Toward Optimal Tabletop Rearrangement with Multiple Manipulation Primitives

Baichuan Huang Xujia Zhang Jingjin Yu

***Abstract*— In practice, many types of manipulation actions (e.g., pick-n-place and push) are needed to accomplish real-world manipulation tasks. Yet, limited research exists that explores the synergistic integration of different manipulation actions for optimally solving long-horizon task-and-motion planning problems. In this study, we propose and investigate planning high-quality action sequences for solving long-horizon tabletop rearrangement tasks in which multiple manipulation primitives are required. Denoting the problem *rearrangement with multiple manipulation primitives* (REMP), we develop two algorithms, *hierarchical best-first search* (HBFS) and *parallel Monte Carlo tree search for multi-primitive rearrangement* (PMMR) toward optimally resolving the challenge. Extensive simulation and real robot experiments demonstrate that both methods effectively tackle REMP, with HBFS excelling in planning speed and PMMR producing human-like, high-quality solutions with a nearly 100% success rate. Source code and supplementary materials will be available at https://github.com/arc-l/remp.**

I. INTRODUCTION

Real-world manipulation tasks, e.g., rearranging a messy tabletop or furniture in the house, often require multiple manipulating primitives (e.g., pick-n-place, pushing, toppling, etc.) to accomplish. When rearranging small/light objects, e.g., a cell phone on a table or a small chair in a room, it is convenient to do a *pick-n-place*, i.e., to pick up the object, lift it above other objects, move it across the space to above its destination on the table, and then place it. On the other hand, for handling large/heavy objects, e.g., a thick book or a heavy couch, *pushing* or *dragging* close to the space's surface is more commonly adopted, executed with added caution. In this case, planning the object's motion trajectory must consider avoiding colliding with other objects more carefully. Solving such long-horizon task-and-motion planning tasks efficiently and optimally is highly challenging, as it involves not only an extended horizon but also selecting among multiple types of manipulation primitives at each step, both of which add to the combinatorial explosion of the search space.

Toward quickly and optimally solving rearrangement tasks using multiple manipulation primitives, we focus on a tabletop setting where both *pick-n-place* and *pushing* are employed to rearrange objects (see Fig. 1). Many objects, such as those shown in Fig. 1(e), cannot be easily picked up and moved around without damaging or disassembling the object. For example, as shown in Fig. 1(f)(g), books and certain boxes cannot be moved around using suction-based pick-n-place manipulation primitive (note that it is also difficult to do pick-n-place using fingered grippers). However, these objects can be effectively rearranged using a *pushing* manipulation primitive in which the suction-based end-effector holds the object on or close to the tabletop and pushes/drags the object around (see Fig. 1(c)). We call the frequently encountered yet largely unaddressed problem *rearrangement with multiple manipulation primitives* (REMP). This study on REMP brings the following contributions:

- With the formulation of REMP, we propose a first formal study of solving long-horizon rearrangement tasks utilizing multiple distinctive precision manipulation primitives with the goal of computing an optimized manipulation sequence. Due to its high practical relevance, REMP constitutes an important specialized task and motion planning problem.
- We developed two novel algorithms for REMP, the first of which is a fast rule-based solution capable of effectively and quickly solving non-trivial REMP instances. The second, leveraging Monte Carlo tree search (MCTS) [1] and parallelism to look further into the planning horizon, delivers a much higher success rate for more challenging

Fig. 1: (a) Overview of system setup, a camera is mounted on the end-effector for perception. (b) - (d) An example case and an intermediate step in solving it. (e) Example objects requiring *push*. (f:) *pick-n-place* may break the book. (g) *pick-n-place* will separate a box, failing to pick it up.

B. Huang and J. Yu are with the Department of Computer Science, Rutgers, the State University of New Jersey, Piscataway, NJ, USA. Emails: {baichuan.huang, jingjin.yu}@rutgers.edu. X. Zhang is with Southern University of Science and Technology, China. This work was done when X. Zhang was an intern at Rutgers University. This work was supported by NSF awards 1845888, 2132972, and 2309866.

tasks, providing higher-quality solutions simultaneously.
- We thoroughly evaluate our methods in simulation and extensive real robot experiments. In particular, our real robot experiments with integrated vision solutions, demonstrate that our algorithms can be readily applied to interact with everyday household objects in real-world scenarios.

II. Related Work

Robot manipulation [2] actions can typically be classified into two primary categories: prehensile [3], [4] and non-prehensile [5]–[7]. In prehensile manipulation, a robot's gripper secures a grasp of the target object, allowing the object to move in with the end-effector. Fingered grasping and suction are two common prehensile manipulation actions. Non-prehensile manipulation leverages contact between objects and the environment, e.g., pushing, dragging, and toppling. While prehensile actions are limited in variety, non-prehensile actions abound [2], [8], [9]. Limiting to tabletop rearrangement, pushing has been used extensively [10], [11] for relocating objects without lifting the object away from the surface. However, pushing objects to follow a desired $SE(2)$ (i.e., 2D translation and rotation) trajectory with precision and speed is difficult. This leads to our design of a suction-based pushing manipulation primitive (Fig. 1(c)), which is a mixed prehensile/non-prehensile primitive.

Precisely and efficiently rearranging objects on a tabletop workspace, especially when object density is high, requires carefully allocating available space on the tabletop to facilitate the relocation of objects. Tang et al. [12] consider removing objects from the workspace during the rearrangement of certain objects, reducing search space. We note that push is also used here but is used in an assistive capacity. Similarly, external space is used to temporally hold objects so that more space becomes free for rearranging in [13]. Rearranging objects in confined dense clutter scenes is considered in [3] where a heuristics-guided lazy search is applied. In [4], [14], [15], rearrangement has been studied in shelve setups. In prior research by Han et al. [16], the problem is translated into a graph-based formulation, factoring in the dependency graph among objects. We note that these studies on tabletop arrangement exclusively work with pick-n-place manipulation. As such, these methods do not readily translate to REMP.

Many research efforts leverage reinforcement learning to address related challenges. The episodic nature of rearrangement often requires considering extended planning horizons. Consequently, combining MCTS with a predictive network is often an effective strategy. For instance, Huang et al. [17] demonstrated the use of push actions to clear space, enabling the target object retrieval from cluttered environments. A similar scenario, albeit in 3D, has also been explored in [18]. The concept of segregating objects into distinct regions using push actions is explored in [19], while transitioning objects from initial to goal positions via push actions is discussed in [7]. Notably, all these studies employ predictive networks to accelerate the simulation phase of MCTS. In contrast, this paper diverges from these approaches in that we primarily leverage MCTS for planning, which already demonstrates commendable performance. Even with recent advancements in network-based push prediction, as seen in [10], there is still significant reliance on traditional path-planning algorithms for guidance. The efficacy of using predictive networks to direct MCTS, especially in push action scenarios necessitating object avoidance, remains an open question.

III. Preliminaries

A. Rearrangement with Multiple Manipulation Primitives

We now specify the concrete *rearrangement with multiple manipulation primitives* (REMP) studied in this work. Let the workspace be $\mathcal{W}$ is a 2D rectangle. The robot is provided with a start image (state) s_s and a goal image (state) s_g containing the initial and desired object arrangements. The robot must rearrange the objects to match the configurations specified in s_g. Two manipulation primitives are permitted: *pick-n-place* $\mathcal{A}_{pp}$ and *push* (from top) $\mathcal{A}_{pt}$. The robot's objective is to complete the task efficiently in terms of the *execution time*. The start and goal states and the objects' transportation should be collision-free, and all objects should remain within the workspace. It is assumed that all tasks are feasible, i.e., there is always a viable solution $\mathcal{P} = \{a_1, a_2, ..., a_n\}$ leading from the start state s_s to the goal state s_g, where $a \in \{\mathcal{A}_{pp}, \mathcal{A}_{pt}\}$.

A state s_t represents the pose of objects at time t. A pick-n-place action is specified by pick pose (x_0, y_0, θ_0) and place pose (x_1, y_1, θ_1). A push action is specified by trajectories $\{(x_0, y_0, \theta_0), ..., (x_n, y_n, \theta_n)\}$, where the robot holds the object against the tabletop at the initial pose, and then push it following the waypoints, ending at the final pose.

One assumption is that the object should be capable of being stably positioned on the workspace, as all primitives are considered to be quasi-static.

B. Monte Carlo Tree Search

The Monte Carlo tree search (MCTS) algorithm has broad applications. It is prevalent in turn-based tasks such as the game of Go [20], but its usage extends beyond such contexts. MCTS plays a crucial role in solving rearrangement tasks [21]–[23]. As an *anytime* tree search algorithm, MCTS is designed to run for a fixed amount of time, each consisting of four stages: selection, expansion, simulation, and backpropagation. Fundamentally, MCTS preferentially exploits nodes that yield superior outcomes. To strike a balance between exploration and exploitation in the selection phrase, an *upper confidence bound* (UCB) [24] formula is utilized (see Eq. 1), where n is the parent node of n', and $Q(n')$ is the total reward n' received after $N(n')$ visits.

$$\underset{n' \in \text{children of } n}{\arg\max} \frac{Q(n')}{N(n')} + c\sqrt{\frac{2\ln N(n)}{N(n')}}, \tag{1}$$

IV. Methodology

This work addresses the primary challenge of synergistic integration of pick-n-place and push actions. Because planning a push involves collision-free path planning in $SE(2)$, which is time-consuming, it poses a significant challenge if many push actions are explored. MCTS offers a solution capable

of elegantly negotiating between the two disparate actions while maintaining optimality, given ample planning time.

A. Action Space Design

Planning requires searching through candidate manipulation actions, which must first be *sampled*. Action space design refers to action sampling, which plays a critical role in dictating the expansion of the tree search because there are an uncountably infinite number of possible manipulation actions. In sampling pick-n-place actions, we must ensure the place pose is collision-free. The same applies to a push action's final pose (though, in addition, the entire trajectory connecting all waypoints for a push action must be collision-free). Four criteria are applied to sample the place/final poses for pick-n-place/push actions at the current state s_t:

1) **Random**. A naive approach randomly samples collision-free place/final poses for pick-n-place/push actions.
2) **Around Current and Goal**. Random sampling is not always efficient. For object o_i, favoring regions around the current and goal poses of o_i in s_g can be helpful.
3) **Grid**. Additionally, we adopt a grid-based sampling strategy. By modeling an object as a 2D polygon, we encapsulate it within a rotated bounding box. This allows us to generate a tiled representation in the workspace, denoted as $\mathcal{W}$, resembling a grid structure. This method proves advantageous for covering boundary areas, which can be hard to sample through random methods.
4) **Direct to Goal**. If o_i can be directly placed at its goal, this pose will be prioritized over the above three samplings in the search process.

Once the place/final poses have been sampled, a $\mathcal{A}_{pp}$ sample is obtained. However, $\mathcal{A}_{pt}$ requires an additional step - generating a trajectory from the current pose to the place pose for o_i. We employ RRT-connect [25] during the tree search and LazyRRT [26] for robot execution to produce such collision-free trajectories with a set time limit. An example of sampling the final pose for a push action is shown in Fig. 2. The criteria for sampling are crucial in solving the problem. Relying solely on standard uniform sampling often proves inefficient, particularly when sampling around boundaries and certain edge cases. While one might consider increasing the sample size to cover these edge cases, this inadvertently leads to many redundant actions that are time-consuming to process.

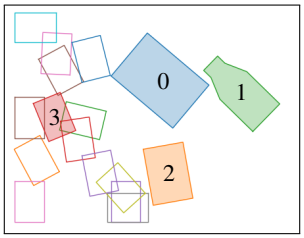

Fig. 2: Consider action sampling for labeled 3 to be manipulated using push (there are a total of four objects). The absence of sampled actions in the right region is attributed to obstructions posed by objects 0, 1, and 2, preventing the movement of object 3 to that area.

B. Hierarchical Best-First Search

The first algorithm we designed for REMP is a (greedy) best-first search algorithm called *hierarchical best-first search* (HBFS). HBFS is outlined in Alg. 1 and operates according to the following sequence of steps:

- (Lines 3-4) When objects can be directly moved to their goal poses, an action cost is computed for each. The action yielding the smallest cost is then applied.
- (Lines 5-9) For each object o_i, HBFS identifies which objects occupy o_i's goal and attempts to displace these obstructing objects in the direction of their respective goals. If no actions are feasible in this direction, a random action is sampled. Again, the action associated with the smallest cost is selected and implemented.
- (Line 10) If the above steps do not yield a viable action, an action is randomly selected for execution.

The above three phases of HBFS may best be viewed as a three-level hierarchical search. To boost its performance and solution optimality, HBFS is implemented by leveraging multi-core capabilities of modern CPUs. This is realized by executing multiple HBFS in parallel and choosing the best action among the returned solutions.

Algorithm 1: Hierarchical Best-First Search (HBFS)

```
1  Function HBFS(s_s, s_g)
2      s ← s_s, A ← ∅
3      for o_i in s do A ← A ∪ {move o_i to its goal}
4      if |A| > 0 then return the lowest cost action from A
5      for o_i in s do
6          obs ← objects occupy the goal pose of o_i
7          for o_j in obs do
8              A ← A ∪
                 {move o_j towards its goal, otherwise at random}
9      if |A| > 0 then return the lowest cost action from A
10     return a randomly sampled action
```

C. Speeding up MCTS with Parallelism

Standard MCTS is more straightforward to implement, but it does not fully utilize multi-core processing capabilities of modern hardware. We introduce parallelism to the expansion and simulation stages of MCTS, leverage tree parallelization techniques [27]. The application of parallelism allows for decoupling the select and expand stages from the simulation stage in an MCTS iteration. The decoupling allows multiple MCTS iterations to be carried out simultaneously, limited only by the number of CPU cores. The standard UCB formula used in MCTS is updated as Eq. 2,

$$\underset{n' \in \text{children of } n}{\arg\max} \frac{Q(n')}{N(n') + \hat{N}(n')} + c\sqrt{\frac{2\ln\left(N(n) + \hat{N}(n)\right)}{N(n') + \hat{N}(n')}}, \tag{2}$$

where the idea of virtual loss [27] is applied by adding one extra virtual visit counts $\hat{N}$ that indicates a node has been selected but not yet simulated and backpropagated. Since the simulation and subsequent backpropagation stages are not yet completed, the tree search algorithm must be notified to update the Q and N of the node. This adjustment minimizes the likelihood of revisiting the node in the next iteration, implementing a conservative approach in anticipation of a potentially poor reward. Once the simulation stage has concluded, Q is updated in backpropagation with

returned reward from simulation result, N increments and $\hat{N}$ decrements.

D. Adaption MCTS for REMP

Given REMP's extremely large search space due to push actions' trajectory planning requirements, modifications are introduced to best apply MCTS to REMP.

Action Space Bias. The action space used in the simulation stage is a subset of that used in the expansion stage. Given that one iteration of the simulation stage constitutes a coarse estimation of action and state, reducing the number of actions leads to a larger number of total iterations.

Biased Simulation. A straightforward implementation of the simulation stage in MCTS is a random policy that indiscriminately selects an action for execution, ultimately obtaining a reward at the final state reached. In our implementation, we adopt a heuristic to guide the action selection in the simulation stage towards the ultimate goal of a given object. Specifically, with a probability of θ_{sim} (dynamically changed based on depth of the search), a random action is selected; otherwise, an attempt is made to select an action that will put an object on its goal pose. This introduces a bias in the simulation stage, which offsets the drawback of limited iterations due to the time-consuming nature of motion planning and collision checking. We note that we did not adopt a recent advancement in MCTS for long-horizon planning that injects a data-driven element to partially learn the reward, e.g., a neural network can be trained to evaluate the quality of an action-state pair [7], [17], [19].

Reward Shaping. We structure the reward to favor the goal state but without introducing undue bias. The reward function plays a critical role as it steers the tree search, composed of three components. Firstly, if the task is accomplished, with all objects placed at their goal poses, a reward of R_g is awarded. Secondly, if an object o_i is located at its goal pose, a reward r_o is given. The cumulative reward from all objects, denoted as R_o, is computed as $R_o = \sum_i r_{o_i}$. Lastly, the reward structure also takes into account the cost associated with the movement of objects. For the pick-n-place action ($\mathcal{A}_{pp}$), the cost corresponds to the Euclidean distance between the pick and place poses, with an additional fixed cost factored in. For the push action ($\mathcal{A}_{pt}$), the cost is determined by the Euclidean distance of the path, also supplemented by a fixed cost. Additionally, a base reward is computed $R_b = R_o(s_0)$ from initial state s_0, which is used to normalize the final reward during the search. For each iteration, a reward is returned by the simulation stage and is updated during the backpropagation stage.

$$R_i = \begin{cases} \max(0, R_g - \mathit{cost} - R_b), & \text{if } s_i \text{ is goal state} \\ \max(0, R_o(s_i) - \mathit{cost} - R_b), & \text{otherwise} \end{cases}$$

Traditionally, Q retains the average reward values derived from simulation results, providing a robust estimation for action over millions of iterations. However, in our case, we aimed to maintain the planning time within reasonable limits. Therefore, we introduced a priority queue to store simulation results, which serves as the Q value in the algorithm. For the purpose of calculation in Eq. 2, we only preserve the top k rewards, similarly in the previous work [17]. There is a possibility that during a simulation, a subsequent action may transition the state to one with lower rewards, thereby negating the benefits of a preceding beneficial action within that simulation. To limit the search time, we choose to return the maximum reward encountered at the intermediate steps during the simulation instead of the final reward. Therefore, the returned reward from a simulation is given by

$$\max(\beta \cdot \max_{i \in m-1}(R_i), R_m) \cdot \gamma^m,$$

where $\beta \in (0, 1]$ is a scaling parameter and m is the total steps used in the simulation. γ is the discount factor, encouraging the problem to be solved in the early stage if possible.

Due to limited space, we omit the pseudo-code of the parallel MCTS algorithm but note that all details for reproducing the algorithm have been fully specified. We call the resulting algorithm *parallel Monte Carlo tree search for multi-primitive rearrangement* or PMMR.

V. Experimental Evaluation

We evaluate HBFS and PMMR methods for REMP in simulated environments and on a real robot. Regardless of whether it is a simulation or a real robot experiment, both algorithms perform *percept-plan-act* loops until the task is solved or the budgeted time is over. All experiments were conducted on an Intel i9-10900K (10 CPU cores) desktop PC and run in Python. As a note, limited testing shows that using Intel i9-13900K (24 CPU cores) reduces the planning time by roughly half, demonstrating the effectiveness and scalability of employing parallelism (code in Python).

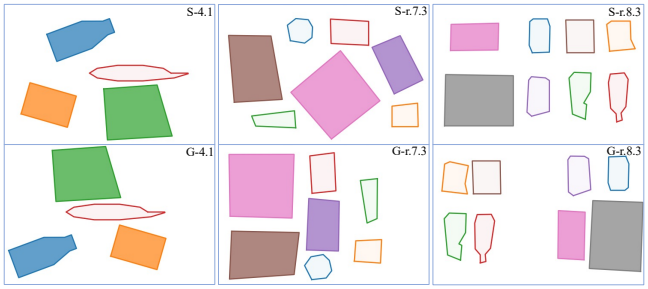

Fig. 3: Example cases. The top row shows the start states and the bottom goal states. Lightly shaded objects can be pick-n-placed; heavily shaded objects must be manipulated using push. Cases 4.1, r.7.3, and r.8.3 are evaluated and presented in Fig. 4. Objects are distinguished by color.

A. Simulation Studies

Simulation is conducted in PyBullet [28]. A real robot setup, consisting of a Universal Robot UR5e + OnRobot VGC-10 vacuum gripper, is replicated. The robot operates under end-effector position control; the workspace measures $0.78 \times 0.52\text{m}^2$. 25 feasible scenarios are created where all objects are confined within the workspace. Object sizes, shapes, and poses are randomly determined in each scenario (see Fig. 3 for some examples). Cases that are trivial to solve (e.g., objects that happen to be mostly small) are filtered. The number

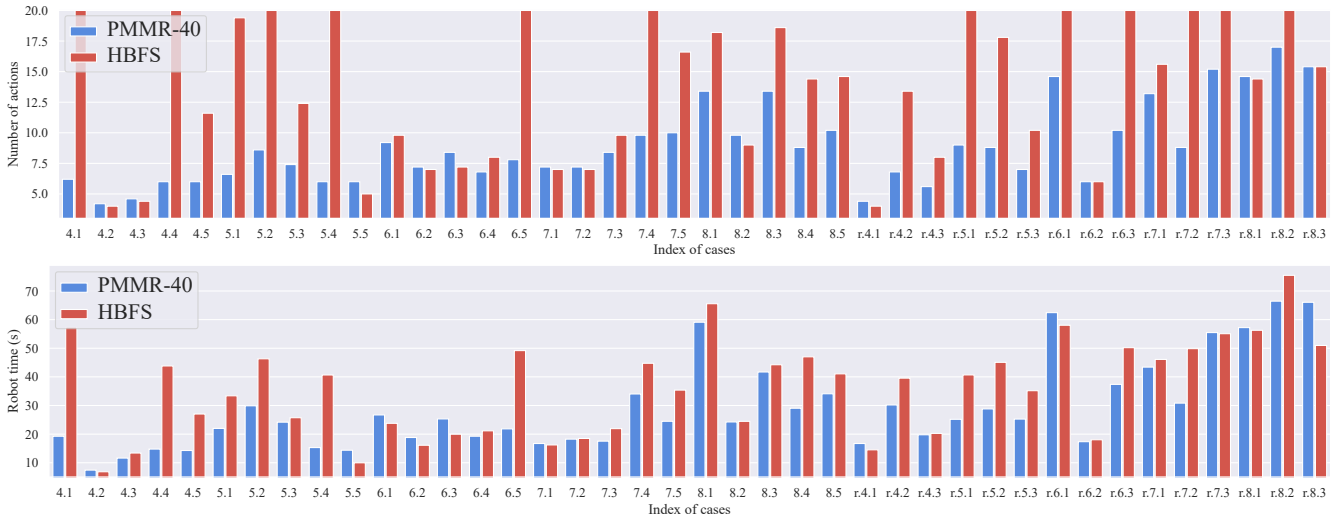

Fig. 4: As an expanded illustration of Tab. I, the upper plot lists the number of actions the robot executes to resolve individual cases. The lower plot lists the robot's execution times in solving the individual cases following the computed plan. For the labels on the horizontal axis, the first digit indicates the number of objects contained within each case, while the second digit represents the index of the cases. Cases beginning with the prefix 'r' are the ones that are constructed for and executed by the real robot setup.

of objects ranges from four to eight; five distinct cases are generated for each specified number of objects.

In evaluating PMMR, each *percept-plan-act* loop runs for a predetermined duration to identify the best next action until the problem is resolved or the maximum number of actions has been exhausted. If the latter occurs, it is treated as a failed case. We denote the corresponding PMMR method as PMMR-X, where X is the maximum number of seconds allowed in a single iteration of the loop. We settled on PMMR-40 as the main PMMR method used in the evaluation. In both simulation and real robot experiments, for PMMR-40, we keep the top $k = 100$ for Q value, $c = 1.5$ in Eq. 2. The max MCTS tree depth D is based on the number of objects $\mathcal{N}$: $D = 2\mathcal{N} + 2$. θ_{sim} is based on the depth d of the node $\theta_{sim} = \max(-0.106 + 0.231d - 0.013d^2, 0.2)$. These numbers are handpicked, representing that as the tree goes deeper, the probability of selecting a random action should be increased. $r_o = 0.7$ for object can be operated by $\mathcal{A}_{pp}$, the $R_g = 2r_0\mathcal{N}$. For an object that can be operated by $\mathcal{A}_{pt}$, the reward is given to $1.1r_o$. We set $\beta = 0.5$ and $\gamma = 0.9$ to scale the reward.

	Robot Time	Completion	Num. of Actions	Plan Time
PMMR-40	29.17s	98.00%	8.90	264.99s
HBFS	36.22s	54.50%	13.72	30.04s

TABLE I: Simulation experiment results for 25 simulated cases and 15 real robot experiments for HBFS and PMMR-40.

Individual experiment results for all cases are shown in Fig. 4 (which also includes cases used for real-robot experiments, to be detailed in Sec. V-C). Detailed experiment results are presented in Tab. I. Here, *robot time* refers to the cumulative time required for the robot to execute all actions, while *completion* refers to the success rate. The number of actions quantifies the execution of atomic actions, represented by $\mathcal{A}_{pp}, \mathcal{A}_{pt}$. The *plan time* is the total planning time. Each case underwent five independent trials.

Failure often happens because the case requires more than 15 actions to solve (even for humans). A failure may be due to sampled actions not containing a solution or the search not being deep enough. Sometimes, the algorithm can recover from an early bad choice, but not always since the number of iterations is capped. We observe that, while HBFS runs relatively fast in comparison to PMMR-40, it frequently fails (55% success v.s. 98% for PMMR-40) and uses many more actions (13.7 v.s. 8.9 for PMMR-40). Visually, as can be seen in the accompanying video, the actions generated by PMMR-40 are much more human-like than those by HBFS (same holds for real robot experiments).

B. Ablation Studies

We investigated the impact of time budgets, the depth of tree search, and the base reward R_b on solving REMP. The time budget is critical; an extended search duration tends to yield better results. However, it is necessary to balance planning time and solution quality. An insufficient search might sample highly suboptimal paths, leading to locally optimal actions. As depicted in Fig. 5 and Tab. II shows the correlation between planning time and solution quality, leading us to select PMMR-40 for our main evaluation.

A shallow MCTS (PMMR-40 ($D = 3$), max MCTS tree depth of 3) was included specifically to compare with HBFS, which has three "depth levels" per iteration.

In terms of reward design, as detailed in section IV, we introduced a base reward R_b, which serves to normalize the reward to 0 at root. Without this adjustment, the tree search might commence with a non-zero reward signal, where a disadvantageous branch may still return a reward, causing the search to frequently explore such branches. By comparing PMMR-40 (no-R_b) in Tab. II with PMMR-40 in Tab. I, we observe that the introduction of R_b aids the tree search.

	Robot Time	Completion	Num. of Actions	Plan Time
PMMR-10	35.60s	94.00%	10.51	103.61s
PMMR-20	32.46s	96.00%	9.86	155.68s
PMMR-60	27.93s	99.50%	8.53	358.44s
PMMR-40 ($D = 3$)	57.83s	32.50%	16.62	641.76s
PMMR-40 (no-R_b)	32.92s	94.00%	9.83	270.65s

TABLE II: Ablation studies for all 40 cases to be compared to Tab. I.

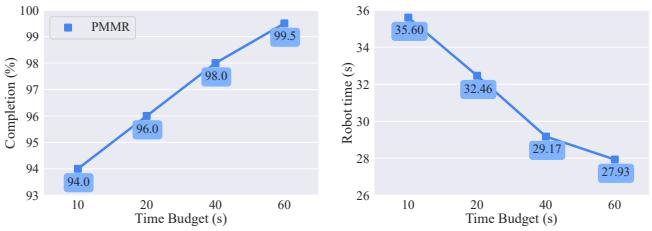

Fig. 5: PMMR is evaluated with different time budgets. The reported values are averaged over 40 cases.

C. Real Robot Experiments

In simulation, we emulate the vacuum function by attaching the object to the end-effector using an extra link via a fixed joint. We employ two vacuum cups to provide sufficient suction power to ensure a robust connection in the real-world setup. The point of suction on the object is taken as its center, assuming this central area is flat. Similar to simulation studies, the number of objects ranges from four to eight, and three distinct cases are generated for each number of objects.

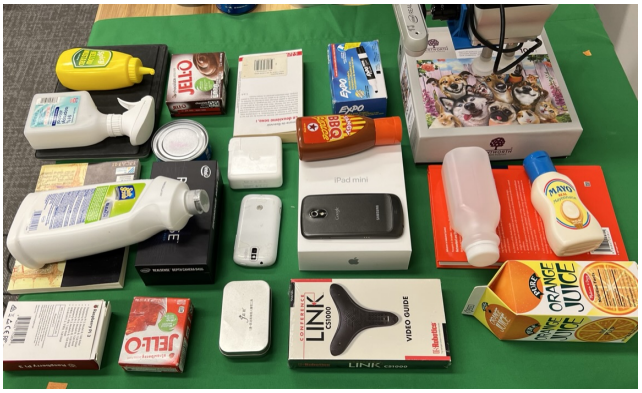

Fig. 6: The full set of objects used in our real robot experiments.

A RealSense D455 camera is affixed to the robot's wrist, capturing the scene from a top-down perspective, and an orthogonal view is rendered from the point cloud. We employ the Segment Anything Model [29] to extract masks of the objects present on the table. Subsequently, OpenCV [30] is applied to determine the contours and approximate them into polygons, which are then used for planning. An additional step determines each object's $SE(2)$ poses. For each case, the experiment is repeated at least three times.

Due to the small sim-to-real gap, We let the algorithm plan the entire sequence of manipulation actions at the beginning, which generally works well. If no solution is found using 20 actions, we mark it as a failure; otherwise, the robot executes the actions. Robot time is not recorded for failure cases, hence its absence in Tab. III. For completeness, in cases where both PMMR and HBFS succeed at least once, HBFS averages 15.15 actions and PMMR 9.56, with robot (execution) times of 96.62 seconds and 95.75 seconds, respectively (but, note that HBFS fails much more frequently). Results from individual benchmarks across 15 cases are presented in Fig. 7. Each case was subjected to three independent trials. In real robot experiments, the cases are intentionally designed to be challenging to solve. A greedy action may exacerbate the problem, making it even more difficult to resolve. Consequently, the completion rate of HBFS is significantly reduced.

	Robot Time	Completion	Num. of Actions	Plan Time
PMMR-40	95.75s*	96.44%	9.56	292.02s
HBFS	96.62s*	38.33%	15.15	29.05s
PMMR-40 (Sim)	–	94.67%	10.44	306.41s
HBFS (Sim)	–	45.33%	14.99	22.18s

TABLE III: Experiment results of real robot trials across 15 cases, with time budgets constrained to a maximum of 40 seconds for a single MCTS run. The robot time is only considered in cases where both methods succeed at least once. Additionally, benchmarks from simulations covering 15 cases are included for sim-to-real gap comparisons. The robot time for PMMR-40 and HBF, denoted with an asterisk, is recorded only for successful cases.

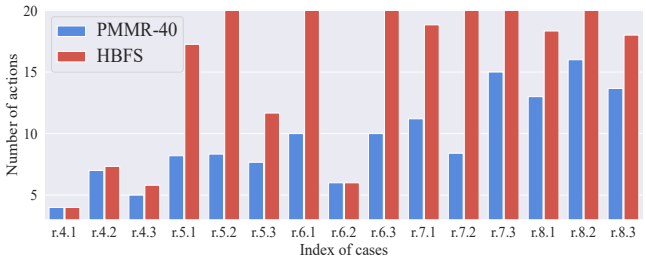

Fig. 7: As an expanded illustration of Tab. III, this plot illustrates the number of actions the robot executes to resolve individual cases.

VI. Conclusion and Future Directions

We make the observation that humans frequently solve manipulation challenges using multiple types of manipulation actions. In contrast, there has been relatively limited research tackling planning high-quality resolutions for long-horizon manipulation tasks exploring the synergy of multiple manipulation actions. Inspired by how humans solve everyday manipulation tasks, in this paper, we propose and study the *rearrangement with multiple manipulation primitives* (REMP) problem. Toward optimally solving REMP, we developed two effective methods, HBFS and PMMR, with PMMR especially adept at solving difficult REMP instances with high success rates and producing high-quality solution sequences, which are confirmed with thorough simulation and real robot experiments, going through the full percept-plan-act loops.

With the current work paving the way, in future studies, we plan to expand the research in two directions: (1) We will explore adding additional manipulation actions (e.g., pushing multiple objects, flipping objects) and solving more difficult rearrangement tasks (e.g., stacking to form complex structures), and (2) Leveraging the current algorithms to generate training data, we will develop data-driven methods (e.g., machine learning and/or reinforcement learning) to further speed up the planning process.

REFERENCES

[1] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*. Springer, 2006, pp. 72–83.

[2] M. T. Mason, "Toward robotic manipulation," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 1–28, 2018.

[3] K. Gao, D. Lau, B. Huang, K. E. Bekris, and J. Yu, "Fast high-quality tabletop rearrangement in bounded workspace," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1961–1967.

[4] R. Wang, Y. Miao, and K. E. Bekris, "Efficient and high-quality prehensile rearrangement in cluttered and confined spaces," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1968–1975.

[5] X. Zhang, S. Jain, B. Huang, M. Tomizuka, and D. Romeres, "Learning generalizable pivoting skills," *arXiv preprint arXiv:2305.02554*, 2023.

[6] X. Yi and N. Fazeli, "Precise object sliding with top contact via asymmetric dual limit surfaces," in *Robotics: Science and Systems, RSS*, 2023.

[7] F. Bai, F. Meng, J. Liu, J. Wang, and M. Q.-H. Meng, "Hierarchical policy with deep-reinforcement learning for nonprehensile multiobject rearrangement," *Biomimetic Intelligence and Robotics*, vol. 2, no. 3, p. 100047, 2022.

[8] Y. Hou, Z. Jia, A. M. Johnson, and M. T. Mason, "Robust planar dynamic pivoting by regulating inertial and grip forces," in *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*. Springer, 2020, pp. 464–479.

[9] N. Doshi, O. Taylor, and A. Rodriguez, "Manipulation of unknown objects via contact configuration regulation," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 2693–2699.

[10] N. Dengler, D. Großklaus, and M. Bennewitz, "Learning goal-oriented non-prehensile pushing in cluttered scenes," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 1116–1122.

[11] C. Pezzato, C. Salmi, M. Spahn, E. Trevisan, J. Alonso-Mora, and C. H. Corbato, "Sampling-based model predictive control leveraging parallelizable physics simulations," *arXiv preprint arXiv:2307.09105*, 2023.

[12] B. Tang and G. S. Sukhatme, "Selective object rearrangement in clutter," in *Conference on Robot Learning*. PMLR, 2023, pp. 1001–1010.

[13] K. Gao, S. W. Feng, and J. Yu, "On running buffer minimization for tabletop rearrangement," *17th Robotics: Science and Systems, RSS*, 2021.

[14] J. Lee, Y. Cho, C. Nam, J. Park, and C. Kim, "Efficient obstacle rearrangement for object manipulation tasks in cluttered environments," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 183–189.

[15] J. Ahn, C. Kim, and C. Nam, "Coordination of two robotic manipulators for object retrieval in clutter," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1039–1045.

[16] 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," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1775–1795, 2018.

[17] B. Huang, T. Guo, A. Boularias, and J. Yu, "Interleaving monte carlo tree search and self-supervised learning for object retrieval in clutter," in *International Conference on Robotics and Automation*, 2022.

[18] X. Lou, H. Yu, R. Worobel, Y. Yang, and C. Choi, "Adversarial object rearrangement in constrained environments with heterogeneous graph neural networks," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 1008–1015.

[19] H. Song, J. A. Haustein, W. Yuan, K. Hang, M. Y. Wang, D. Kragic, and J. A. Stork, "Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 9433–9440.

[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[21] Y. Labbé, S. Zagoruyko, I. Kalevatykh, I. Laptev, J. Carpentier, M. Aubry, and J. Sivic, "Monte-carlo tree search for efficient visually guided rearrangement planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3715–3722, 2020.

[22] B. Huang, A. Boularias, and J. Yu, "Parallel monte carlo tree search with batched rigid-body simulations for speeding up long-horizon episodic robot planning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[23] K. Gao, J. Yu, T. S. Punjabi, and J. Yu, "Effectively rearranging heterogeneous objects on cluttered tabletops," *arXiv preprint arXiv:2306.14240*, 2023.

[24] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

[25] J. Kuffner and S. 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.

[26] R. Bohlin and L. E. Kavraki, "A randomized approach to robot path planning based on lazy evaluation," *COMBINATORIAL OPTIMIZATION-DORDRECHT-*, vol. 9, no. 1, pp. 221–249, 2001.

[27] G. M. B. Chaslot, M. H. Winands, and H. J. van Den Herik, "Parallel monte-carlo tree search," in *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008. Proceedings 6*. Springer, 2008, pp. 60–71.

[28] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[29] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," *arXiv:2304.02643*, 2023.

[30] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.