Coordination of multiple mobile manipulators for ordered sorting of cluttered objects

Jeeho Ahn1,2, Seabin Lee1, and Changjoo Nam1,*

Abstract—We present a coordination method for multiple mobile manipulators to sort objects in clutter. We consider the object rearrangement problem in which the objects must be sorted into different groups in a particular order. In clutter, the order constraints could not be easily satisfied since some objects occlude other objects so the occluded ones are not directly accessible to the robots. Those objects occluding others need to be moved more than once to make the occluded objects accessible. Such rearrangement problems fall into the class of nonmonotone rearrangement problems which are computationally intractable. While the nonmonotone problems with order constraints are harder, involving with multiple robots requires another computation for task allocation.

In this work, we aim to develop a fast method, albeit sub-optimally, for the multi-robot coordination for ordered sorting in clutter. The proposed method finds a sequence of objects to be sorted using a search such that the order constraint in each group is satisfied. The search can solve nonmonotone instances that require temporal relocation of some objects to access the next object to be sorted. Once a complete sorting sequence is found, the objects in the sequence are assigned to multiple mobile manipulators using a greedy task allocation method. We develop four versions of the method with different search strategies. In the experiments, we show that our method can find a sorting sequence quickly (e.g., 4.6 sec with 20 objects sorted into five groups) even though the solved instances include hard nonmonotone ones. The extensive tests and the experiments in simulation show the ability of the method to solve the real-world sorting problem using multiple mobile manipulators.

I. INTRODUCTION

Mobile manipulators have been receiving much attention due to their ability to move and manipulate simultaneously. We see many promising applications such as logistics, manufacturing, and services. There even have been several products ready to be commercialized like Stretch from Boston Dynamics [1] and another Stretch from Hello Robot [2].

When we are to manipulate multiple objects, it might be essential to consider what to manipulate first. If we unload a truck or de-palletize, the objects (e.g., boxes, totes) to be moved would have a specific order. In general, it is a common practice not to pile up heavy containers on top of small, lighter ones for safety. In a fulfillment center or manufacturing process, products might need to be sent to the next destination in a specified order to meet process requirements. Some items should be delivered in a first attempt in the literature. The method can satisfy the order constraint and handle both the monotone and nonmonotone instances of the problem.

We propose a method to solve the ordered sorting problem in clutter using a team of robots. In a dense clutter, some objects need to be relocated more than once (i.e., nonmonotone problem [3]) if the next object to be sorted is blocked by other objects. This class of problem is more challenging than the multi-robot object rearrangement problem, which is known to be $NP$-hard [4] even without the order and group specifications.

We aim to solve the ordered sorting problem in clutter using a team of robots. In a dense clutter, some objects need to be relocated more than once (i.e., nonmonotone problem [3]) if the next object to be sorted is blocked by other objects. This class of problem is more challenging than the multi-robot object rearrangement problem, which is known to be $NP$-hard [4] even without the order and group specifications.

We propose a method to solve the ordered sorting problem in clutter using a team of robots, which is the first attempt in the literature. The method can satisfy the order constraint and handle both the monotone and nonmonotone instances of the problem.

The following are the contribution of this work:

- We propose a method to solve the ordered sorting problem in clutter using a team of robots, which is the first attempt in the literature. The method can satisfy the order constraint and handle both the monotone and nonmonotone instances of the problem.
- We provide formal analyses of the proposed method such as completeness, optimality, and time complexity.
- We show extensive test results and comparisons of the method. We also test the method using multiple mobile manipulators in dynamic simulation to see how the method works effectively with physical robots.
II. RELATED WORK

We study the problem of sorting objects in clutter using a team of robots where the order of sorting objects must be satisfied. There have been a line of research on multi-robot object manipulation, yet none of them directly tackles the same problem that we are interested in.

The problem of coordinating multiple robots to transport a common, usually large, object has been solved by various approaches [5]–[7] such as path planning and formation control. While these works consider object manipulation using coordinated mobile robots, they focus on transporting a large object together but not sorting objects.

There are some studies on multi-robot sorting [8], [9], [8] proposes a trajectory planning method for two robots using a common tool (i.e., stick or string) to sort objects into two groups. [9] also proposes a method to generate a path using a Voronoi diagram where two robots tie to a rope follow the path to separate objects. These works focus on the use of a tool for coordinated sorting without any consideration on the order of sorting. Moreover, they assume that all objects are directly accessible so no nonmonotone instance occurs.

One of the closely related work to ours is [10], which proposes a method to remove multiple objects in a bounded space. The robots have to navigate through a narrow entrance. This work aims to set an order of removal from possible accessibility dependencies between objects in order for them to be reachable, or at least, easier to be reached. Although the method finds a sequence of object removal efficiently even in clutter, it neither assumes removal priorities of objects nor grouping requirements, so it is less constrained.

Another close work is [11], which considers object rearrangement within a confined space. The proposed method finds a path of a single robot to rearrange (less than ten) objects to achieve a final configuration of the objects. The problem does not directly concern the ordered sorting. Nevertheless, the method can be used for our sorting problem if the final configuration requires the objects to be grouped in a specific order. However, the method generates a path of a single robot, which is not straightforward to distribute the single-robot path to multiple robots.

Despite these previous efforts, we are not able to find a work that considers the same problem as ours: coordination of multiple robots to sort multiple objects in clutter (with possible occlusions) by given types and priorities. Thus, we aim to fill the gap by proposing an efficient method for rearrangement planning and multi-robot task allocation.

III. PROBLEM DESCRIPTION

A. Assumptions

We have several assumptions to focus on the ordered sorting of cluttered objects: (i) One robot can manipulate one object at a time, and an object needs only one robot to manipulate it. (ii) At least one object must be accessible to one of the robots. (iii) The robots do not use nonprehensile actions. (iv) The environment is known (e.g., geometry of objects, where to sort and temporarily place objects). (v) The number of robots is far lower than the number of objects.

(b) A nonmonotone instance

Fig. 2: Examples of problem instances. (a) Seven objects are sorted into three groups \( G_1, \ldots, G_3 \). The objects have specific orders to be sorted. (b) In dense clutter, robots would not access all objects directly. The object that must be sorted in the beginning (i.e., \( r_1, g_1, b_1 \)) are occluded.

B. Problem formulation

In a workspace \( \mathcal{W} \), the goal is to sort \( N \) objects in \( \mathcal{O} \) into \( K \) groups using a team of \( M \) robots in \( \mathcal{R} \). The objects have a specific order to be sorted. A group \( G_i \), for \( i = 1, \ldots, K \), is a tuple of objects where \( |G_i| > 0 \) and \( \sum_{i=1}^{K} |G_i| = |\mathcal{O}| = N \). Throughout the paper, we will use different object colors, such as red, green, blue, and so on, to better distinguish objects in different groups. Without loss of generality, we assume that the indices of objects in the groups are in the ascending order. In Fig. 2a, we have \( G_1 = (r_1, r_2) \), \( G_2 = (g_1, g_2, g_3) \), and \( G_3 = (b_1, b_2) \). In \( G_1 \), \( r_1 \) must be sorted before \( r_2 \). Without occlusions between objects, sorting an object belonging to one group is independent from other groups. On the other hand, the objects in different groups would be interrelated if an object in one group is blocked by other objects in different groups.

The problem consists of two challenging subproblems. The first subproblem is the ordered sorting problem in which occlusions between objects could prevent robots from complying with the order constraint. This subproblem mandates solving the manipulation planning among movable obstacles (MAMO) problem, which is proven to be \( NP \)-hard [3]. Furthermore, our problem needs to consider the order constraint. The second subproblem is the multi-robot task allocation (MRTA), determining which robots sort which objects. Since we assume \( M < N \), the allocation should consider an extended time frame, which falls into the strongly \( NP \)-hard ST-SR-TA problem (Single-Task robots, Single-Robot tasks, Time-extended Assignment) according to the MRTA taxonomy [12]. As the entire problem is computationally intractable, developing an optimal method is not practically useful. Thus, we solve the subproblems for faster planning and the sorting task separately.

We aim to find a sequence of objects to be sorted, or a tuple \( \mathcal{O}_S \), with the minimum length. Then we repeatedly assign the robots to the objects in \( \mathcal{O}_S \) until all objects are sorted. In the example in Fig. 2a, a valid solution is \( \mathcal{O}_S = (r_1, g_1, r_2, g_2, b_1, g_3, b_2) \). Another solution \( \mathcal{O}_S = (g_1, g_2, g_3, r_1, r_2, b_1, b_2) \) is also valid. Both sequences result in the stacks shown in Fig. 2a. In these monotone instances,
\(|O_S| = N\). If some objects are not accessible unless the occluding objects are relocated, the problem instance is nonmonotone and hence \(|O_S| > N\). As shown in Fig. 2b, none of the first objects in \(G_1, \ldots, G_3\) is accessible, so one or more of the occluding objects should be relocated. Even in this nonmonotone case, we want to minimize \(|O_S|\) by relocating only the necessary objects.

### IV. METHODS

We propose our method that solves the sorting problem using search and a greedy MRTA method.

#### A. Search-based planning

We develop four versions of the planning method that are based on uninformed search (Breadth-First and Depth-First Search) and informed search (Best-First and \(A^*\) Search). They share most procedures shown in Alg. 1. They have different implementations for the frontier (lines 1–7) and evaluation functions (lines 39–43). The priority queue used in the informed search enqueues the element with the minimum value. Other specific implementations of functions are explained in Sec. IV-A.2.

In Alg. 1, each node in the search tree represents an object configuration (i.e., the status of \(O\) and \(G_i\)). The root represents the initial state configuration with empty depots. The root is inserted into the frontier (line 13), the while loop (lines 14–45) iterates until all objects are sorted (line 17) or no object can be manipulated anymore (line 21).1

In the while loop, line 15 chooses the node to expand, which varies depending on the search strategy. The node chosen to expand represents a particular object configuration where some objects are sorted or temporarily sent to arbitrarily chosen free spaces that we refer as buffer locations. Among accessible objects (line 19), line 24 finds the objects that can be sorted while complying with the order constraint. Suppose that the accessible objects \(O_A = \{g_1, g_3, b_3\}\), the sets of objects in the depots \(G_1 = \{r_1\}\), \(G_2 = \emptyset\), and \(G_3 = \{b_1, b_2\}\). Then the objects to be sorted next are \(O_G = \{r_2, g_1, b_1\}\). Thus, the next objects to allocate to robots are \(O_N = O_A \cap O_G = \{g_1, b_1\}\).

If \(O_N = \emptyset\), all objects in \(O_G\) are not accessible owing to occlusion (so nonmonotone). At least one object in \(O_G\) needs to become accessible to continue. Thus, lines 25–33 find the minimum number of objects (line 28) to be relocated to buffers temporarily. While they are waiting in the buffers, they are included in \(O_A\) so can be sorted in their turn.

If \(O_N \neq \emptyset\) or \(O_N\) becomes nonempty after relocating occluding objects, all objects in \(O_N\) are sorted so one or more depots are filled (line 36). After sorting, lines 38–44 add a new node to the frontier. Once a goal node is found, \(O_S\) contains all objects to be sorted. In nonmonotone instances, it contains some objects sent to buffers temporarily. If the search finishes without returning Success, the complete sorting sequence cannot be found (line 46).2

1) **Heuristics for informed search:** Let \(f(v), h(v),\) and \(g(v)\) be the evaluation, heuristic, and path cost function for node \(v\), respectively. Since our goal is to minimize \(O_S\), \(h(v)\) is the number of unsorted objects that are not yet in \(G\). In other words, \(h(v) = N_R = N - \sum_{i=1}^{K} |G_i|\). If monotone, the robots need to manipulate only the remaining \(N_R\) objects. Otherwise, if nonmonotone, the robots manipulate more than \(N_R\) objects since some objects are sent to buffers. The actual number of manipulated objects is always less than or equal to \(h(v)\), which is an admissible heuristic that never overestimates the number of manipulated objects. The path cost function \(g(v)\) is simply the number of manipulated objects from the root to node \(v\).

Due to the order constraint, a robot transporting an object to a depot should wait until a currently working robot at

---

**Algorithm 1** **SORTOBJ**

**Input:** Objects \(O\), \(K\), robots \(R\), workspace \(W\), method

**Output:** Result

1: if method = BFS then
2: Initialize a queue \(F\) as the frontier
3: else if method = DFS then
4: Initialize a stack \(F\) as the frontier
5: else if method = BestFirst or method = AStar then
6: Initialize a minimum priority queue \(F\) as the frontier
7: end if
8: Initialize \(K\) stacks: \(G_1, \ldots, G_K\)
9: Let \(v\) be the node representing the initial configuration of \(O\)
10: if method = BestFirst then
11: \(v\).value = 0
12: end if
13: F.QUEUE(v) // Implemented as PUSH for DFS
14: while \(F\) is not empty do
15: \(v = F\).DEQUEUE() // Implemented as POP for DFS
16: if \(\sum_{i=1}^{K} |G_i| = |O|\) then
17: return Success
18: end if
19: \(O_A = \) GETACCESSIBLEOBS\((O, R, W)\) // Get accessible objects from robots
20: if \(O_A = \emptyset\) then
21: return Infeasible // No object can be manipulated
22: end if
23: \(O_G = \) GETNEXTOBS\((G_1, \ldots, G_K)\) // Get the set of the objects to be sorted next
24: \(O_N = O_A \cap O_G\)
25: if \(O_N = \emptyset\) then
26: for each object \(o\) in \(O_G\) do
27: \(O_R = \) RELOCOBS\((O, o, R, W)\) . . . . . . // Find objects to be relocated to make \(o\) accessible (Sec. IV-A.2)
28: Keep \(O_R\) shortest . . . . // Send the least objects to buffer
29: end for
30: Relocate objects in \(O_R\) to buffers . . . . . . . . // Nonmonotone
31: \(O_A = \) GETACCESSIBLEOBS\((O, R, W)\)
32: \(O_N = O_A \cap O_G\)
33: end if
34: for each object \(o\) in \(O_N\) do
35: Get the group index \(i\) of \(o\)
36: \(G_i\).PUSH\((o)\) . . . . . . // Sort \(o\) to its group
37: end for
38: Let \(v\) be the node representing the current config. of \(O\)
39: if method = BestFirst then
40: \(v\).value = \(h(v)\) . . . . . . . // Detailed in Sec. IV-A.1
41: else if method = AStar then
42: \(v\).value = \(h(v) + g(v)\) . . . . // Detailed in Sec. IV-A.1
43: end if
44: F.QUEUE\((v)\)
45: end while
46: return Failure . . . . . . . . . // Search fails if the frontier is empty

---

1In Sec. III-A, we assume that at least one object can be manipulated. Thus, the instances that we consider do not incur the latter infeasible case.

2In Sec. IV-C, we show that Alg. 1 is complete so never returns Failure (Theorem 4.2).
the depot leaves. Frequent waiting decreases the throughput of sorting. Also, there may be higher chances of crashes or deadlocks if many robots wait at a depot. We deal with this problem by defining a secondary criterion for the search for the nodes with the same $f(v)$. We penalize $f(v)$ if objects belonging to the same group are in the sorting sequence consecutively. Given $M$ robots, we consider the most recent $M$ objects in the current $O_S$ to compute the penalty. The window size could vary depending on the environment.

2) **Utilities**: Function $\text{GetAccessibleObjs}$ finds the objects that are accessible to robots. We employ a polynomial-time method proposed in [13] to determine the accessibility of an object without performing motion planning. The method needs to know only the sizes of an object and the end-effector to find a collision-free direction that the end-effector can approach to grasp the object. Since this method does not consider the kinematic constraints of the whole robot, an object that is determined to be accessible could turn out to be inaccessible in motion planning. Nevertheless, it dramatically decreases the planning time by removing the computation-intensive motion planning as demonstrated in [14]–[16].

Next, $\text{GetNextObjs}$ finds the next objects to be sorted in each group $G_1, \ldots, G_K$. The input is the groups implemented by stacks. Top operation for all $K$ groups tells the lastly sorted objects in each group. Then the next object to be sorted in each group can be found. Last, $\text{ReLocObjs}$ finds the objects to be relocated to make an inaccessible object ($o$ in the input) accessible. If the objects to be sorted found by $\text{GetNextObjs}$ are not accessible, this function tells what to relocate to buffers. We employ a method proposed in [15] that runs in polynomial time in the number of objects.

B. Greedy task allocation

With $N > M$, an assignment between $O_S$ and $R$ can be computed by repeatedly assigning a few objects to a few idle robots in a batch. However, this batch allocation requires some robots to wait until a sufficient number of robots become idle. Thus, we implement an online greedy task allocation algorithm assigning each object in $O_S$ to each idle robot that can access the object. If more than one robot can access the object, the closest robot is assigned.

C. Analysis of the algorithms

**Lemma 4.1.** The heuristics $h(v)$ in Alg. 1 is admissible.

We omit the proof due to the space limit, but the lemma is already proved informally in Sec. IV-A.1.

**Theorem 4.2.** All versions of Alg. 1 are complete.

**Proof.** BFS is known to be complete. $A^*$ Search is complete with an admissible heuristic.³ DFS and Best-First Search are complete if the state space is finite. In our sorting problem, the flow of objects is one-way: from the clutter to depots directly or via buffers. They never move back to the clutter from the depots or buffers. Sorted objects in the depots do not move. Objects in buffers are accessible, so they do not move inside the buffers. Thus, our problem is with a finite state space. □

**Theorem 4.3.** Alg. 1 with BFS and $A^*$ are optimal.

**Proof.** BFS is optimal if the path cost is a non-decreasing function of the depth of the node [17]. As described in Sec. IV-A.1, $g(v)$ is the number of manipulated objects until $v$, which must increase as the search depth increases. $A^*$ is optimal with an admissible heuristic. Thus, both versions of Alg. 1 are optimal in the length of $O_S$. □

**Proposition 4.4.** Each generation of a search node in Alg. 1 runs in polynomial time.

**Proof.** All the search methods run in exponential time in the depth of the search tree $O(b^d)$ where $b$ is the largest branching factor and $d$ is the depth. In the worst case, both quantities can be up to $N$. Although the entire search could take exponential time, each node generation is done efficiently (lines 14–45). Our implementation of $\text{GetAccessibleObjs}$ has $O(N^3)$ time complexity. Inside $\text{GetAccessibleObjs}$, the accessibility check for each object takes $O(N^2)$ [13], which repeats up to $N$ times. $\text{GetNextObjs}$ runs the constant-time top operation of stacks $K$ times. Thus, the function runs in $O(K)$. The most excessive computation could occur in lines 26–29, where $\text{ReLocObjs}$ repeats $|O_C|$ times. The quantity $|O_C|$ is bounded by the number of groups $K$ as $\text{GetNextObjs}$ finds one object to be sorted in each group. In [15], $\text{ReLocObjs}$ is shown to be with $O(N^4)$. Thus, the time complexity of the entire for loop is $O(N^4K)$, which dominates all the operations in the while loop. □

Even though the algorithms do not have polynomial time complexity, DFS and Best-First Search find a solution quickly, which will be shown in experiments (Sec. V). We also show that the solution quality of DFS and Best-First are the same as the optimal solution from $A^*$ even though the two methods are not guaranteed to find an optimal solution.

V. EXPERIMENTS

The first set of experiments evaluates the performance of Alg. 1 with a vast amount of random instances. Next, we run another set in a dynamic simulation using the Unity-ROS integration (Fig. 3) [18]. All experiments are done with the system with AMD 5800X 3.8GHz/32G RAM, Python 3.8.

A. Algorithm tests

We increase $N$ from 10 to 30 at intervals of 5 and have 1, 3, and 5 for $K$. For each $(N, K)$, we generate 20 instances where object locations are sampled uniformly at random. The number of objects that belong to the same group is randomly sampled while zero object in a group is not allowed. The object shape is disc whose diameter is randomly sampled around 30 cm. Then, we use the same set of 300 instances for testing the four versions for a fair comparison. Some instances are visualized using the simulator (Fig. 3).

We measure the objective value, the length of $O_S$, which indicates the number of objects that the robots should manipulate until the end. Another important metric is the task planning time measuring how long Alg. 1 takes to generate $O_S$. We do not have the notion of robots in this algorithm.

³All the well-known proofs are given in [17].
Fig. 3: Example scenes of the experiments in simulation for $K = 5$ (objects with five colors) and varying $N$. The left picture shows the mobile manipulators (UR5 and Jackal) that manipulate objects using a suction gripper.

| $N$ | 10 | 15 | 20 | 25 | 30 |
|-----|----|----|----|----|----|
| $K$ | 1  | 2  | 3  | 4  | 5  |
| 10  | 0.5 | 0.1 | 0  | 0  | 0  |
| 15  | 0.8 | 0.2 | 0  | 0  | 0  |
| 20  | 1.0 | 0.3 | 0.05 | 0  | 0  |
| 25  | 0.95 | 0.3 | 0.15 | 0  | 0  |
| 30  | 0.95 | 0.5 | 0  | 0  | 0  |

(a) Ratio of nonmonotone instances

(b) The mean length of $O_S$

test. Thus, the task planning time does not include the time for motion planning or task allocation.

Other metrics include the success rate of task planning given a time limit (30 sec). Although the proposed algorithm is shown to be complete, we want to achieve fast planning. Thus, we set the tight limit even for the most challenging instance with $(N, K) = (30, 1)$. We also calculate the repetitiveness of a sorting sequence $O_S$ to see the effect of adopting the secondary criterion for breaking ties (presented in Sec. IV-A.1). This metric measures how many objects belonging to the same group are listed in $O_S$ in a row. A higher value means robots often use the same depot in series.

Although having only one group ($K = 1$) makes the problem merely stacking objects but not sorting, we test it since the problem becomes more difficult. The function GETNEXTOBJS in line 23 of Alg. 1 gives at most $|O_G| = K$ objects. The chance to have fewer objects in $O_N = O_A \cap O_G$ increases with a smaller $K$. A larger $N$ incurs more occlusions between objects so $|O_A|$ is likely to decrease, resulting in small $|O_N|$. If $O_N = \emptyset$, we need to solve the nonmonotone problem. As an example, we show the ratio of nonmonotone instances out of 20 random instances in Table Ia. As $K$ decreases and $N$ increases, a significant portion of test instances fall to the nonmonotone class. We show the length of $O_S$ in Table Ib. Since $N = 10$ is the only case where all the four versions can finish without excessive running time, we show the result for $N = 10$. As expected, the length $|O_S|$ increases as $K$ decreases. This can be justified by the fact that a small $K$ leads to more nonmonotone instances appear, which require additional moves to buffers.

Interestingly, all four versions have the same $|O_S|$ if they run the same instance. Fig. 4) shows two complete search trees for a toy monotone and a nonmonotone instance in the gray boxes. They have inaccessible objects denoted by dotted outlines. With a slight abuse of notation, we inscribe the object to be sorted in the search node except for the root node and the node where an object is sent to buffers. All leaf nodes have the same depth, which all represent the goal states. Depending on the search strategy, one of the leaf nodes is reached via possibly different paths. In any case, sorting is completed with the minimum $|O_S|$, which is possible owing to the way Alg. 1 generates nodes. In each iteration (lines 14–45), objects are manipulated only necessarily resulting in $|O_S| = N$ obviously in the monotone case. In the nonmonotone case, some objects should be sent to buffers. In Fig. 4b, we omit part of the tree owing to the space limit but the subtrees from the children of the root have the same height. However, some instances would have situations where one or more of the branches have different numbers of objects sent to buffers so the depths of the leaf nodes are not identical. Nevertheless, the four versions produce the same $|O_S|$ for the same instance in our tests. Probably many instances have the trees like those shown in Fig. 4. Otherwise, all search methods seem to reach the goal node at the shallowest level.

The computation time of the four versions is shown in Fig. 5 and Table II. BFS and A* show prohibitively long time even for moderately-sized instances so we do not test them beyond $N = 10$ so show the result only in Table Ia. Best-First and DFS are the fastest. As discussed above, any leaf node in the tree is a goal node. DFS directly goes down deep so reaches the leftmost leaf quickly. The heuristic function of Best-First Search is designed to expand the node with a
Fig. 5: The computation time of Alg. 1 to generate a sorting sequence. Since BFS and A* are prohibitively slow with N > 10, we only show the results of two fast versions of Alg. 1.

**TABLE II**: The average running time and standard deviation of Alg. 1 in seconds from 20 random instances for each (N, K). BFS and A* run slow with large N so the result with N = 10 is shown.

| Method      | N | 1   | 3   | 5   | 10  |
|-------------|---|-----|-----|-----|-----|
| Mean        |   | 0.8498 | 0.6608 | 0.5957 | 0.8419 | 0.6329 | 0.5714 |
| SD          |   | 0.1349 | 0.1234 | 0.0130 | 0.3112 | 0.1481 | 0.0131 |
| Mean        |   | 1.3131 | 0.9814 | 0.0131 | 1.2942 | 0.9195 | 0.0134 |
| SD          |   | 0.3183 | 0.2401 | 0.4822 | 11.0923 | 5.3343 | 4.3813 |
| Mean        |   | 19.7516 | 19.1491 | 8.4924 | 19.3819 | 10.1660 | 3.8581 |
| SD          |   | 7.5329 | 3.9095 | 1.4130 | 7.5891 | 3.7654 | 1.4150 |
| Mean        |   | 21.8812 | 19.1176 | 12.1503 | 24.0828 | 18.8628 | 13.3311 |
| SD          |   | 13.0826 | 6.4647 | 2.2581 | 12.8005 | 6.6927 | 3.0683 |

(a) Best-First and Depth-First Search

(b) Breadth-First and A* Search

Table: Lesser number of unsorted objects. Thus, it is likely to choose the node which is at the greater depth rather than the nodes at the same level. Since both tend to go down in the tree, an increase in the branching factor (closely related to K) does not significantly affect the search time. Rather, an increase of K allows more objects in O_S so causes a less number of nonmonotone instances (Table Ia). Thus, the running time decreases as K increases. It is clear that the running time increases with larger N as shown in Fig. 5.

Since BFS explores all nodes at the current depth, a higher branching factor (with a larger K) increases the running time significantly (Table IIb). In A*, the evaluation function adds the number of unsorted objects (h(v)) and the number of sorted objects so far. For those trees with leaves with an identical depth, h(v) is the same for all nodes in the same level. Thus, the search behaves like BFS. Nevertheless, A* is faster because it can explore a deeper level if some leaf nodes have smaller depths so the search goes down.

The success rate is summarized in Table III. As expected, Best-First and DFS show higher success rates even with the tight time limit (30 sec). As discussed, the instances with K = 1 do not represent the sorting problem so we sidelong them from discussion. With K > 1, the search finishes quickly in the most of the instances. With a slightly longer time limit (i.e., 32 sec), the success rates achieve 100% for all instances with K > 1. BFS and A* are not adequate for those applications where a lack of responsiveness causes catastrophic failures.

We compare the repetitiveness of the four versions for N = 10. We do not consider K = 1 since O_S is always filled with the same type of objects. Generally, Best-First and A* show lower values because they adopt the secondary cost to penalize consecutive objects belonging to the same group. With a larger K, it is natural to have more diversity in O_S. From this result, we can expect that the execution time of Best-First and A* can be shorter than that of BFS and DFS because the depots are less congested, which will be confirmed in Sec. V-B.

**B. Experiments in dynamic simulation**

We test Best-First and DFS which are shown to be fast enough for practical applications. Since extensive tests are already done in Sec. V-A, we pick a subset of the test sets for this experiments in simulation. From the previously generated sets, we choose the sets with N = 10, 20, 30 and K = 3, 5 resulting in 120 instances in total. We modify the diameters of the disc objects such that the object with higher priority is the largest (30 cm). This modification is to visually show the precedence constraint where a smaller object should be stacked on top of larger ones. We have three mobile manipulators (M = 3) as shown in Fig. 3, which is in line with our assumption of N > M. The number of robots could vary but we fix it to limit the number of control variables, without loss of generality.

Initially, K depots are initialized. The depots and robots are located on the periphery of a clutter of objects. Once Alg. 1 finds O_S, the robots start manipulating objects. The centralized task allocator assigns the object at the head of O_S to an idle robot. Each robot has its own global path planner and local controller for collision-free navigation and motion planner for manipulation. The robot picks and places an

---

4 The space limit precludes inserting large screenshots but the video clip at https://youtu.be/oMBNWeFYGDw should help.
TABLE IV: The average task planning, motion planning, and execution time measured in the experiments in dynamic simulation

| Measure (sec) | Method          | Task planning time | Motion planning time | Execution time |
|---------------|-----------------|--------------------|----------------------|----------------|
|               | Best-First      | 3                  | 5                    | 3              |
| N=10          | Mean            | 0.7252             | 0.6468               | 0.7192         |
|               | SD              | 0.0303             | 0.0506               | 0.0444         |
|               | Median          | 0.6992             | 0.6483               | 0.7187         |
| N=20          | Mean            | 3.0579             | 4.9592               | 4.8709         |
|               | SD              | 0.0194             | 0.1089               | 0.1351         |
|               | Median          | 0.1316             | 0.1021               | 0.1053         |

Fig. 7: The total planning time of Best-First Search

VI. CONCLUSION

We consider the problem of coordinating multiple mobile manipulators to sort objects in a particular order while the order cannot be satisfied unless some objects are rearranged to access the objects to be sorted. To solve this computationally intractable problem, we present a search-based method with four different search strategies. They are shown to be complete where two of them are shown to produce solutions fast enough. We demonstrate our methods handle both monotone and nonmonotone rearrangement instances while securing high success rate by interactive seeking for feasible tasks through task and motion planning method. With this work as a baseline, we would develop further by generalizing object geometry. Also, we seek to extend our work to be used in other applications.

REFERENCES

[1] B. Dynamics, “Stretch,” https://www.bostondynamics.com/products/stretch, [Online].
[2] H. Robot, “Stretch,” https://hello-robot.com/product, [Online].
[3] M. Stillman and J. Kuffner, “Planning among movable obstacles with artificial constraints,” The Int. J. of Robotics Research, vol. 27, pp. 1295–1307, 2008.
[4] J. Ahn, C. Kim, and C. Nam, “Coordination of two robotic manipulators for object retrieval in clutter,” in Proc. of Int. Conf. on Robotics and Automation (ICRA). IEEE, 2022, pp. 1039–1045.
[5] Z. Feng, G. Hu, Y. Sun, and J. Soon, “An overview of collaborative robotic manipulation in multi-robot systems,” Annual Reviews in Control, vol. 49, pp. 113–127, 2020.
[6] A. Yamashita, T. Ara, J. Ota, and H. Asama, “Motion planning of multiple mobile robots for cooperative manipulation and transportation,” IEEE Trans. on Robotics and Automation, vol. 19, pp. 223–237, 2003.
[7] I. Mas and C. Kitts, “Object manipulation using cooperative mobile multi-robot systems,” in Proc. of World Congress on Engineering and Computer Science, 2012.
[8] A. Yamashita, J. Sasaki, J. Ota, and T. Ara, “Cooperative manipulation of objects by multiple mobile robots with tools,” in Proc. of Japan-France/2nd Asia-Europe Congress on Mechatronics, vol. 310, 1998, p. 315.
[9] T. Maneewarn and P. Riniprawat, “Sorting objects by multiple robots using voronoi separation and fuzzy control,” in Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), vol. 2, 2003, pp. 2043–2048.
[10] W. N. Tang, S. D. Han, and J. Yu, “Computing high-quality clutter removal solutions for multiple robots,” in Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2020, pp. 7963–7970.
[11] J. Ota, “Rearrangement planning of multiple movable objects by a mobile robot,” Advanced Robotics, vol. 23, pp. 1–18, 2009.
[12] B. P. Gerkey and M. J. Mataric, “A formal analysis and taxonomy of task allocation in multi-robot systems,” The Int. J. of Robotics Research, vol. 23, pp. 939–954, 2004.
[13] J. Lee, Y. Cho, C. Nam, J. Park, and C. Kim, “Efficient obstacle rearrangement for object manipulation tasks in cluttered environments,” in Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA), 2019, pp. 2917–2924.
[14] C. Nam, J. Lee, S. H. Cheong, B. Y. Cho, and C. Kim, “Fast and resilient manipulation planning for target retrieval in clutter,” in Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA), 2020.
[15] C. Nam, S. H. Cheong, J. Lee, D. H. Kim, and C. Kim, “Fast and resilient manipulation planning for object retrieval in cluttered and confined environments,” IEEE Trans. on Robotics, 2021.
[16] J. Ahn, J. Lee, S. H. Cheong, C. Kim, and C. Nam, “An integrated approach for determining objects to be relocated and their goal positions inside clutter for object retrieval,” in Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA), 2021.
[17] S. Russell and P. Norvig, “Artificial intelligence: a modern approach,” 2002.
[18] U. Technologies, “Unity Robotics Hub,” https://github.com/Unity-Technologies/Unity-Robotics-Hub, [Online].