown in Fig. 1 (left) and the **box-moving** domain shown in Fig. 1 (right).

Packaging (PA): In this domain, each problem instance includes 2 to 6 robots, 3 to 5 goal objects, 2 to 13 movable objects besides the goal objects, 1 start region and 3 goal regions. As in Kim et al. (2022), we omit motion planning and simply check for collisions at the picking and placing configurations computed by inverse kinematics solvers in this domain, because collisions in this domain mainly constrain the space of feasible picking and placing configurations. We use Kinova Gen2 lightweight robotic arms. For each benchmark problem instance, we conduct 20 trials with a timeout of 1, 200 seconds. For all methods, we also count a trial as failed, if all possible task plans have been tried.

Box-moving (BO): In this domain, we evaluate our framework for mobile manipulating tasks where the robots have to move target objects from one room to the other room (Fig. 1 (right)). We use simulated PR2 robots. In this domain, each problem instance includes 2 robots, 2 goal objects, 6 movable objects besides the goal objects, 1 start region and 1 goal region. For simplification, we do not consider handover actions. For each benchmark problem instance, we conduct 20 trials with a timeout of 1, 200 seconds. For both methods, we also count a trial as failed, if all possible task plans have been tried.

We use bidirectional rapidly-exploring random trees (LaValle 2006) for motion planning and IKFast (Diankov 2010) for inverse kinematics solving. All methods share the same grasp sets, the same sets of single-robot actions and the same sets of collaboration actions. All experiments were run on an AMD Ryzen Threadripper PRO 3995WX Processor with a memory of 64GB.

5.3 Results

We refer to the number of time steps as *makespan* and the number of moved objects as *motion cost*.

Planning time and success rate. Table 1 shows that our method outperforms both baseline methods on all problem instances with different numbers of goal objects and movable objects with respect to both the planning times and success rates. Ap1 and our method achieved higher success rates on all problem instances than Ap2 because the reverse search strategy (Sec. 4.2.2) utilized in Ap1 and our method can find

Table 1 Comparison of the proposed method with two baseline methods in the two benchmark domains regarding the success rate, planning time, makespan and motion cost

Problem instance	Success rate %			Planning time (s)			Makespan			Motion cost		
	Ap1	Ap2	Ours	Ap1	Ap2	Ours	Ap1	Ap2	Ours	Ap1	Ap2	Ours
PA5	**100.0**	80.0	**100.0**	5.6 (±1.3)	6.1 (±2.1)	**2.4 (±0.2)**	3.0 (±0.2)	2.9 (±0.2)	**2.8 (±0.2)**	3.8 (±0.2)	**3.6 (±0.2)**	**3.6 (±0.2)**
PA7	80.0	70.0	**100.0**	39.8 (±12.8)	10.5 (±2.9)	**4.0 (±0.9)**	3.7 (±0.3)	**3.0 (±0.3)**	3.1 (±0.2)	4.8 (±0.3)	4.3 (±0.2)	**4.1 (±0.2)**
PA10	55.0	40.0	**90.0**	129.2 (±58.2)	N/A	**19.6 (±6.1)**	4.6 (±0.6)	N/A	**4.2 (±0.3)**	5.6 (±0.6)	N/A	**5.2 (±0.4)**
BO8	85.0	35.0	**100.0**	246.5 (±54.2)	N/A	**182.2 (±48.3)**	4.8 (±0.2)	N/A	**3.4 (±0.3)**	7.6 (±0.1)	N/A	**5.0 (±0.6)**

The numbers in the names of the problem instances indicate the numbers of the goal objects and the movable objects besides the goal objects. In PA5, PA7 and PA10, each problem instance has 3 goal objects and 2 robots. We omit the planning time and solution quality results for Ap2 on PA10 and BO8 because its success rate is significantly lower than those of the other two methods

feasible object placements much more efficiently than the forward search strategy used in Ap2. Moreover, Ap2 can generate task plans that include irrelevant objects while Ap1 and our method focus on manipulating the important objects, like blocking objects for necessary manipulation or goal objects. Our method achieved higher success rates with shorter planning times than Ap1 on the difficult problem instances PA7, PA10 and BO8 because our method first generates promising task skeletons (Sec. 4.2.1) that use the information about the collaborative manipulation capabilities of the individual robots to prune the task plan search space, which can be extremely large when there are many objects and multiple robots (Pan et al. 2021). The main cause of failure of our method is running out of task skeletons which can be addressed by incrementally adding more task skeletons during the search process.

Solution quality. Table 1 shows that our method can generate effective task-and-motion plans with respect to the motion cost and the makespan. Our method first generates task skeletons with short makespans by incrementally increasing time step limit and with low motion costs by incorporating the motion cost into the objective function of the MIP formulation (Sec. 4.2.1). On the other hand, our MCTS exploration strategy motivates the planner to search for effective plans with small numbers of moved objects. It should be noted that, although Ap2 generated plans with shorter makespans for PA7, it has lower success rates and longer planning times than our method. Also, Ap1 generated plans that move significantly more objects for PA7, PA10 and BO8 than our method because it uses a depth-first search strategy for finding feasible plans (Stilman et al. 2007).

Scalability evaluation. We evaluated the scalability of our method in the PA domain with 18 movable objects, including 5 goal objects and 2 to 6 robots. Table 2 shows that our method can solve these large problem instances. For problem instances with 3 and more robots, it achieved higher success rates compared to the problem instances with 2 robots. Moreover, our method can achieve shorter makespans and lower motion costs when more robots are involved. These results show that our method can effectively utilize multiple robots to address challenging planning problem instances and generate intelligent collaboration strategies for multiple robots.

However, in our experiment, the success rates for problem instances with 5 and 6 robots are lower than the success rates for problem instances with 3 and 4 robots. The required planning time also increases when more robots are added, starting from the problem instances with 3 robots. This is because adding more robots into the system will lead to more cluttered environments and more difficult collision avoidance between robots. In future work we will explore potentially mitigating this issue by carefully designing the layout of robots (Tay and Ngoi 1996).

Table 2 The results of the proposed method in domain PA regarding the success rate, planning time, makespan and motion cost

Problem instance	Success rate %	Planning time (s)	Makespan	Motion cost
2 robots	60.0	148.4 (±36.8)	6.1 (±0.4)	8.9 (±0.4)
3 robots	80.0	99.0 (±48.6)	4.9 (±0.3)	8.2 (±0.5)
4 robots	85.0	109.1 (±33.6)	4.7 (±0.3)	8.2 (±0.4)
5 robots	75.0	207.0 (±48.7)	4.1 (± 0.2)	8.0 (± 0.3)
6 robots	70.0	362.7 (± 64.6)	3.4 (± 0.2)	7.7 (± 0.4)

The numbers in the names of the problem instances indicate the numbers of the robots

Fig. 5 A human operator is installing a bolt into the roof bolter (https://bit.ly/3tfYOMY)

6 Application study: roof bolting

Roof bolting is an essential operation within the underground mining cycle, as it aims to provide support to the exposed roof and ribs of the new excavation (Peng and Tang 1984; Mark 2002) (Fig. 5). [2] The roof bolting operation follows immediately after the extraction task and reinforces the roof to provide a safe working environment. Roof bolting is utilized in almost all coal mining operations around the world.

The roof bolt binds the unstable roof together, preventing movement in a rock mass. There are several types of bolt installation techniques, depending on the mechanics of the bolt and the rock. This application study focuses on a technique where installation of the bolts is done by drilling a hole in the roof, inserting the resin and inserting the bolt. The roof bolting operation is a labor-intensive task that requires the operators of the machine to install and replace detachable drill steels and cutting bits, holding and positioning of resin cartridges and 1.2 to 3 m (4 to 10 foot) long bolts in a pattern that can be half a meter square. During the roof bolts installation process, the operators are at risk from working in the proximity of potentially unsupported roof, loose bolts, hydraulic-powered equipment, gas and heavy tools in awkward conditions. Apart from these safety risks, the operators are also vulnerable to inhalation of dust and noise from drilling and bolting processes which can be traced to the several pumps from the roof bolter machinery (Jiang and Luo 2021). The operation of these machines requires attention to the risks, which, combined with fatigue, leads to accidents, injuries and severe injuries including fatalities. Therefore, more and more research efforts have been put into developing robot systems that are capable of carrying out the sequence of roof-bolting operations to achieve a high-impact health and safety intervention for roof-bolter operators (Van Duin et al. 2013; Schafrik et al. 2022).

The bolting machines have been automated before, but these modifications are not popular with the community because autonomous machines are highly restricted in their usage. They are setups for a single-purpose drilling and bolting operation, where in most mining and civil construction, flexible installation is desired.

Figure 6 shows a robot-assisted roof-bolting system constructed in our lab (Schafrik et al. 2022). In a roof-bolting task, the system does following actions step-by-step: (i) drill a hole in the roof with a drill steel; (ii) remove the drill steel; (iii) install resin; (iv) install a bolt. To perform these actions successfully, the roof-bolter operator and the roof bolter need to collaborate seamlessly. The role of the roof bolter is to drill the roof and install the resin and the bolt into the roof. The role of the operator is to pick up the drill steel, the resin and the bolt and hand them over to the roof bolter. In our robot-assisted roof-bolting system, we replace the human roof-bolter opera-

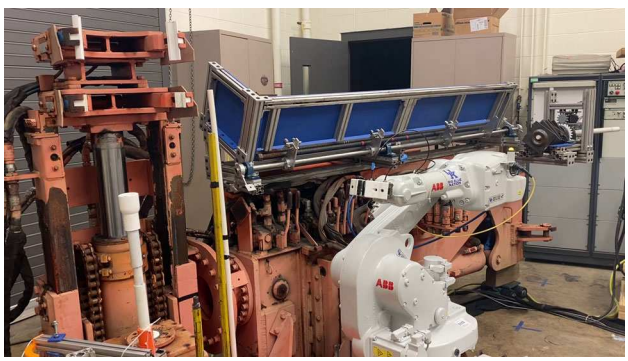


Fig. 6 The roof bolting system

[2] https://bit.ly/3tfYOMY

tor with an ABB IRB 1600 robot because of its high accuracy and flexibility.

Industrial robots have been widely deployed in factories (Nikolaidis and Shah 2012) in isolation from people, where their tasks can be pre-defined in the form of waypoints. However, underground mine is usually cluttered and dynamic. For example, human workers who are focused on other tasks may leave tools around unconsciously. The left tools and other objects in the environment will become obstacles blocking the roof-bolting operation. The robot arm then has to clear its operation space, i.e., move movable obstacles out of the way. Moreover, to perform roof-bolting tasks, it is critical to coordinate the roof bolter and robot arm because of their different capabilities. On one hand, we need the roof bolter to drill holes in the roof and install the bolts into the roof; on the other hand, we need the robot arm to hand bolts, resins and drill steels over to the roof bolter and rearrange movable obstacles. To automatically generate manipulation plans to coordinate the roof bolter and the robot arm, the planning framework should first compute the occlusion and reachability information for the roof bolter and the robot arm (Sec. 4.1) and then generate effective manipulation plans accordingly.

We observe that in each step of the roof-bolting operation we have a target object whose target configuration is specified and numerous objects that can be treated as movable obstacles. By treating the roof bolter as the second robot, we propose to formulate each step of the roof-bolting operation as a multi-robot geometric task-and-motion planning (MR-GTAMP) problem.

6.1 Formulating roof-bolting operation as MR-GTAMP problems

In the roof-bolting task, we need to move the drill steel, the resin and the bolt to their target configurations in the roof. In our application study, we only focus on bolt placement. Other actions can be formulated as MR-GTAMP problems similarly. We assume the target configuration of the bolt has been pre-defined. We formulate an MR-GTAMP problem where we have two robots, i.e., a roof bolter and an ABB IRB 1600 robot arm. These two robots have different reachability: the roof bolter can place the bolt into its target configuration, whereas the robot arm can pick up the bolt from its initial configuration. Moreover, the robot arm can reach most of the movable objects in the environment.

The reachability of the roof bolter and the robot arm can be computed by calling motion planning algorithms (Sec. 4.1) and can be easily encoded using collaborative manipulation task graphs (CMTGs) (Sec. 4.2.1). We will then use our proposed framework to compute executable task-and-motion plans for the roof bolter and the robot arm that account for their different manipulation capabilities.

6.2 Two example scenarios

In our application study, we run our proposed planner for two example scenarios. We show the environment setups in simulation and the built CMTGs (Figs. 7,8,9,10). We denote the ABB robot arm and the roof bolter as R_1 and R_2. For each action, we denote the object that is moved as M_i, the grasp that is used by robot R_k as g_{M_i,R_k} and the region to which the object is moved as Re_j.

Example scenario 1. In the first example scenario, we have the bolt as a target object (object M_1) and three movable obstacles (objects M_2, M_3, M_4) (Fig. 7). The CMTG for moving object M_1 is shown in Fig. 8 (left). The CMTG shows that to move object M_1, the ABB robot arm and the roof bolter have to perform a handover action. Object M_4 blocks the ABB robot arm from picking up object M_1 and object M_3 blocks the ABB robot arm from picking up object M_4. Given the CMTG, we can generate a task skeleton. During grounding (Sec. 4.2.2) the generated task skeleton, the planner finds that object M_2 blocks the handover action between the ABB robot arm and the roof bolter. The planner then generates a new CMTG to move object M_2 (Fig. 8 (right)).

Example scenario 2. In the second example scenario, we have a target object, bolt (object M_1) and two movable obstacles (objects M_2, M_3) (Fig. 9). The CMTG for moving object M_1 is shown in Fig. 10 (left). The CMTG shows that to move

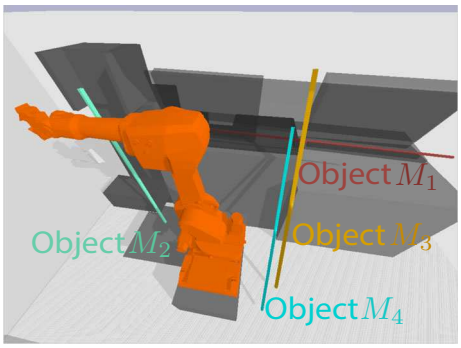


Fig. 7 Example scenario 1 in the simulation

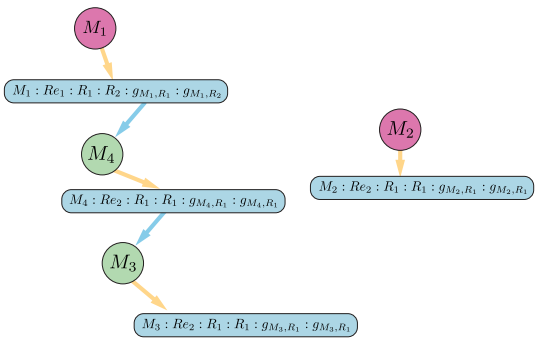


Fig. 8 Generated CMTGs for example scenario 1. R_1 and R_2 represent the ABB robot arm and the roof bolter

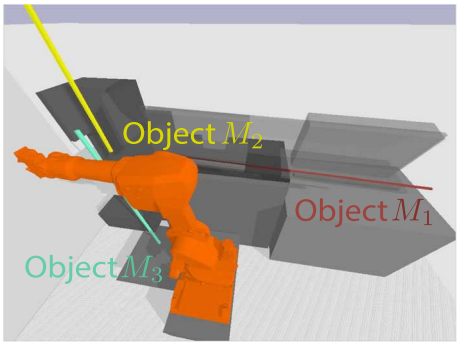


Fig. 9 Example scenario 2 in the simulation

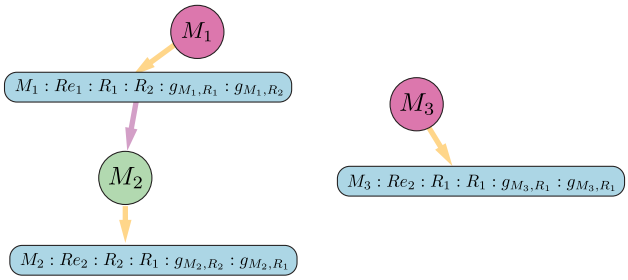


Fig. 10 Generated CMTGs for example scenario 2. R_1 and R_2 represent the ABB robot arm and the roof bolter

object M_1, the ABB robot arm and the roof bolter have to perform a handover action. Object M_2 blocks the roof bolter from placing object M_1 to its target configuration. During grounding (Sec. 4.2.2) the generated task skeleton based on the CMTG, the planner finds that object M_3 blocks the handover action between the ABB robot arm and the roof bolter. The planner then generates a new CMTG to move object M_3 (Fig. 10 (right)).

6.3 Planning and execution details

Planning. We conduct 5 trials on an AMD Ryzen Threadripper PRO 3995WX Processor with a memory of 64GB for each scenario. The average planning time for example scenario 1 and example scenario 2 are 144.1($\pm$21.5) seconds and 100.8($\pm$15.4) seconds. We observe that most of the planning time is spent on task skeleton grounding (Sec. 4.2.2) where motion planning is extensively called. The average planning time spent on motion planning for example scenario 1 and example scenario 2 are 143.4($\pm$21.5) seconds and 100.2($\pm$15.4) seconds. This is because it is challenging to plan collision-free motion trajectories to move large objects such as the bolt and drill steel in a confined workspace. On the other hand, it only takes 0.6($\pm$0.0) seconds and 0.6($\pm$0.0) seconds on average to compute task skeletons (Sec. 4.2.1) for example scenario 1 and example scenario 2.

Execution. In Figs. 11 and 12 we show the execution of the generated plans in simulation and real-world. We include videos of scenarios 1 and 2 in the supplemental material. The execution time of the generated plans for example scenario 1 and example scenario 2 are 250.0 seconds and 270.0 seconds. To execute the planned motion trajectories on the ABB robot, we first manually smooth the motion trajectories by downsampling the waypoints of the motion trajectories. We then automatically generate ABB robot instructions in RAPID (Robotics 2007) from the waypoints. Each waypoint is a robot configuration defined in the ABB robot's joint space and is as an argument passed to MoveJ command in RAPID.

7 Discussion

In this paper, we presented a framework for MR-GTAMP problems by proposing a novel MIP formulation to utilize information about the collaborative manipulation capabilities of the individual robots to generate promising task skeletons for guiding the planning search. We proposed an efficient task-skeleton grounding algorithm inspired by the previous work on MAMO (Stilman et al. 2007). The proposed components are integrated via a Monte-Carlo Tree Search exploration strategy that searches for effective task-and-motion plans. We showed that our framework outperforms two baselines on two challenging MR-GTAMP problems with respect to the planning time and success rates, can generate effective plans with respect to the resulting plan length and the number of objects moved, and can scale up to large problem instances. We also showed that our framework can be applied in the roof-bolting operation for underground mining, where a robotic arm coordinates with an autonomous roof bolter.

Limitations. Our work is limited in many ways. In our work, we consider only *monotone* instances of the MR-GTAMP problem, where each object is moved only once. This assumption limits us from solving problem instances that require moving one object multiple times (Kim and Shimanuki 2020) such as Tower of Hanoi, object swapping tasks. We leave the extension to *non-monotone* problem instances for future work. Our framework also pre-defines handover regions for different robots to compute collaborative manipulation information (Sec. 4.1). This approach may be limited for dynamic environments such as human homes, thus we plan to incorporate a handover region searching process in the task-skeleton grounding component (Sec. 4.2.2) in the future. We have also assumed full observability of the scene and therefore cannot handle uncertainties, noise in robot perception (Muguira-Iturralde et al. 2022). We plan to account for sensing limitations in the future (Nikolaidis et al. 2009, 2016). Currently, our approach aims to generate plans with short plan lengths and small numbers of moved objects. However, we do not consider the length of the resulting motion trajectories and the corresponding robot execution

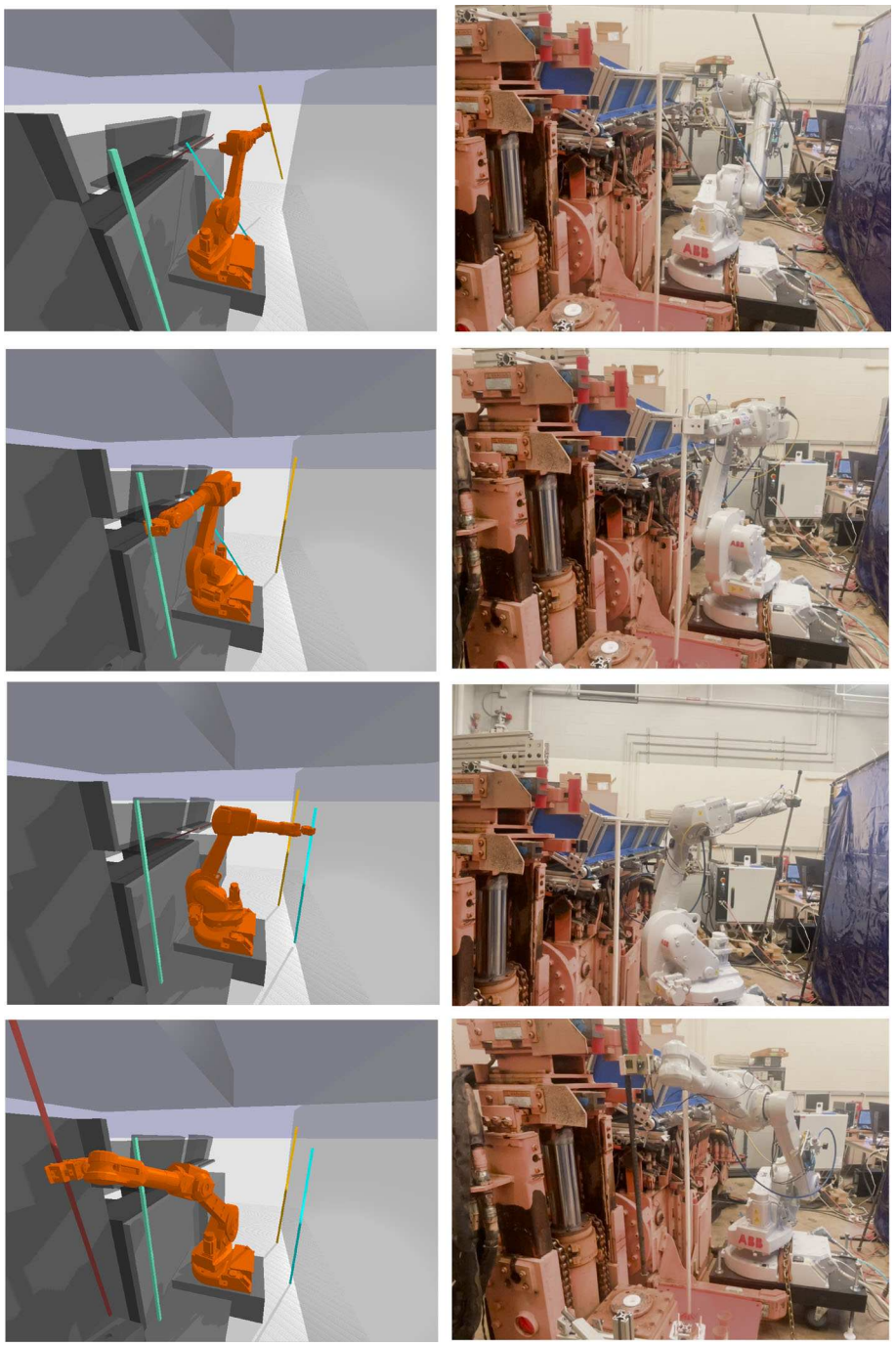


Fig. 11 Frames showing the execution of the generated plan for example scenario 1 in both simulation (Left) and real-world (Right)

time (Chen et al. 2022; N et al. 2023), thus we plan to account for these evaluation metrics in the future.

Future work also includes using learning to improve the planning efficiency (Kim et al. 2022) and extending the developed techniques to more general MR-TAMP problems (Pan et al. 2021) and more diverse environments (Fontaine et al. 2021; Zhang et al. 2020).

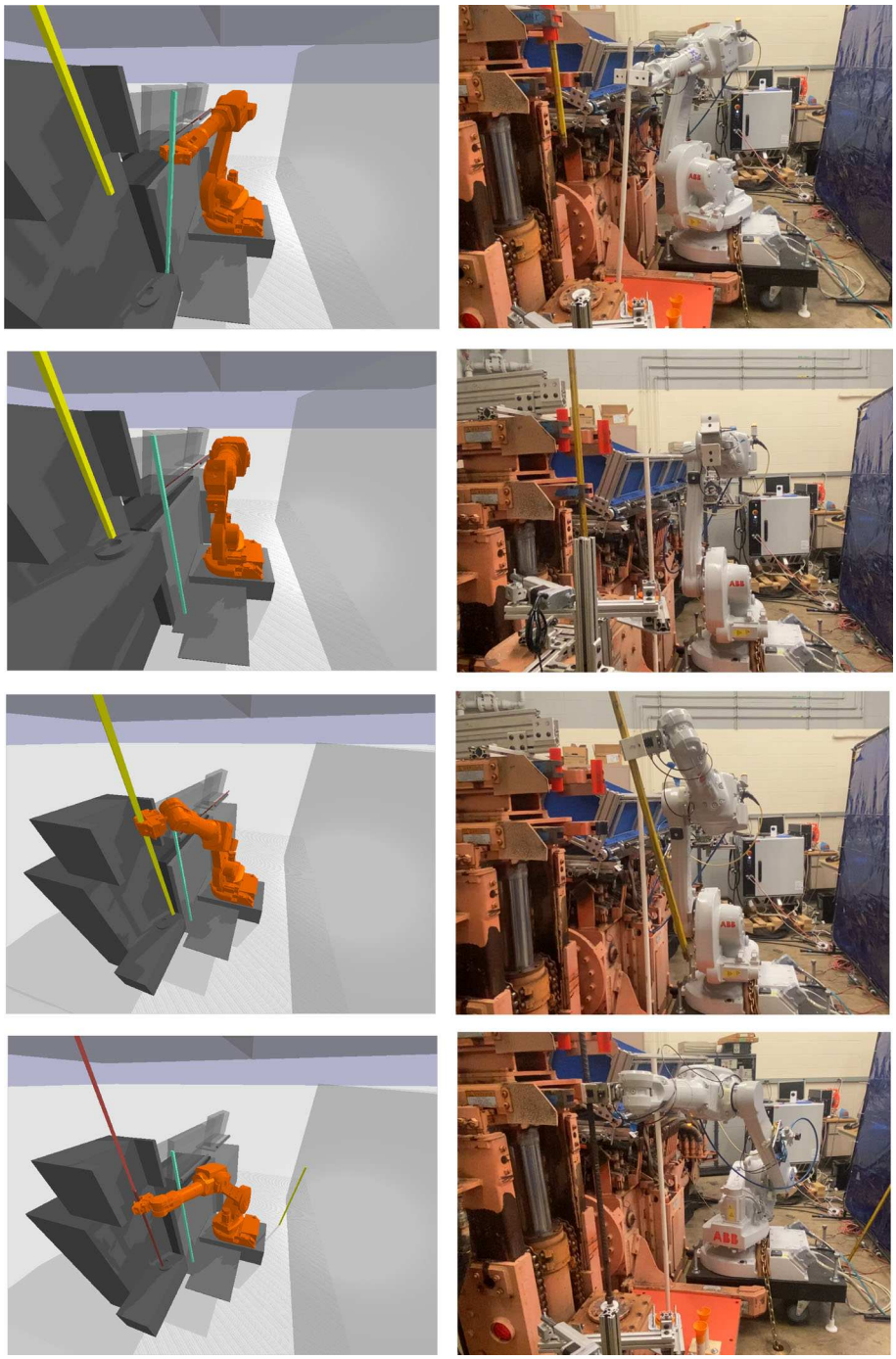


Fig. 12 Frames showing the execution of the generated plan for example scenario 2 in both simulation (Left) and real-world (Right)

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10514-023-10148-y.

Acknowledgements This work was supported by the National Science Foundation NRI # 2024936 and the Alpha Foundation for the Improvement of Mine Safety and Health # AFC820-68.

Author Contributions HZ led the algorithm development, experiments, and paper editing; SC and JL helped with the algorithm development and paper editing; JZ and PK helped with the experiments; SK, ZA, SS and SN supervised the project.

Funding Open access funding provided by SCELC, Statewide California Electronic Library Consortium

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

References

Ahn, J., Kim, C., & Nam, C. (2021). Coordination of two robotic manipulators for object retrieval in clutter. arXiv preprint arXiv:2109.15220

Behrens, J. K., Stepanova, K., & Babuska, R. (2020) Simultaneous task allocation and motion scheduling for complex tasks executed by multiple robots. In *IEEE international conference on robotics and automation* (pp. 11443–11449). https://doi.org/10.1109/ICRA40945.2020.9197103

Bidot, J., Karlsson, L., Lagriffoul, F., & Saffiotti, A. (2017). Geometric backtracking for combined task and motion planning in robotic systems. *Artificial Intelligence, 247*, 229–265.

Chen, J., Li, J., Huang, Y., Garrett, C., Sun, D., Fan, C., Hofmann, A., Mueller, C., Koenig, S., & Williams, B. C. (2022). Cooperative task and motion planning for multi-arm assembly systems. arXiv preprint arXiv:2203.02475 (2022)

Coumans, E., & Bai, Y. (2016–2019). PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org

Cplex, I. I. (2009). V12. 1: User's manual for cplex. *International Business Machines Corporation, 46*(53), 157.

Danielczuk, M., Kurenkov, A., Balakrishna, A., Matl, M., Wang, D., Martín-Martín, R., Garg, A., Savarese, S., & Goldberg, K. Mechanical search: Multi-step retrieval of a target object occluded by clutter. In *IEEE international conference on robotics and automation* (pp. 1614–1621). https://doi.org/10.1109/ICRA.2019.8794143

Danna, E., Fenelon, M., Gu, Z.,& Wunderling, R. (2007) Generating multiple solutions for mixed integer programming problems. In *Integer programming and combinatorial optimization* (pp. 280–294)

Dantam, N. T., Kingston, Z. K., Chaudhuri, S., & Kavraki, L. E. (2018). An incremental constraint-based framework for task and motion planning. *IJRR, 37*(10), 1134–1151.

Diankov, R. (2010) Automated construction of robotic manipulation programs. PhD thesis, Carnegie Mellon University

Fontaine, M., Hsu, Y.-C., Zhang, Y., Tjanaka, B., & Nikolaidis, S. (2021). On the Importance of Environments in Human-Robot Coordination. In *Proceedings of robotics: Science and systems, virtual*. https://doi.org/10.15607/RSS.2021.XVII.038

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2020). Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *ICAPS* (vol. 30, pp. 440–448)

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., & Lozano-Pérez, T. (2021). Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems, 4*(1), 265–293. https://doi.org/10.1146/annurev-control-091420-084139

Griva, I., Nash, S. G., & Sofer, A. (2009). *Linear and nonlinear optimization* (Vol. 108). Philadelphia, Pennsylvania, USA: Siam.

Hartmann, V. N., Orthey, A., Driess, D., Oguz, O. S., & Toussaint, M. (2021). Long-horizon multi-robot rearrangement planning for construction assembly. arXiv preprint arXiv:2106.02489 (2021)

Hauser, K. (2013). Minimum constraint displacement motion planning. In *Robotics: science and systems*. https://doi.org/10.15607/RSS.2013.IX.017

Hoffmann, J. (2001). Ff: The fast-forward planning system. *AI Magazine, 22*, 57–62.

Hun Cheong, S., Cho, B. Y., Lee, J., Kim, C., & Nam, C. (2020). Where to relocate?: Object rearrangement inside cluttered and confined environments for robotic manipulation. In *IEEE international conference on robotics and automation* (pp. 7791–7797). https://doi.org/10.1109/ICRA40945.2020.9197485

Jiang, H., & Luo, Y. (2021). Development of a roof bolter drilling control process to reduce the generation of respirable dust. *International Journal of Coal Science & Technology, 8*(2), 199–204.

Khodeir, M., Agro, B., & Shkurti, F. (2023). Learning to search in task and motion planning with streams. *IEEE Robotics and Automation Letters, 8*(4), 1983–1990. https://doi.org/10.1109/LRA.2023.3242201

Kim, B., & Shimanuki, L. (2020). Learning value functions with relational state representations for guiding task-and-motion planning. In *Conference on robot learning* (vol. 100, pp. 955–968)

Kim, B., Kaelbling, L. P., & Lozano-Pérez, T. (2019). Adversarial actor-critic method for task and motion planning problems using planning experience. In *AAAI conference on artificial intelligence* (vol. 33, pp. 8017–8024). https://doi.org/10.1609/aaai.v33i01.33018017

Kim, B., Shimanuki, L., Kaelbling, L. P., & Lozano-Pérez, T. (2022). Representation, learning, and planning algorithms for geometric task and motion planning. *The International Journal of Robotics Research, 41*(2), 210–231. https://doi.org/10.1177/02783649211038280

King, J. E., Cognetti, M., & Srinivasa, S. S. (2016). Rearrangement planning using object-centric and robot-centric action spaces. In *IEEE international conference on robotics and automation* (pp. 3940–3947). https://doi.org/10.1109/ICRA.2016.7487583

Kogo, T., Takaya, K., & Oyama, H. (2021) Fast milp-based task and motion planning for pick-and-place with hard/soft constraints of collision-free route. In *2021 IEEE SMC* (pp. 1020–1027). IEEE

Krontiris, A., & Bekris, K. E. (2016). Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner. In *ieee international conference on robotics and automation* (pp. 3924–3931). https://doi.org/10.1109/ICRA.2016.7487581

Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A., & Karlsson, L. (2014). Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research, 33*(14), 1726–1747.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge, United Kingdom: Cambridge University Press.

Mansouri, M., Pecora, F., & Schüller, P. (2021). Combining task and motion planning: Challenges and guidelines. *Frontiers in Robotics and AI*. https://doi.org/10.3389/frobt.2021.637888

Mark, C. (2002). The introduction of roof bolting to US underground coal mines (1948-1960): a cautionary tale. In *Proceedgins of the 21st international conference on ground control in mining* (pp. 150–160)

Muguira-Iturralde, J., Curtis, A., Du, Y., Kaelbling, L. P., & Lozano-Pérez, T. (2022). Visibility-aware navigation among movable obstacles. 2023 IEEE ICRA

N., H. V., & Marc, T. (2023). Towards computing low-makespan solutions for multi-arm multi-task planning problems. arXiv preprint arXiv:2305.17527

Nam, C., Lee, J., Hun Cheong, S., Cho, B. Y., & Kim, C. Fast and resilient manipulation planning for target retrieval in clutter. In

IEEE international conference on robotics and automation (pp. 3777–3783). https://doi.org/10.1109/ICRA40945.2020.9196652
Nikolaidis, S., & Shah, J. (2012) Human-robot teaming using shared mental models. In *Proceedings of IEEE/ACM international conference on human-robot interaction, workshop on human-agent-robot teamwork*
Nikolaidis, S., Dragan, A., & Srinivasa, S. (2016). Viewpoint-based legibility optimization. In *ACM/IEEE international conference on human-robot interaction* (pp. 271–278). https://doi.org/10.1109/HRI.2016.7451762
Nikolaidis, S., Ueda, R., Hayashi, A., & Arai, T. (2009). Optimal camera placement considering mobile robot trajectory. In *2008 IEEE international conference on robotics and biomimetics* (pp. 1393–1396). IEEE
Pan, T., Wells, A. M., Shome, R., & Kavraki, L. E. (2021). A general task and motion planning framework for multiple manipulators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3168–3174). https://doi.org/10.1109/IROS51168.2021.9636119
Peng, S. S., & Tang, D. (1984). Roof bolting in underground mining: A state-of-the-art review. *International Journal of Mining Engineering, 2*(1), 1–42.
Quispe, A. H., Amor, H. B., & Christensen, H. I. (2016). Combining arm and hand metrics for sensible grasp selection. In *IEEE international conference on automation science and engineering* (pp. 1170–1176). https://doi.org/10.1109/COASE.2016.7743537
Ren, T., Chalvatzaki, G., & Peters, J. (2021). Extended tree search for robot task and motion planning. arXiv preprint arXiv:2103.05456
Robotics, A. (2007). *Operating manual robotstudio*. Sweden: Västerås.
Rodríguez, C., & Suárez, R. (2016). Combining motion planning and task assignment for a dual-arm system. In *IEEE/RSJ international conference on intelligent robots and systems* (pp. 4238–4243). https://doi.org/10.1109/IROS.2016.7759624
Schafrik, S., Kolapo, P., & Agioutantis, Z. (2022) Development of an automated roof bolting machine for underground coal mines. In *Proceedings of annual SOMP conference*, Namibia
Shome, R., Solovey, K., Yu, J., Bekris, K., & Halperin, D. (2021). Fast, high-quality two-arm rearrangement in synchronous, monotone tabletop setups. *IEEE Transactions on Automation Science and Engineering, 18*(3), 888–901. https://doi.org/10.1109/TASE.2021.3055144
Silver, T., Chitnis, R., Curtis, A., Tenenbaum, J., Lozano-Perez, T., & Kaelbling, L. P. (2020) Planning with learned object importance in large problem instances using graph neural networks. arXiv preprint arXiv:2009.05613
Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature, 550*, 354.
Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., & Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE ICRA* (pp. 639–646). IEEE
Stilman, M., Schamburek, J.-U., Kuffner, J., & Asfour, T. (2007). Manipulation planning among movable obstacles. In *IEEE international conference on robotics and automation* (pp. 3327–3332). https://doi.org/10.1109/ROBOT.2007.363986
Takano, R., Oyama, H., & Yamakita, M. (2021). Continuous optimization-based task and motion planning with signal temporal logic specifications for sequential manipulation. In *2021 IEEE international conference on robotics and automation (ICRA)* (pp. 8409–8415). https://doi.org/10.1109/ICRA48506.2021.9561209
Tay, M., & Ngoi, B. (1996). Optimising robot workcell layout. *The International Journal of Advanced Manufacturing Technology, 12*, 377–385.
Toussaint, M. (2015). Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Proceedings of the 24th international conference on artificial intelligence. IJCAI'15* (pp. 1930–1936). AAAI Press.
Toussaint, M., & Lopes, M. (2017). Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In *2017 IEEE ICRA* (pp. 4044–4051). IEEE
Toussaint, M., & Lopes, M. (2017). Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In *IEEE international conference on robotics and automation* (pp. 4044–4051). https://doi.org/10.1109/ICRA.2017.7989464
Van Duin, S., Meers, L., Donnelly, P., & Oxley, I. (2013). Automated bolting and meshing on a continuous miner for roadway development. *International Journal of Mining Science and Technology, 23*(1), 55–61.
Yang, Z., Garrett, C. R., & Fox, D. (2022). Sequence-based plan feasibility prediction for efficient task and motion planning. arXiv preprint arXiv:2211.01576
Zhang, H., Chan, S.-H., Zhong, J., Li, J., Koenig, S., & Nikolaidis, S. (2022). A mip-based approach for multi-robot geometric task-and-motion planning. In *2022 IEEE 18th international conference on automation science and engineering (CASE)* (pp. 2102–2109). IEEE
Zhang, H., Fontaine, M., Hoover, A., Togelius, J., Dilkina, B., & Nikolaidis, S. (2020). Video game level repair via mixed integer linear programming. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 16*(1), 151–158.

Hejia Zhang received a B.E. in Bioengineering from Zhejiang University in 2017 and a M.S. in Computer Science from University of Southern California in 2019. He is interested in Robotics, Artificial Intelligence, and Human-Robot Interaction.

Shao-Hung Chan received a B.S. in Electrical Engineering from National Cheng Kung University in 2017 and an M.S. in Electrical Engineering from National Taiwan University in 2019. He is interested in Artificial Intelligence, Heuristic Search, and Robotics. He received a USC Annenberg Graduate Fellowship in 2019 and a Ph.D. Sandwich Program Fellowship at Ben-Gurion University of the Negev in 2021.

Jie Zhong received a B.S. in Mechanical Engineering from Xi'an Jiaotong University in 2020 and an M.S. in Mechanical Engineering from University of Southern California in 2023. He is interested in artificial intelligence, mechatronics, and robotics.

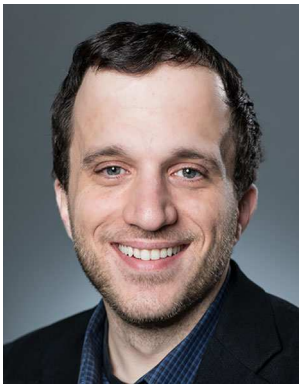
Jiaoyang Li is an assistant professor in the Robotics Institute at Carnegie Mellon University. Her research focuses on multi-robot planning and coordination. She obtained a B.Eng. degree in Automation from Tsinghua University, China in 2017 and a Ph.D. degree in Computer Science from the University of Southern California, USA in 2022.

Peter Kolapo is a passionate and well-trained mining engineer with extensive expertise and enthusiasm for mining automation, rock engineering and geomechanics. Peter Kolapo is a PhD candidate in the Department of Engineering at the University of Kentucky, USA. Peter holds a Bachelor of Mining Engineering from the Federal University of Technology Akure, Nigeria and completed his master's degree in mining engineering at the University of Witwatersrand, where he specialized in rock mechanics and engineering. Peter's current research entails developing an automated roof bolting machine for underground mining and tunneling operations. Peter has experience in testing the accuracy of terrestrial laser scanning technologies, monitoring underground rock mass movement, measuring in-situ stress at a deep-level underground gold mine, shaft pillar stability analysis, coring of rock and laboratory testing of rock samples all in South Africa. Peter has a broad engineering background, covering the automation of mining equipment, ground control technologies, underground stability analysis, numerical modeling, rock drilling efficiency, laboratory testing of rock samples, in-situ stress measurement, closure rate in underground excavation, support design and pillar design.

Sven Koenig is Dean's Professor of Computer Science at the University of Southern California. Most of his research centers around techniques for decision making (planning and learning) that enable single situated agents (such as robots or decision-support systems) and teams of agents to act intelligently in their environments and exhibit goal-directed behavior in real-time, even if they have only incomplete knowledge of their environment, imperfect abilities to manipulate it, limited or noisy perception or insufficient reasoning speed. Sven is a fellow of the Association for the Advancement of Artificial Intelligence (AAAI), the Association for Computing Machinery (ACM), the Institute of Electrical and Electronics Engineers (IEEE), and the American Association for the Advancement of Science (AAAS). Additional information about Sven can be found on his webpages: http://idm-lab.org.

Zach Agioutantis is the Mining Engineering Foundation Professor and Chair of the Department of Mining Engineering at the University of Kentucky. Prior to that, he was a Professor and the Director of the Rock Mechanics Laboratory at the Technical University of Crete in Greece for over 25 years. Over the years, he has taught several mining engineering courses at the undergraduate and graduate levels. He has authored and co-authored more than 80 peer-reviewed journal papers and more than 300 conference papers on subjects related to subsidence, applied and theoretical rock mechanics, soil mechanics, slope stability, computer applications in mining and geotechnical engineering, mining systems and mining systems simulation, as well as sustainability in mining operations.

Steven Schafrik is an Associate Professor in the Department of Mining Engineering at the University of Kentucky. He has extensive research experience in the application of computer systems in mining engineering, mine ventilation, communication and navigation in underground environments, virtual reality systems to train miners, and numerous others. He has led multiple teams on research projects on mine equipment automation, underground mine communications, and dust control and scrubbing systems. Several government agencies have funded his research in addition to private sector entities. He is active in professional societies, including the Society for Mining, Metallurgy, and Exploration (SME) where he holds several administrative positions. He is the advisor to the University of Kentucky mine rescue team.



Stefanos Nikolaidis is an Assistant Professor of Computer Science at the University of Southern California. His research draws upon expertise on artificial intelligence, procedural content generation and quality diversity optimization and leads to end-to-end solutions that enable deployed robotic systems to act robustly when interacting with people in practical, real-world applications. Stefanos completed his PhD at CMU's Robotics Institute and received an MS from MIT, a MEng from the University of Tokyo and a BS from the National Technical University of Athens. Stefanos' research has been recognized with an NSF CAREER award, an oral presentation at NeurIPS and best paper awards and nominations from HRI, IROS, and ISR.