Multi-Robot Multi-Object Rearrangement in Assignment Space

Martin Levihn  Takeo Igarashi  Mike Stilman

Abstract—We present Assignment Space Planning, a new efficient robot multi-agent coordination algorithm for the PSPACE-hard problem of multi-robot multi-object push rearrangement. In both simulated and real robot experiments, we demonstrate that our method produces optimal solutions for simple problems and exhibits novel emergent behaviors for complex scenarios. Assignment Space takes advantage of the domain structure by splitting the planning up into three stages, effectively reducing the search space size and enabling the planner to produce optimized plans in seconds. Our algorithm finds solutions of comparable quality to complete configuration space search while reducing the computing time to seconds, which allows our approach to be applied in practical scenarios in real-time.

I. INTRODUCTION

We present a novel approach to the problem of multi-object rearrangement planning and execution. Efficient object rearrangement is an important task for robot systems with applications ranging from assembly to household service. Moving multiple objects to target locations leads to long execution times for a single robot. Multi-robot systems can perform such tasks faster through collaboration. However, searching the space of all possible motion combinations for multiple robots has a branching factor that is exponential in the number of robots. Fig. 1 shows an example challenge of setting dishes which is solved by our planner in seconds.

Rearrangement is known to be a PSPACE-hard problem even for simplified single-agent domains [16], [18]. Since complete and optimal planners are presently not feasible, prior work has looked into stochastic optimization [10] methods and territory strategies [4], [2] that split the working regions of robots. In contrast, we present a search space reduction technique that yields near-optimal solutions for simple cases and novel behaviours in complex environments.

Considering all possible motions of all the robots and objects would lead to a configuration space of dimension exponential in the number of entities [15]. Instead, we first plan the optimal paths that the objects must traverse to reach their configuration. The remaining problem in Assignment Space is to decide which robot should push which object at any given time, taking into account the space occupied by other robots and objects at that time and the time required to reach the object. We address this problem by first finding the optimal assignment of robots to objects, resolving the motion constraints and finally introducing a credit system to ensure that robots do not stay idle but rather prepare for future assignments by navigating to appropriate locations.

M. Levihn (levihn@gatech.edu), T. Igarashi (takeo@acm.org) are with the JST ERATO, Japan. T. Igarashi is with the University of Tokyo, Japan. M. Levihn, M. Stilman (mstilman@cc.gatech.edu) are further with the Georgia Institute of Technology, USA.

II. RELATED WORK

Existing research on multi-robot manipulation planning can be categorized into two areas:

1) Joint manipulation of a single object.
2) Joint manipulation of many objects with each object manipulated by an individual robot at a given time.

Our problem and solution are most closely related to the latter category. We therefore briefly overview the former. Early work on single object cooperation of multiple robots was done by Rus, et al. [5], showing multiple robots cooperating in rearranging a piece of furniture. Wang et al. presented the concept of object closure by multiple robots in order to successfully manipulate an object [6]. Yamashita, et al. demonstrated multiple robot cooperation in a 3D environment [7]. The authors used the concept of dividing the motion planner into a global path and a local manipulation planner. These approaches do not directly address the problem of moving multiple objects.

Multi-object rearrangement planning adds the complexity of deciding which object should be moved at what time and by which robot. Previously, Inoue presented a planner for a group of robots with multiple tasks [10]. Robots could carry multiple objects at a time. The planning algorithm utilized simulated annealing and scheduling with prioritization for synchronization of the robots. In contrast, our algorithm is entirely deterministic and requires no prioritization.

Fujii, et al. introduced a territorial approach for multiple robot rearrangement planning in [4] and [2]. The authors proposed a planning algorithm that divides the workspace of the robots into distinct regions. Territories are constructed using a Voronoi diagram and assigned to robots based on the robots workspace distances to the regions. Robots always stay within their assigned regions and objects are passed between territories via delivery points on the territory boundaries. Task constraints are solved using an extended project
Our work is motivated by two principles:

1) Our planner should give a practical and efficient solution to the object rearrangement problem using multi-robot systems to minimize task completion time.

2) Our planner must be able to take advantage of simple, low-cost multi-robot systems such as those that can only push.

A complete, provably optimal planner is presently infeasible for practical, online solutions. However, we seek to produce solutions of comparable quality in reasonable time. First, we developed a complete optimal planner to gain insight into typical optimal behaviors. The optimal planner minimizes time to complete the rearrangement task and is based on an A* [3] search over the full configuration space in a time-and-space discretization of the robots, objects and their environment. Its search is guided by a heuristic estimate between objects and goals.

To demonstrate the time complexity of an optimal solution in this domain, we present two example plans produced by the complete optimal planner mentioned (Fig. 2 and 3). While the computational times of the planner were orders of magnitude longer than our proposed planner, the solutions themselves provided the insights for our approach which yields the same results on both problems.

Consider Fig. 2. Two robots \( R_0 \) and \( R_1 \) must rearrange objects \( O_0 \), \( O_1 \) and \( O_2 \) to their goal locations \( G_0 \), \( G_1 \) and \( G_2 \). A naive plan would guide \( R_0 \) and \( R_1 \) to push the adjacent objects \( O_1 \) and \( O_2 \) towards their goal positions. \( R_0 \) would finish the task by coming back and pushing the remaining \( O_0 \) to its final place.

However, the optimal plan, shown in Fig. 2, requires \( R_0 \) to be reassigned from \( O_1 \) to \( O_0 \) part-way through the execution. In this case, both \( O_0 \) and \( O_1 \) are completed concurrently. While the solution is interesting, it is not obvious what heuristic means might be used to achieve such a behavior by an efficient planner.

Next, consider the domain in Fig. 3. For object \( O_0 \) to be pushed into \( G_0 \) by \( R_0 \), the robot would have to relocate to the right side of \( O_0 \) part-way through the plan. The optimal plan avoids unnecessary movement:

(a) \( R_0 \) starts pushing \( O_0 \). Simultaneously, \( R_1 \) prepares to cooperate by moving towards the obstacle corner.

(b) Once \( R_0 \) reaches the corner with \( O_0 \), \( R_1 \) takes over and start pushing \( O_0 \) towards its goal location \( G_0 \).

This solution not only efficiently utilizes all the robots but also it introduces a preparatory motion for \( R_1 \) that brings it within pushing range of the obstacle once it arrives.

The optimal planner takes approximately 20s to produce both plans. In contrast, our algorithm produces these same plans in less than 0.5s.

IV. Problem Formulation

The problem considered by this work is extended from [17], [9]. A world \( W \) is a gridded 2D-workspace populated with:

- \( \text{Static Obstacles: } S = \{S_0, S_1, \ldots, S_k\} \)
- \( \text{Objects: } O = \{O_0, O_1, \ldots, O_n\} \)
- \( \text{Robots: } R = \{R_0, R_1, \ldots, R_m\} \)

The world configuration at time \( t \) is defined by the positions \( R_i^t \) and \( O_j^t \) of each of the robots and objects at time step \( t \) respectively: \( W^t = \{R_0^t, \ldots, R_m^t, O_0^t, \ldots, O_n^t\} \).

The rearrangement problem requires a fully specified starting configuration, \( W_0 \), and a partially specified \( W^g \) with only the goal locations \( O^g = O_0^g, \ldots, O_n^g \) for the objects.
planner must find a sequence of collision-free actions for the robots that result in the relocation of the objects from their initial configuration $O^i$ to the goal $O^g$.

The robots operate in the environment through the use of primitives. Without loss of generality, we define two primitives for the pushing domain:

- **Move** $(m(i, x))$ is a displacement of $R_i$ to location $x$.
- **Push** $(p(i, j, x))$ is displacement of object $O_j$ to location $x$ by robot $R_i$.

Since we use non-holonomic robots, a robot $R_i$ performing a push on $O_j$ must be located directly behind the object and oriented towards the pushing direction. Let $R(O_j, x)$ denote this configuration for any robot $R_i$ with respect to $O_j$ at $x$.

V. PLANNING ALGORITHM

Our planning algorithm takes in the initial world state $W^0$ and goal object configurations $O^g$. It outputs an ordered list of sets of primitives for the robots that achieves $O^g$.

The algorithm operates in three stages:

- **Primitive Generation** Our algorithm computes the object displacements required to reach $O^g$ from $O^i$ (Fig. 4(a)). These displacements are used to derive the required push primitives (Fig. 4(b)).
- **Assignments** Assignment space planning is utilized to assign each primitive to specific robots (Fig. 4(c))
- **Post Processing** Finally we compute motion paths that displace the robots according to their assigned primitives (Fig. 4(d)).

We now consider each step in detail:

A. Primitive Generation:

- For each object $O_i$, a path $P_i$ is built from its start to goal location $(O^i_i \rightarrow O^g_i)$ by using A* search.

- A* is performed with the constraint that a valid push configuration $R(O_j, x)$ for each node must exist.

- The sequence of push primitives, $p(i, j, x)$, is computed from each generated path with $O_j$ and $x$ specified but not yet assigned to a robot $i$.

- Each object path is found independently from the other objects paths. This results in least commitment concerning prioritisation of objects displacement at this planning stage. Ordering constraints on the objects are resolved in the next planning stage which also accounts for robot motion.

B. Assignment Space Planning

After obtaining the push primitives from V-A, robots must be assigned to these primitives.

Rather than search the space of all possible assignments of all primitives, we specifically focus on assignment of push primitives to robots. This defines a new, reduced search space that allows us to use simple graph search techniques to perform the search. This is the Assignment Space (AS).

In order to define Assignment Space as a search space we must define: states, state expansions and costs.

**Definition 1 (AS State).** Let $a_i$ be either a push primitive $p(i, j, x)$ or a no-op $(-)$. An AS state, $s^l = \{a_0, \ldots, a_m\}$ assigns each robot $R_i$ either a push primitive or a no-op.

Given an ordered sequence of AS states, $(s^0, \ldots, s^l)$, a primitive $p(i, j, x)$ is considered to be assignable in $s^l$ if all prior push primitives on $O_j$ have been assigned in some earlier states $s^k (k < t)$. A valid $s^l$ must use only assignable primitives or no-ops.

Note that an AS state $s^l$ is equivalent to a world configuration $W^k$ resulting from executing all the assigned push primitives from assignment states $s^k (k \leq l)$. Hence, the goal state $s^g$ is equivalent to $W^g$.

**Definition 2 (AS Expansion).** The expansion operation on a state generates states for all valid robot assignments for assignable push primitives.

An assignment of the robot $R_i$ to the push primitive $p(i, j, x')$ is valid if and only if a motion plan exists that displaces $R_i$ to the push configuration $R(O_j, x)$ and the location $x'$ and its associated $R(O_j, x')$ are not occupied by other robots, objects or obstacles.

**Definition 3 (AS Cost).** The cost of a state $s$ is the maximum cost of the optimal motion paths necessary to displace an assigned robot to $R(O_j, x)$, the push configuration associated with its assigned push primitive $p(i, j, x')$ plus the cost of executing the push primitive.

The number of assignments does not alter the cost, which is just defined by the single most expensive assignment. Note that this implicitly encodes the execution time of transitioning from one world state to another without directly searching the space of all possible robot motions.

Given the definition of the reduced assignment search space, we apply $A^*$. We present an intuitive description of the subroutines for expanding nodes and computing cost.
Algorithm 1 EXPAND-NODE($n$)

1: $nodes := \emptyset$;
2: $prim := \emptyset$; \{assignable push primitives\}
3: $reach := \emptyset$; \{reachable push configurations for $R_i$\}
4: for $\forall O_j \in O$ do
5:  if all primitives $p(i, j, x)$ have been assigned then
6:    skip;
7:  end if
8:  $p(i, j, x) := \text{next unassigned primitive on } O_j$;
9:  if $x$ or $R(O_j, x)$ of $p(i, j, x)$ are blocked then
10:    skip;
11:  end if
12:  append($prim, p(i, j, x)$);
13: end for
14: for $\forall R_i \in R$ do
15:  for $\forall p(i, j, x) \in prim$ do
16:    if $R_i$ can reach $R(O_j, x)$ then
17:      append($reach[i], p(i, j, x)$);
18:    end if
19:  end for
20:  append($reach[i], \emptyset$) \{no assignment\}
21: end for
22: $nodes = \text{GET-COMBS}(reach, 0, \emptyset, \emptyset)$;
23: return $nodes$;

Algorithm 2 GET-COMBS($reach, k, cur, nodes$)

1: if $i \equiv |R|$ then
2:  $APPEND(nodes, cur)$;
3:  pop($cur$);
4:  return $nodes$;
5: end if
6: for $\forall p \in reach[k]$ do
7:  if $p(i, j, x) \notin cur$ then
8:    $append(cur, (p(i, j, x)))$
9:    $\text{GET-COMBS}(reach, k + 1, cur, nodes)$
10: end if
11: end for
12: pop($cur$);
13: return $nodes$;

Algorithm 1 gives pseudocode for the state expansion function. The algorithm first determines the assignable push primitives in lines 4-13 and the robots that can reach the push configurations associated with these push primitives in lines 14-21. Line 20 adds the possibility of the no-op assignment. All valid assignment combinations are then created by utilizing algorithm 2. The algorithm recursively goes over each robots possible assignments in lines 6-9 and, upon reaching the recursion depth, creates the new states. Line 7 enforces the constraint that a push primitive cannot be assigned to more than one robot in a given state.

Algorithm 3 provides pseudocode for the cost function. For the heuristic function, we use the number of remaining, unassigned push primitives. The algorithm terminates once the next node to be popped from the open queue has no assignable push primitives remaining.

1) Reducing Expansion Time: Valid node expansion and cost computation require verified motion paths for each robot assigned to a motion primitive. To obtain these motion paths, the swept volumes of the objects and other robots must be taken into account during each path construction. Both computationally expensive operations are frequently called.

In order to resolve this bottleneck, our implementation does not take the swept volumes of the other robots into account and delays the construction of valid parallel motion paths to a post-processing step. Only the object configurations are considered. Consequently motion paths can be computed using standard $A^*$, yielding significantly lower computation times. The resulting motion plans are used as estimates for the existence of a path for the expansion operation and state cost calculation.

Intuitively, robots can occupy the same space, they just need to wait their turn. Our experimental results show that this decision does not significantly alter solution optimality.

2) Constraints: The ordering constraints discussed in V-A are automatically resolved by the expansion operation and the search over the assignment space.

C. Post Processing
As the assignment space search is not guaranteed to construct valid motion plans, our proposed method does perform a post processing step.

For each node in the assignment space solution, motion paths are re-constructed in order of original estimated path length. Each new paths takes into account the swept volumes of all the paths generated prior to it in both time and space.

D. Optimization - Limited Node Expansion
It is not necessary to perform a node expansion after every push primitive. Nodes are only required to be expanded if
any of the requirements in Def. 2 become violated for the current assignment. To save computation time, the chosen assignment can be kept constant for multiple push primitives. This will result in shorter computation time but solutions as visualized in Fig. 2 might not be found.

Our algorithm can limit the node expansion operation to cases where either of the following occurs:

- The current assignment becomes invalid
- The push configuration for the next primitive on an object changes drastically in relation to the current state

The last condition is introduced to capture cases where an object needs to be pushed in a different direction and a significant displacement of the currently assigned robot would be required.

Our implemented system also allows the user to set an upper limit of constant push primitives after which a node expansion has to be performed. This feature is disabled during the evaluation in section VII.

VI. CREDIT SYSTEM

While the algorithm in Sec. V-B is efficient, it alone does not generate the optimal solution to Fig. 3(b) or the optimized behaviors in the attached video. This is due to the fact that the assignment search space does not consider the motion of unassigned robots. Consequently in Fig. 3(b), $R_1$ would remain in its initial configuration until it is assigned to perform the horizontal push. This assignment cannot occur until $R_0$ has completed the vertical pushes. Such circumstances unnecessarily increase task completion time. Fig. 4(d) demonstrates that the timing for $R_1$’s move primitives is unclear. We resolve this by incorporating ideas from market-based robot task allocation [13] through a system of credits.

A. Preliminaries

Conceptually, even if a robot is not assigned to any push primitive for an assignment state, it could still execute move primitives with cost up to the cost of the state without increasing the state’s cost. It will therefore accumulate credit for remaining unassigned. The credit each robot receives in a state is equal to the maximum cost of a push operation as described in Algorithm 3. The accumulated credit can pay for the costs of a motion plan in a later state.

Fig. 5 visualizes the credit concept. The robot $R_1$ is not assigned to a push primitive in Fig. 5(a) or 5(b). During this time, it accumulates credit. In other words, $R_1$ can reach every configuration within the credited region (gray circle) with a cost of 0 and everything outside the credited region with a cost discounted proportional to the credited region size (diameter of the circle). Since the assignment of robots is based on the cost of reaching $R(O_j, x)$, the credit method will influence the assignment of robots to objects.

B. Paying Off Credit

The credit that a robot used to discount the cost of a motion plan has to be paid off. If credit was used to discount the cost of a motion plan then this assumes that the robot is proportionally closer to the assigned object than it actually is for the state it was assigned. We therefore extend the post-processing step to create motion plans that use credit.

The post-processing step iterates over each solution node and creates valid motion plans by taking the swept volumes of previously computed motion plans for other robots and objects into account. In Section V-C this could be done for each node independently, however the credit system requires interleaving of the states.

If the post-processing step detects that credit was used in a state to discount motion plan costs, swept volumes of the previous (consecutive) states in which the credits were granted are also taken into account for creating the motion plan. The resulting motion plan in turn, is then split up according to the states the credit was granted and the partial motion plans are saved in the appropriate states.

The credit method assumes that the robot can reach every configuration within the credit region with a direct path. However, due to the movement of robots and objects, this assumption may not hold. If this assumption does not hold, the post-processing step may fail in creating motion plans for the robot that is equal to the used credit. In case of such a failure, the post-processing steps first checks if the robot was granted more credit than it used, e.g. if it was not assigned for a longer period. If this is true, the post-processing step attempts to move the robot in earlier steps as well. This serves to recover from failure without increasing overall cost. Only if a credited path cannot be constructed over all previous states where the robot is idle does the cost of the overall plan increase.

VII. EXPERIMENTS AND EVALUATION

We have validated our algorithm using real robot trials and performed extensive simulations to demonstrate its efficiency.
We applied these algorithms to three sets of configurations, each with a diverse degree of difficulty: We evaluated four types of algorithms:

1) **Basic**: The most basic version of our approach
2) **Basic + Credit System**: As presented in Section VI
3) **Basic + Credit + Limited Expansion**: As presented in Section V-D
4) **Full Configuration Space Search**: As discussed in Section III, a complete and optimal planner purely for benchmarking

We applied these algorithms to three sets of configurations, each with a diverse degree of difficulty:

- **Set 1**: Two simple configurations, as shown in Fig.2(a) and Fig.3(a). The purpose of this set was to evaluate the basic capabilities of our proposed method.
- **Set 2**: 10 randomly generated configurations. The purpose of this set was to gather performance statistics. However, for this set the full configuration space search was unable to produce a plan within 24 hours computation time. Hence, we designed a simpler version in Set 3.
- **Set 3**: 10 manually designed small scale configurations solvable using a full configuration space search. This set allows for plan quality comparisons to the optimal solutions.

### A. Simulated Experiments

Fig. 6 shows an instance of our 2D simulation environment, which was discretized using a $26 \times 16$ grid. Primitives were restricted to the four adjacent grid cells (left, right, up, down) of the current location of the robot or object. All experiments were performed on an Intel Core 2 Duo (2.80 GHz).

We evaluated four types of algorithms:

1) **Basic**: The most basic version of our approach
2) **Basic + Credit System**: As presented in Section VI
3) **Basic + Credit + Limited Expansion**: As presented in Section V-D
4) **Full Configuration Space Search**: As discussed in Section III, a complete and optimal planner purely for benchmarking

We applied these algorithms to three sets of configurations, each with a diverse degree of difficulty:

- **Set 1**: Two simple configurations, as shown in Fig.2(a) and Fig.3(a). The purpose of this set was to evaluate the basic capabilities of our proposed method.
- **Set 2**: 10 randomly generated configurations. The purpose of this set was to gather performance statistics. However, for this set the full configuration space search was unable to produce a plan within 24 hours computation time. Hence, we designed a simpler version in Set 3.
- **Set 3**: 10 manually designed small scale configurations solvable using a full configuration space search. This set allows for plan quality comparisons to the optimal solutions.

### B. Comparative Results

1) **Computation Time**: Fig. 7 summarises the planning times obtained for sets 1 and 2\(^1\).

Interestingly, the results demonstrate an average computation time reduction of 35% if the credit system is utilized. Even more, Map 9, visualized in Fig. 8, was not solvable at all using just the basic algorithm before running out of memory. These results can be explained by the reduced solution length if the credit method is used as discussed below.

Additionally limiting the node expansion operations, showed to further improve the average computation time savings to 44%. The additional savings are obtained from the reduced search tree depth.

2) **Plan Quality**: Fig. 9 shows the final plan costs for Set 1 and 2. The results confirm that the credit system reduces the plan cost compared to the basic algorithm (25% on average). The limited node expansion in turn showed only slight increases in the plan costs (6% on average).

Fig. 10 shows the costs for Set 3. Comparatively, it is observed that the plans generated by our proposed methods exhibit similar quality to the optimal solutions. Our method (Basic + Credit) created optimal solutions in 4 cases, and yield only a 14% plan cost increase on average.

### C. Experiments with a Physical Multi-Robot System

In addition to simulation, we conducted experiments with robots to demonstrate the applicability of our proposed method in a real physical environment.

Our physical system consisted of:

- A set of custom-made circular mobile robots

\(^1\)Detailed results for set 3 are not included as it was manually modified to alter computation time. On average, the full configuration space search for Set 3 took 45 min and our proposed methods 1.2 sec.
In this work, we have proposed and evaluated an efficient algorithm for solving the multi-robot multi-object rearrangement problem. Our algorithm was shown to be effective both in simulation and in real robot environments. Not only is it sufficiently efficient for online use, but also it is able to generate optimal solutions for simple domains. Finally, even in complex domains as shown in the video, our algorithm produces optimized emergent behaviours that clearly show the collaboration between the robots.

We achieved this performance by first reducing the configuration space search problem into the assignment space for a robot to push an object, it could generate an object plan such that no robot can actually reach the space. We anticipate that feedback and small perturbations to the plans at each level of the hierarchy can be made to produce a complete algorithm.

The applicability of the algorithm is currently limited to small scale systems with up to 4-5 robots and objects. This limitation steers from the large branching factor during the node expansion operation. In future work we will therefore investigate techniques to reduce the branching factor such as only considering up to \( n \) robots within a vicinity of each object as possible assignments during the node expansion operation.

**Acknowledgements**

This research was partially supported by NSF grant IIS-1017076. We thank Youichi Kamiyama for providing the robots.

**References**


