IEEE IROS. Preprint Version. Accepted June, 2021

A General Task and Motion Planning Framework For Multiple Manipulators

Tianyang Pan, Andrew M. Wells, Rahul Shome and Lydia E. Kavraki

Abstract— **Many manipulation tasks combine high-level discrete planning over actions with low-level motion planning over continuous robot motions. Task and motion planning (TMP) provides a powerful general framework to combine discrete and geometric reasoning, and solvers have been previously proposed for single-robot problems. Multi-robot TMP expands the range of TMP problems that can be solved but poses significant challenges when considering scalability and solution quality. We present a general TMP framework designed for multiple robotic manipulators. This is based on two contributions. First, we propose an optimal task planner designed to support simultaneous discrete actions. Second, we introduce an intermediate scheduler layer between task planner and motion planner to evaluate alternate robot assignments to these actions. This aggressively explores the search space and typically reduces the number of expensive task planning calls. Several benchmarks with a rich set of actions for two manipulators are evaluated. We show promising results in scalability and solution quality of our TMP framework with the scheduler for up to six objects. A demonstration indicates scalability to up to five robots.**

I. INTRODUCTION

Typical robot manipulation tasks combine discrete and continuous reasoning. Such problems can be described by a high-level specification that captures the discrete relationships between states and actions. These actions involve continuous motions by the robot that cannot be easily represented in a discrete planner. General TMP solvers [1], [2], [3] provide a way to search across discrete choices of actions and continuous robot motions. Unfortunately, these methods have typically been limited to the single-robot case. The availability of multiple manipulators opens up more complex scenarios. Some problems can only be solved with multiple robots (e.g., lifting a heavy object) and others can be solved more efficiently by robots working in parallel. Multi-robot TAMP poses challenges to scalability, in particular when considering optimal task plans.

Consider the real-robot demonstration in Fig. 1; a typical problem for TMP frameworks such as [1], [2], [3]. A domain specification language such as the Planning Domain Definition Language (PDDL) [4] can be used to specify such tasks. Existing techniques repeatedly call a task planner, evaluate motions for actions in the task plan, and return feedback about infeasible actions to get new task plans. A large number of such task planning calls can slow down the computation significantly. To fully utilize multiple robots we need to consider robots executing actions in parallel during the same step such that the optimal task plan has the fewest number of steps. In multi-robot TMP, the computational cost of task planning is typically much larger than the time taken to find a feasible motion plan (Fig 4), because the worst case branching factor of the discrete search space is the cardinality of the Cartesian product of all actions for all robots.

The authors are with the Dept. of Comp. Sci. at Rice Univ. Emails: {tp36, andrew.wells, rahul.shome, kavraki} @rice.edu. The work was supported in part by NSF 1830549 and Rice Univ. Funds. AMW was supported by NASA 80NSSC17K0162.

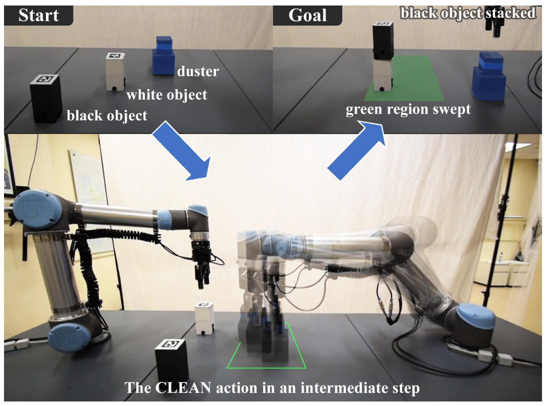


Fig. 1: A motivating demonstration. Top left shows the start with two objects, and a duster. The objective is to a) clean the green area of the table, and b) stack the black block. *Bottom:* The cleaning action requires the white object to be moved away in an intermediate step.

In multi-robot TMP when more than one robot can perform the same action in a step, the task planner might not have information to distinguish between certain robot-action assignments. For example, consider a task plan where the left arm moves block A and the right arm moves block B. If both blocks are reachable by both arms, we trivially know that left moving B and right moving A will still accomplish the same task. Exploring alternative, equivalent robot-action assignments through repeated task planner calls is inefficient. Given a candidate task plan and task definition, the valid permutations of robot-action assignments can be deduced. It is therefore desirable to attempt motion planning for all these candidate assignments before returning to task planning.

We propose a general, multi-robot TMP framework with two major contributions to the state-of-the-art: a) **Task Planner for Simultaneous Actions:** We design an efficient task planner based on Z3 [5], that supports simultaneous actions to address task planning for multiple robots more efficiently. b) **Multi-Robot Scheduler:** We propose an intermediate scheduler layer between task planning and motion planning to evaluate valid permutations of robot-action assignments in a task step. This essentially allows us to group equivalent robot-action assignments together to minimize calls to the task planner and to prioritize search for what is heuristically the most promising assignment. Failure to find any valid permutation triggers a sequential execution fallback. The proposed framework can find an initial, feasible solution quickly and eventually can report a task plan that is optimal in terms of the number of steps.

Our experiments indicate promising performance of our method in problems that current state-of-the-art TMP solvers cannot solve. The results (Sec. V-C) show that our approach can scale better than directly applying a task planner from single-robot TMP [1] to the multi-robot case. The proposed scheduler also improves our performance in benchmark problems with two UR5 robots and up to six objects. In a demonstration we also show scalability to up to five robots. A real demonstration (Fig. 1) is performed on two UR5 arms.

II. RELATED WORK

Different approaches have been proposed to integrate task planning and motion planning for a single robot. In [1], task planning is handled by a general-purpose satisfiability modulo theories (SMT) solver [5]. The SMT solver generates a discrete task plan, and the actions are refined by a motion planner sequentially. If the motion planner fails to instantiate an action, information about the failure is passed back to the SMT solver in the form of a motion constraint. The motion constraint is incrementally added to the solver, which forces the solver to find a new plan. [6] focuses on detecting the "culprit" behind each failure and using that as a constraint. Other approaches such as [2], [3], [7] use heuristic search based on [8], [9] in a similar manner. There exist some optimization-based TMP approaches [10], [11], but they do not have completeness guarantees, while approaches like [1] show probabilistic completeness. There are other lines of work in this area, that almost exclusively focus on single-robots. Of those that consider multiple robots [12], [13], [14], the focus is on mobile robots rather than on manipulation.

Previous works have studied multi-robot planning of manipulators. While coordinated multi-robot motion planning suffers from the curse of dimensionality, some sampling-based approaches have been developed to reduce computational complexity while still tackling the general coordinated problem [15], [16]. Multi-robot task planning was viewed as having two components [17], task decomposition (TD) and task allocation (TA). Approaches such as [18] only consider TA, assuming the decomposition is given, while we do address both aspects implicitly. In object rearrangement [19], the TMP problem with manipulators is simplified by assuming only pick-and-place actions, which can only reason about object locations. Multi-arm rearrangement planning [20], [21] have leveraged inherent structure in the search space in this simplified setting. Rearrangement, though efficient in some common problem domains, is not powerful enough to address general problem specifications and richer actions. Consider Fig. 1 where the white object has identical start and goal poses but must be moved temporarily to satisfy the constraints imposed by a CLEAN action. For rearrangement planning, efficient solutions have been proposed for tabletop settings [20], [21], [19], [22], as well as works that focus on industrial scenarios [23], [24]. These works rely on assumptions that limit their applicability to general domains and do not cover the breadth of the problem we consider.

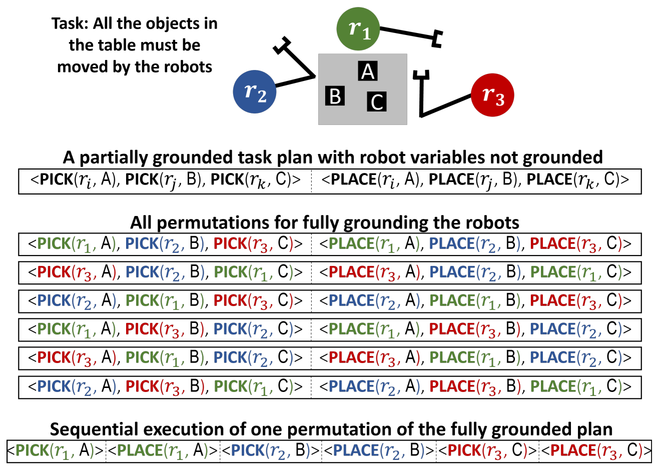
Single robot TMP frameworks are inherently limited in their capability. For instance, a single static robot cannot deal with problems that involve objects beyond its reach, but multiple robots might coordinate to address such problems. Single-robot frameworks can be typically extended to multi-robot scenarios by assuming the robots form a composite robot. However, this exacerbates the curse of dimensionality. In [25], the authors examine 2-arm problems, but do not extend to more robots. They rely on solving relaxed variants with heuristic search, whereas ours focuses on task-optimal solutions using an SMT solver, scheduling, and incremental solving. We also show applicability to up to five robots.

In [1] an approach to single-robot TMP was introduced, and [26] reduced time spent attempting infeasible motion plans. However, the task planning in these approaches does not scale to multiple robots. We present a general framework with a new task-planning approach and use a scheduling layer to improve multi-robot scalability.

III. PROBLEM STATEMENT

We are interested in a multi-robot task-motion planning (MR-TMP), with a set of n robots $\mathcal{R} = \{r_1, r_2, ..., r_n\}$.
Motion Planning: In multi-robot motion planning, we consider n robots moving in a shared workspace. Each robot r_i is represented by a state in a d_i-dimensional configuration space $\mathcal{C}_{r_i} \subset \mathbb{R}^{d_i}$. The composite configuration space $\mathcal{C} = \mathcal{C}_{r_1} \times \mathcal{C}_{r_2} \cdots \times \mathcal{C}_{r_n} \subset \mathbb{R}^{\sum_i d_i}$ is the Cartesian product of the individual spaces. We denote the free composite configuration space as $\mathcal{C}_{\text{free}} \subseteq \mathcal{C}$. We say a *valid* motion described by a trajectory $\pi : [0,t] \to \mathcal{C}_{\text{free}}$ is a time parametrized curve in $\mathcal{C}_{\text{free}}$. We call it a solution to a motion planning problem between a start q_0 and a goal state q_1 if $\pi(0) = q_0, \pi(t) = q_1$. π_i is the i'th robot's motion.
Task Planning: Task planning (TP) is defined over a $Domain_{\text{TP}}$ consisting of a finite set of states, a finite set of actions that allow transition between states, a finite set of constraints, an initial state and a finite set of goal states (TP goals). Typically, problems are described in some specification language such as PDDL. Many off-the-shelf solvers can be used to find a solution (assuming one exists) [9], [8], [27]. This solution is called a *task plan*, which is a sequence of discrete actions: $A = (a^0, a^1, ..., a^h)$. Here h denotes the horizon of the task plan for a single robot.
MR-TMP: In the current work, we are interested in the MR-TMP problems that are *synchronous*, where the robots start and end the execution of actions *synchronously*. Any robot that finishes an action should wait until every robot has completed the execution of its current action. We define the MR-TMP task plan as a sequence of ℓ steps, i.e., each robot's horizon h_i is expanded to match ℓ by including NO-OP actions[1]. Each step is a tuple comprised of the set of n

[1]A robot assigned a NO-OP action at a step does not have to do anything that step but could move to avoid other robots.


Task: All the objects in the table must be moved by the robots
r_1
r_2
r_3
A
B
C

A partially grounded task plan with robot variables not grounded

<PICK(r_i, A), PICK(r_j, B), PICK(r_k, C)> <PLACE(r_i, A), PLACE(r_j, B), PLACE(r_k, C)>

All permutations for fully grounding the robots

<PICK(r_1, A), PICK(r_2, B), PICK(r_3, C)> <PLACE(r_1, A), PLACE(r_2, B), PLACE(r_3, C)>
<PICK(r_3, A), PICK(r_2, B), PICK(r_1, C)> <PLACE(r_3, A), PLACE(r_2, B), PLACE(r_1, C)>
<PICK(r_2, A), PICK(r_1, B), PICK(r_3, C)> <PLACE(r_2, A), PLACE(r_1, B), PLACE(r_3, C)>
<PICK(r_1, A), PICK(r_3, B), PICK(r_2, C)> <PLACE(r_1, A), PLACE(r_3, B), PLACE(r_2, C)>
<PICK(r_3, A), PICK(r_1, B), PICK(r_2, C)> <PLACE(r_3, A), PLACE(r_1, B), PLACE(r_2, C)>
<PICK(r_2, A), PICK(r_3, B), PICK(r_1, C)> <PLACE(r_2, A), PLACE(r_3, B), PLACE(r_1, C)>

Sequential execution of one permutation of the fully grounded plan

<PICK(r_1, A)><PLACE(r_1, A)><PICK(r_2, B)><PLACE(r_2, B)><PICK(r_3, C)><PLACE(r_3, C)>

Fig. 2: This demonstrates a problem where the TP cannot differentiate between the robots for actions in the shared workspace; thus the robot variables r_i, r_j, r_k are partially grounded. All the permutations for fully grounding the robots are equivalent to the TP, and can be evaluated instead by the scheduler. Note the actions can also be executed sequentially.

actions being performed by each arm at that step along with the corresponding motions, i.e., the j'th step is an n-tuple

$$s_j = \langle (a^j_{r_1}, \pi^j_{r_1}), (a^j_{r_2}, \pi^j_{r_2}), \cdots, (a^j_{r_n}, \pi^j_{r_n}) \rangle$$

where each a is an action (possibly NO-OP), and the concurrent multi-robot motion for the j'th step is

$$\pi^j = (\pi^j_{r_1}, \pi^j_{r_2}, \cdots, \pi^j_{r_n})$$

Definition 1: Given a TP domain $Domain_{\text{TP}}$ and a motion planning domain $Domain_{\text{MP}}$, *synchronous* MR-TMP is the problem of finding a feasible ℓ-step task-motion plan:

$$\overline{\mathbf{T}} = (s_1, s_2, \cdots, s_\ell),$$

such that after executing $\overline{\mathbf{T}}$, TP goal is reached.

A feasible synchronous task and motion planning solution $\overline{\mathbf{T}}$ must satisfy the following conditions:

a) The projected set of sequences of actions for each robot within $\overline{\mathbf{T}}$ satisfies $Domain_{\text{TP}}$, i.e.,

$$(a^1_{r_i}, a^2_{r_i}, \cdots a^\ell_{r_i}) \quad SATISFIES \quad Domain_{\text{TP}}\ \forall r_i$$

and successfully transitions the initial to the goal TP state. The $Domain_{\text{TP}}$ satisfaction depends on all actions that started or ended at or before each step.

b) Each successive pair of motions is continuous for each robot: The end configuration of $\pi^j_{r_i}$ is the start configuration of $\pi^{j+1}_{r_i}$, $\forall j < \ell$. This means that each successive pair of composite configurations describes a coordinated motion for the j^{th} step from $(\pi^j_{r_j}(0), \pi^j_{r_j}(0), ..., \pi^j_{r_j}(0))$ to $(\pi^{j+1}_{r_1}(0), \pi^{j+1}_{r_2}(0), ..., \pi^{j+1}_{r_n}(0))$.

c) The composite motion at the j^{th} step π^j that corresponds to the step s^j needs to satisfy $Domain_{\text{MP}}$, i.e., it cannot have collisions with obstacles or other robots.

We focus on minimizing the number of steps in the task plan, ℓ, which is called *makespan*.

IV. METHOD

We present a framework for MR-TMP, which consists of three layers: *a task planner, a scheduler, and a motion planner*. Each layer operates at a different level of abstraction. The task planner reasons over the discrete task domain modeled in PDDL, describing actions, preconditions, and effects. The output is a symbolic task plan consisting of partially grounded robots and actions (which we will call a partially grounded task plan). Note that grounding a symbolic expression means to assign values to the variables in it. In order to execute the partially grounded task plan on the robot, each robot and action has to be grounded and then each action has to be refined to robotic motions through motion planning step by step. Such motions might be infeasible due to geometric collisions, despite the task step being valid in the task domain. This information will be discovered in the motion planning step, and passed back to the task planner as motion constraints. The replanning inside the task planner can be done in an incremental fashion, which avoids replanning from scratch with each new constraint, so as to reduce the task planning time. In this way, the strategy will eventually discover the geometrically feasible groundings of the actions involved in the task plan. Task planning can be naively extended to multi-robot problems by treating the robots as a composite robot. However, this naive approach does not scale well. In our work we design an efficient task planning module that allows simultaneous actions for the multi-robot problems, using an SMT solver [5].

We also add a middle scheduler layer that is aware of the geometric feasibility of motion plans. It evaluates different robot-to-action assignments at each step. The goal of the scheduler is twofold. Firstly, it can reduce the number of task planner calls by evaluating equivalent groundings, which the naive approach would only search by re-querying the task planner. Secondly, it can bias search by ordering equivalent plans based on geometric insight.

A. Task Planning: Simultaneous SMT Planning

The input to the TP is $Domain_{\text{TP}}$, encoded using PDDL. The task planner determines *which* actions should be taken *when* based on robot reachability and availability. We modify an SMT-variant of a typical SAT planner similar to [1] by relaxing the mutual-exclusion axioms. In a SAT planning formulation, mutual-exclusion axioms are used to ensure that only one action is taken at each time step. *We instead allow one action per robot at each timestep.* We need to ensure the chosen simultaneous actions have no conflicts, which we encode as extra cross-robot mutual-exclusion constraints inspired by the work in [27]. For example, when we encode the transition from the time step j to $j+1$, actions $a^j_{r_i}$ and $a^j_{r_k}$ can execute simultaneously only if the preconditions of $a^j_{r_i}$ are not violated by the effects of $a^j_{r_k}$, and vice versa. Extra book-keeping is required to track which motion failures are caused by which robots and to block actions by pushing the corresponding constraints onto the SMT solver's stack.

Note that TP cannot distinguish among robots with the same capabilities, and the abstraction only includes predicates defining which regions each robot can reach. Thus the output of the task planner is agnostic to the geometry within a region. Therefore, it may not always be possible to fully ground the *robot* variable of a task action. If the actual poses involved in some actions of a task step can be reached by multiple robots, the task planner does not distinguish between them; instead, a partially grounded task plan is returned, where the *robot* variable symbolically represents *any robot that can reach this location.*

B. Scheduler

The scheduler, which is aware of the geometry, is responsible for fully grounding the *robot* variables in the partially grounded task plan (see example scenario in Fig. 2).

If n robots have the same capabilities, i.e., the problem resides in the commonly reachable workspace, each step can have at most $n!$ possible valid robot assignments to the n actions. The scheduler evaluates these permutations.

The scheduler leverages geometric information (e.g., robot, object, and grasp poses) to a) accelerate constraint discovery, and b) heuristically prioritize promising permutations that can help reduce motion planning failures. As in [26], the heuristic does not affect probabilistic completeness. If the motion planning keeps failing, the scheduler can exhaust all permutations for a step. After complete failure at a step, we combine all the motion constraints and add them back to the task planner. *This means we only need to call the task planner once to evaluate $n!$ fully-grounded plans in the worst case.* This typically reduces the number of task planner calls.

C. Motion Planner

A probabilistically complete sampling-based motion planner is used as the underlying centralized motion planning primitive. Specifically we use RRT-Connect [28]. Each action is also associated with a goal sampler that informs the motion planner of configurations to reach. In typical manipulation domains this involves solving inverse kinematics problems for grasping, placement, or handoff poses.

D. Algorithm

The proposed framework is shown in Alg. 1. The task planner performs incremental task planning, which allows simultaneous actions (Alg. 1 line 5). A candidate task plan is sent to the scheduler, which assigns each action to a real robot by finding a permutation of robots for each appropriate step (Alg. 1 line 11). The scheduler predicts the best permutation based on heuristics. In our implementation, the permutations are sorted by a heuristic measure of the maximum Euclidean distance between the base pose of the robots and its corresponding action end-effector pose. Note that other types of heuristics apply (e.g., estimates of motion duration). The actions in the permuted step are refined by the motion planner (Alg. 1 line 13). Coordinated motion planning is performed if a step involves simultaneous actions. If the motion planning fails, the information is reported to the scheduler, which tries the next best permutation of that step (Alg. 1 line 11), until all permutations are exhausted. Then we invoke a fallback behavior that sequences the simultaneous actions (Alg. 1 line 19). This sequential execution helps to find the first feasible task-motion plan faster. Whenever motion planning fails, we encode the failure as a new motion constraint (Alg. 1 line 15). If all permutations are exhausted and the fallback also fails, we end the motion refinement and feed all of the motion constraints to the task planner at once in the next iteration of the high level loop TP (Alg. 1 line 5). If all the steps have a valid motion plan, the task and motion plan $\overline{\mathbf{T}}$ is reconstructed fully. The search keeps running till the task planner runs out of options (Alg 1 line 6), which triggers a reset of constraints and increases the motion planning time budget. If a plan $\overline{\mathbf{T}}$ was completely refined without sequential fallbacks (Alg. 1 line 23), this is the best makespan coordinated task plan, and the search ends.

This algorithm demonstrates *anytime* behavior - it can quickly obtain an initial feasible task and motion plan that might include sequential executions, but keeps searching till an optimal makespan coordinated solution is discovered.

Algorithm 1: MR-TMP

```
Input: $Domain_{\mathrm{TP}}, Domain_{\mathrm{MP}}$, TP-$Goal$
Output: Output task and motion plan, $\overline{\mathbf{T}}$
1  INITIALIZE($Domain_{\mathrm{TP}}$)
2  $\overline{\mathbf{T}} \leftarrow \emptyset$;   $\phi \leftarrow \emptyset$; // Motion Constraints (MC)
3  $\tau \leftarrow \tau_{\mathrm{init}}$; // Set initial MP time budget
4  while $\overline{\mathbf{T}} = \emptyset$ do
       // Task planning
5      $A \leftarrow$ Z3SIM($Domain_{\mathrm{TP}}, \phi$, TP-$Goal$)
6      if $A = \emptyset$ then
7          $\phi \leftarrow \emptyset$;   $\tau \leftarrow \tau + \Delta\tau$;   continue;
8      foreach $s \in A$ do
9          $\pi \leftarrow \emptyset$ // Initialize empty motion plan
10         while $\pi = \emptyset$ do
               // The best permutation of the step
11             $s \leftarrow$ NEXTBESTPERMUTATION($s$)
12             if $s \neq \emptyset$ then
                   // Motion planning
13                 $\pi \leftarrow$ MP($s, Domain_{\mathrm{MP}}, \tau$)
14                 if $\pi = \emptyset$ then
15                     $\phi \leftarrow$ ADDNEWMC($\pi, s$)
16             else
17                 break // No permutations left
18         if $\pi = \emptyset$ then
               // Sequential fallback
19             $\pi \leftarrow$ SEQ($s, Domain_{\mathrm{MP}}, \tau$);
20         if $\pi \neq \emptyset$ then
21             $\overline{\mathbf{T}} \leftarrow$ APPEND($\overline{\mathbf{T}}, \{s, \pi\}$)
22         else  $\overline{\mathbf{T}} \leftarrow \emptyset$;   break ;
23     if $\overline{\mathbf{T}} \neq \emptyset$ and NOSEQFALLBACK then break;
       // Otherwise continue finding optimal plan
24 return $\overline{\mathbf{T}}$
```

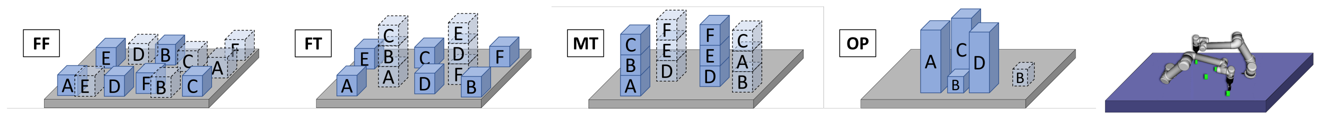


Fig. 3: *Left 4*: Example benchmarks. Solid-line cubes are initial locations, and dashed-line ones show the task goals. *Right*: The simulation setup in DART [29], showing that the robots must coordinate in shared workspace.

V. RESULTS

We focus on highlighting the benefits of using our approach in experimental evaluations by successfully and optimally (over makespan) solving larger problems faster.

A. Methods

We present a comparison of our proposed MR-TMP framework with three baseline TMP approaches.

Single-robot TP: A single-robot TMP framework [1] using the same domain as our method.

Fully-Composite TP: A MR-TMP framework built upon a state-of-the-art single-robot TMP framework [1] assuming the robots form a composite robot. For n robots, the PDDL encoding is a cross product of n copies of the a single-robot encoding. This brute-force enumeration of combinations of multi-robot actions drastically increases the size of the input to the taks planner, which severely limits the applicability of optimal TP solvers like Z3. When Z3 does not scale for fully composite optimal planning, a typical greedy solver is used, which usually only finds suboptimal plans.

(Baseline) Optimal Simultaneous TP Without Scheduler (WS): This baseline follows Algo. 1 without the scheduler. This reflects the TP changes, but it has to reinvoke task planner once per motion failure.

(Proposed) Optimal Simultaneous TP With Scheduler finding Initial (PI)/Final (PF) solution: This follows Algo 1 and uses the simultaneous variant of Z3 and a scheduler.

B. Benchmark Problems

Setup: We evaluate the scalability and the performance of the methods in a tabletop block-world domain with cuboidal objects admitting top-down grasps. The considered actions are PICK, PLACE, STACK, UNSTACK, HANDOFF.

For all the evaluated methods, we discretize the workspace into *regions of unique reachability*. The only geometric information that we encode in PDDL for the task planner is which region is reachable to which robots. This is more efficient than discretizing the space as in [1][2]. Whenever such a region has to be used, a valid location is sampled online. In our problems, we have two robots, and three reachability regions. We test our methods for randomly sampled problems involving $3-6$ objects. All benchmarks are run 50 times, and we ensure 25 of these involve starts and goals in the commonly reachable region. All experiments were performed on a 4.0GHz Intel i7 processor with a memory limit of 4GB, and a timeout of 600s. *It should be noted that the same framework is powerful enough to tackle the variety of benchmark problems posed here.*

[2]Note that such discretization do not require prior knowledge of the workspace, and can be automated.

Benchmarks: We created four tabletop scenarios to test the baselines for two robots. Each of the robots can only reach part of the workspace, and collaboration might be necessitated by problems spanning different reachability regions The benchmarks are introduced in Fig. 3.

Flat to flat (FF): Note that the starts and goals can overlap, so this is not assured to be a monotone problem[3].

Flat to tower (FT): Again, the starts and goals can overlap, and the objective is to build two towers.

Move towers (MT): The order of the blocks in two towers is randomly sampled, making the problem non-monotone.

Obstructed Pick (OP): This is specially designed in such a way that the target object is obstructed by (two or three) taller adjacent objects. It violates assumptions typically made in rearrangement and motivates the importance of restrictive geometric constraints arising from real scenes.

C. Analysis of Results

Solution Quality: For the discussion below on TP and MR-TMP results, the solution quality is represented by the number of steps (or makespan) in the task plan.

TP Results: We first compare our proposed simultaneous variant of Z3 to the fully-composite encoding. *Composite planning using* Z3 *planner runs out of memory for more than three objects with two robots.* We instead use a typical feasible Fast Downward (FD) [9] solver for the composite encoding on these problems. For FD's heuristic parameter we used *lazy greedy* because it performed better than alternatives (*LAMA and Autotune*) on our example problems. Table I shows the task planning comparison on FF, FT, MT with 4 or 6 objects. Even though we use a composite FD, in most cases it is slower to find the initial task plan compared to our method. In addition, our method returns an optimal task plan, while FD is mostly sub-optimal, especially for MT, where its solutions are more than twice the optimal makespan.

TABLE I: Task planning benchmark

	(Ours) Z3-Simultaneous		**FD composite**	
Problem	**Makespan**	**Time(s)**	**Makespan**	**Time(s)**
FF4	5	1.10	7	8.18
FF6	7	24.06	11.5	103.44
FT4	4	0.47	7.5	4.20
FT6	6	3.86	7	23.55
MT4	7	1.17	17	3.13
MT6	10	70.11	27	21.5

MR-TMP **Results:** Fig. 4 shows the timing results among the MR-TMP comparison methods. There are two entries for our proposed method: the initial solution, which might include some sequential executions, and also our final output. Note that the proposed method can get both the first feasible

[3]A monotone problem is one where the object requires only one grasp, whereas non-monotonicity can require multiple interactions per object.

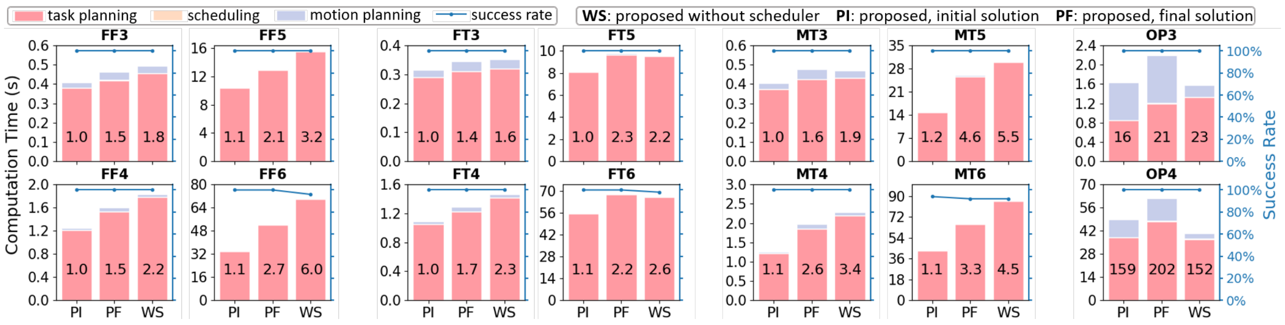


Fig. 4: Planning time and success rate comparison on the benchmarks. Each subplot shows the benchmarking result for one scenario (e.g. FF3 is the FF problem with 3 objects). Each bar represents the total planning time of one method (note the different scale of each subplot). The scheduling time is negligible compared to task planning and motion planning. The number of task planner calls is shown inside each bar. The success rate is the line chart with y-axis on the right side.

plan and the final optimal plan with higher success rate than the method without a scheduler. The initial solution is faster on average, and only reports slightly worse makespan from some instances of sequentially scheduled steps (Fig. 4). We can keep running our method to obtain optimal makespan solutions. The solution quality for multiple robots is significantly better than the single robot solutions as many actions can be performed in parallel. It should be noted that a single robot cannot solve any problems lying outside its reachability. The optimal makespan solution is achieved by both our framework's final output, as well as the variant without a scheduler. Not using scheduling causes an increased computation time in most of the benchmarks. In FT6, WS failed in one problem, while PF found the optimal plan in 9.5 mins, leading to a slightly worse average. Fig. 4 also indicates that task planning can be much more expensive than motion planning in typical tabletop environments [26], [30], which motivates our method design.

Note on number of task planner calls: The scheduler explores equivalent robot assignments to multi-robot actions. In randomized problems (FF, FT, MT) this strategy saves task planning calls to find the optimal feasible task plan. In FT5, PF is marginally worse than WS due to the stochasticity of planning. In OP the goal can be achieved with a simple pick and place, but geometric constraints from the taller blocks require all equivalent plans in smaller horizons to be explored. Eventually the task planner gets a feasible task plan among horizons 4 and 5 respectively in OP3 and OP4. The order in which these task plans are eliminated in PI, PF and WS cannot be controlled. Sometimes WS attempts single action steps involving the target object early on in the search. These can immediately invalidate a large set of equivalent actions, as opposed to synchronized steps which include a constraining action. PF has slower runtimes in OP benchmarks, but nonetheless finds the optimal task plans.

The average makespan of each methods on the benchmarks is shown in Fig. 5, which also shows results from single robot solutions (for the problems where they succeeded).

Scalability Test: Fig. 6 shows a scalability test on a FF problem instance. We tested for 2 to 5 robots, with tasks requiring each robot to move two objects. Our method scales up to 5 robots. For 6-robots we timed out, taking 35mins.

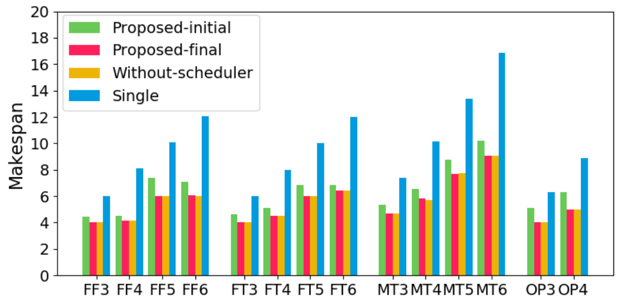


Fig. 5: The average makespan of the output schedule. The results are grouped by problem class. The problem type and number of objects are indicated below each group of bars.

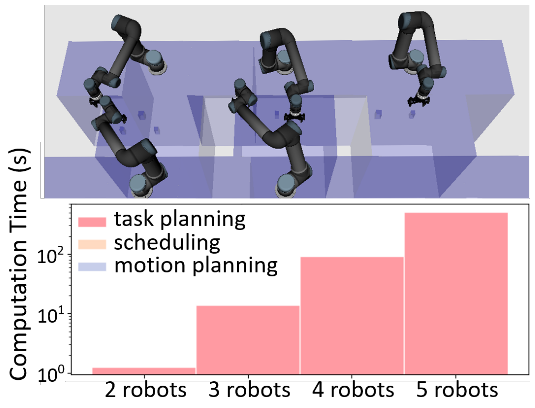


Fig. 6: *Top:* A simulated scene in DART [29]. *Bottom:* The total planning time. We have n robots and $2n$ objects in each scene, creating one instance of an FF problem.

Real-World Demonstration - CLEAN: A motivating demonstration is designed using a CLEAN action, and is shown in Fig 1 (also included in our video). The problem is an interesting display of the power of our framework. Note that the white object has the same location in both the start and goal states, but is required to be moved due to the constraints of the CLEAN action. Our ability to express such arbitrary constraints is a key benefit of our generality. The solution was executed on two UR5 robots. The task plan consisted of 7 steps, and took 1.8s to compute.

VI. CONCLUSION

In this work we have studied the general MR-TMP problem and proposed an efficient framework that can address problems involving multiple manipulators. We successfully identified ways to overcome bottlenecks of performance and scalability that arise from general MR-TMP. We introduced a simultaneous task planner that scales more efficiently than composite planning. We presented a novel intermediate scheduling layer that offers an anytime behavior with fast initial solutions, and eventual optimal task plans. We focus on robot-action assignments for the scheduler, but it should bring efficiency to reasoning over any equivalence class of task plans, for example object-action assignments. We demonstrated the benefits of our approach in benchmarks on two arms, a scalability test for up to five arms, and in a real-world example.

There are rich avenues for future work to incorporate reasoning about complex actions and grasping, and exploring quality guarantees of the complete TMP solution.The search in MR-TMP can also benefit from experience and use learning to improve heuristics. The current work provides an important step towards applying multi-robot TMP solvers to real-world applications.

REFERENCES

[1] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[2] S. Srivastava *et al.*, "Combined task and motion planning through an extensible planner-independent interface layer," in *Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 639–646.

[3] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: Leveraging symbolic planning for efficient task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, 2018.

[4] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld, "Pddl - the planning domain definition language," 08 1998.

[5] L. de Moura and N. Bjørner, "Z3: an efficient smt solver," vol. 4963, 04 2008, pp. 337–340.

[6] F. Lagriffoul and B. Andres, "Combining task and motion planning: A culprit detection problem," *The Int. Journal of Robotics Research*, vol. 35, no. 8, pp. 890–927, 2016.

[7] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, J. C. Beck, O. Buffet, J. Hoffmann, E. Karpas, and S. Sohrabi, Eds. AAAI Press, 2020, pp. 440–448. [Online]. Available: https://aaai.org/ojs/index.php/ICAPS/article/view/6739

[8] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, pp. 253–302, 2001.

[9] M. Helmert, "The fast downward planning system." *Journal Artificial Intelligence Resesearch*, vol. 26, pp. 191–246, 2006.

[10] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[11] D. Driess, J.-S. Ha, and M. Toussaint, "Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image," 07 2020.

[12] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *Int. Journal of Humanoid Robotics (IJHR)*, vol. 2, no. 04, pp. 479–503, 2005.

[13] F. Gravot, S. Cambon, and R. Alami, "aSyMov: a planner that deals with intricate symbolic and geometric problems," in *Int. Symp. on Robotics Research (ISRR)*. Springer, 2005, pp. 100–110.

[14] J. Motes, R. Sandström, H. Lee, S. Thomas, and N. M. Amato, "Multi-robot task and motion planning with subtask dependencies," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3338–3345, 2020.

[15] K. Solovey, O. Salzman, and D. Halperin, "Finding a needle in an exponential haystack: Discrete rrt for exploration of implicit roadmaps in multi-robot motion planning," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 591–607.

[16] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris, "drrt*: Scalable and informed asymptotically-optimal multi-robot motion planning," *CoRR*, vol. abs/1903.00994, 2019. [Online]. Available: http://arxiv.org/abs/1903.00994

[17] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International journal of robotics research*, vol. 23, no. 9, pp. 939–954, 2004.

[18] S. Choudhury, J. K. Gupta, M. J. Kochenderfer, D. Sadigh, and J. Bohg, "Dynamic multi-robot task allocation under uncertainty and temporal constraints," *arXiv preprint arXiv:2005.13109*, 2020.

[19] S. D. Han, N. M. Stiffler, A. Krontiris, K. E. Bekris, and J. Yu, "Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps," *arXiv preprint arXiv:1711.07369*, 2017.

[20] R. Shome and K. E. Bekris, "Synchronized multi-arm rearrangement guided by mode graphs with capacity constraints," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Oulu, Finland, 06/2020 2020. [Online]. Available: https://arxiv.org/pdf/2005.09127.pdf

[21] R. Shome, K. Solovey, J. Yu, K. E. Bekris, and D. Halperin, "Fast and high-quality dual-arm rearrangement in synchronous, monotone tabletop setups," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Mérida, México, 12/2018 2018.

[22] A. Krontiris and K. Bekris, "Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner," 05 2016, pp. 3924–3931.

[23] V. Tereshchuk, J. Stewart, N. Bykov, S. Pedigo, S. Devasia, and A. G. Banerjee, "An efficient scheduling algorithm for multi-robot task allocation in assembling aircraft structures," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3844–3851, Oct 2019.

[24] J. K. Behrens, R. Lange, and M. Mansouri, "A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks," in *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*. IEEE, 2019, pp. 8705–8711. [Online]. Available: https://doi.org/10.1109/ICRA.2019.8794022

[25] A. Akbari, F. Lagriffoul, and J. Rosell, "Combined heuristic task and motion planning for bi-manual robots," *Autonomous Robots*, vol. 43, pp. 1575–1590, 2018.

[26] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1255–1262, 2019.

[27] J. Rintanen, "Madagascar : Scalable planning with SAT," 2014.

[28] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.

[29] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018. [Online]. Available: https://doi.org/10.21105/joss.00500

[30] F. Lagriffoul, N. T. Dantam, C. Garrett, A. Akbari, S. Srivastava, and L. E. Kavraki, "Platform-independent benchmarks for task and motion planning," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3765–3772, 2018.