IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012

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.

Fig. 1. Multiple robots must push multiple objects to designated goals.

II. Related Work

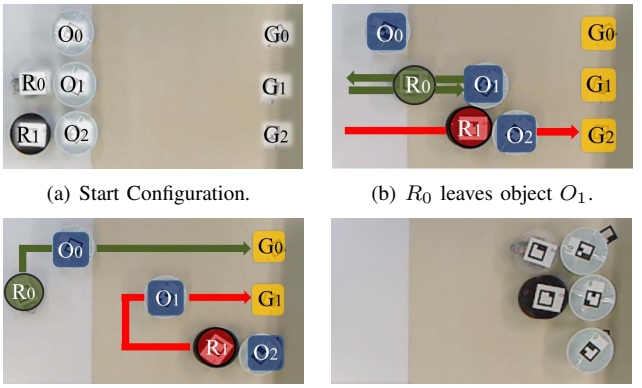
Existing research on multi-robot manipulation planning can be categorized into two areas:

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



(a) Start Configuration. (b) R_0 leaves object O_1. (c) R_1 completes O_1 and R_0 completes O_0. (d) Goal configuration, robots arrive simultaneously.

Fig. 2. Real robot execution of an optimal plan generated by our algorithm for a relatively simple problem where the goals are directly accessible.

scheduling problem (PSP) solver. We observe that while the territorial approach eliminates the synchronization problem between robot motion paths, the resulting behaviours can be significantly suboptimal. As robots are not permitted to leave the assigned territories, such methods can lead to frequent unnecessary object passing in simple examples and robot idling when their territories are not in use.

Oyama, et al. [8] improved the optimality of the territorial approach by instantiating delivery points at narrow corridors. However, this method still faces the earlier challenges on simple problems and furthermore does not address the feasibility of switching between robots in corridors. Further, the algorithm requires object grasping, which is not directly applicable to our domain where mobile robots are required to push the objects.

More generally, market-based coordination techniques have been extensively studied as methods for task allocation among teams of robots [13]. Market-based coordination *decompose-then-allocate* has been applied to multi robot teams with one of the earliest works presented in [12]. Our algorithm is related to this category, however it differs from existing work due to a more centralized allocation approach with a specific focus on object rearrangement.

Outside of the assignment of single robots to specific tasks, Yamashita, et al. [14] proposed the use of tools to allow the simultaneous manipulation of multiple objects by multiple robots. However this work was primarily concerned with the introduction of tools and did not provide a global planner that explicitly reasons about robot assignments.

To our knowledge, there is no existing planning algorithm capable of producing both optimal results as demonstrated in Fig. 2 and emergent optimized behaviors for more complex domains as shown in the attached video.

III. Motivation

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 $0.5s$.

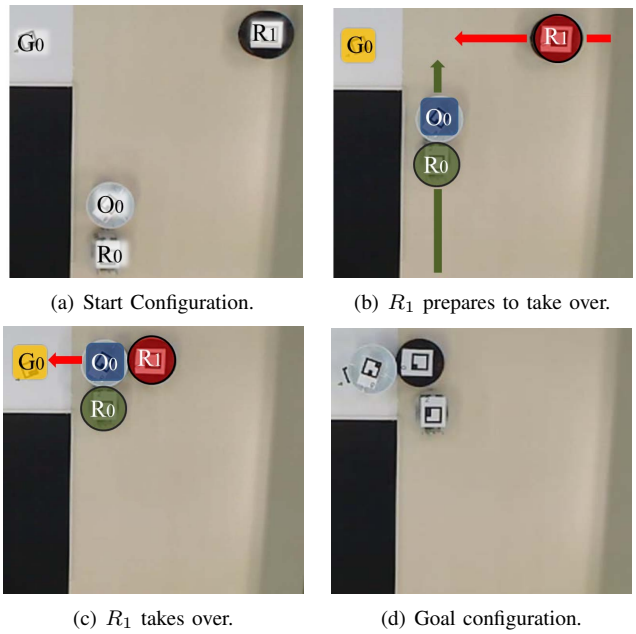
IV. Problem Formulation

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

- **Static Obstacles:** $\mathcal{S} = \{S_0, S_1, ..., S_k\}$
- **Objects:** $\mathcal{O} = \{O_0, O_1, ..., O_n\}$
- **Robots:** $\mathcal{R} = \{R_0, R_1, ..., R_m\}$

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

The rearrangement problem requires a fully specified starting configuration, W^0, and a partially specified W^g with only the goal locations $\mathcal{O}^g = O_0^g, ..., O_n^g$ for the objects. The



(a) Start Configuration. (b) R_1 prepares to take over.

(c) R_1 takes over. (d) Goal configuration.

Fig. 3. Real robot execution of our optimal plan where the object cannot be directly pushed to the goal.

planner must find a sequence of collision-free actions for the robots that result in the relocation of the objects from their initial configuration $\mathcal{O}^s$ to the goal $\mathcal{O}^g$.

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 $\mathcal{O}^g$. It outputs an ordered list of sets of primitives for the robots that achieves $\mathcal{O}^g$.

The algorithm operates in three stages:

- **Primitive Generation** Our algorithm computes the object displacements required to reach O^g from O^s (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 ($O_i^s \rightarrow O_i^g$) 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_k be either a push primitive $p(i,j,x)$ or a no-op $(-)$. An AS state, $s^t = \{a_0, \ldots, a_m\}$ assigns each robot R_i either a push primitive or a no-op.*

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

Note that an AS state s^l is equivalent to a world configuration $\mathcal{W}^t$ 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 $\mathcal{W}^g$.

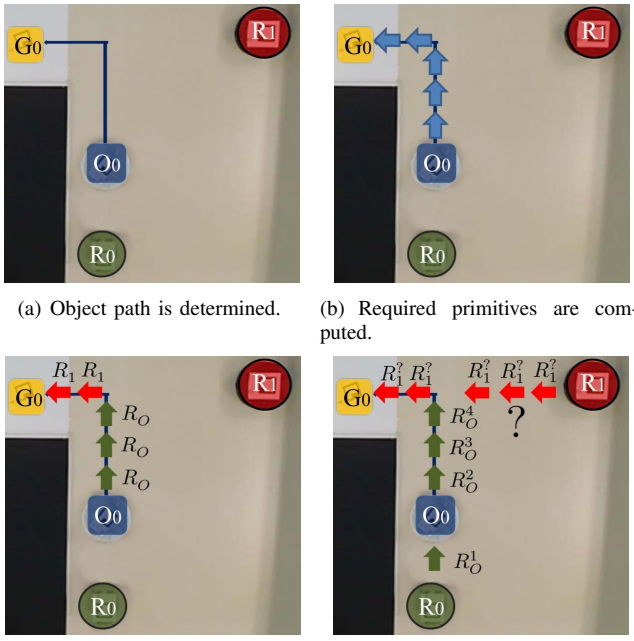
Definition 2 (AS Expansion)**.** *The expansion operation on a state generates states for all valid robot assignments for assignable push primitives.*

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.



(a) Object path is determined. (b) Required primitives are computed.

(c) Primitives get assigned. (d) Robot motion paths are created.

Fig. 4. Algorithm workflow example.

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 can not 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.

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** $\forall O_j \in \mathcal{O}$ **do**
5: **if** all primitives $p(i,j,x)$ have been assigned **then**
6: skip;
7: **end if**
8: $p(i,j,x) :=$ 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** $\forall R_i \in \mathcal{R}$ **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($reach[i]$, $-$) {no assignment}
21: **end for**
22: $nodes =$ GET-COMBS($reach, 0, \emptyset, \emptyset$);
23: return $nodes$;

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

1: **if** $i \equiv |\mathcal{R}|$ **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: GET-COMBS($reach$, $k+1$, cur, $nodes$)
10: **end if**
11: **end for**
12: pop(cur);
13: return $nodes$;

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

Algorithm 3 COST($node$)

```
max = 0;
for p(i, j, x) ∈ node do
    path_cost=GET-PATH-COST(R_i, R(O_j, x));
    push_cost=GET-PUSH-COST(p(i, j, x));
    if path_cost + push_cost > max then
        max = path_cost + push_cost;
    end if
end for
return max;
```

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

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.

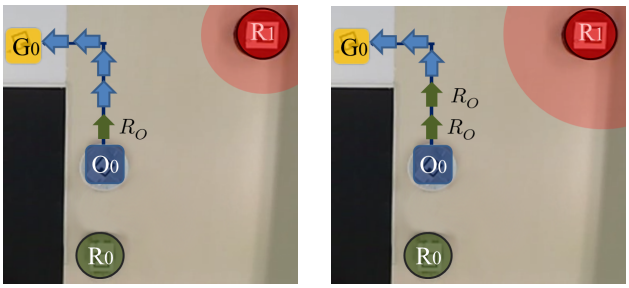
VI. Credit System

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_1s *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



(a) Robot not assigned, receives credit.

(b) Robot sill not assigned, credit increases.

Fig. 5. Visualization of the credit method. Robots credit is proportional to the time they are unassigned.

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

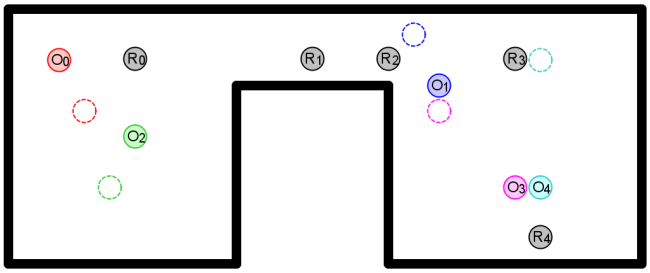


Fig. 6. An instance of our simulation environment.

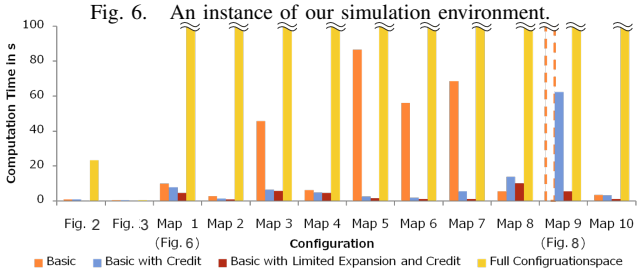


Fig. 7. Runtime comparison. The first two data groups corresponds to *Set 1*, the remaining 10 to *Set 2*. The full configuration space search returned results only for set 1 within 24 hours computation time. Map 9 was not solvable with the credit method disabled before running out of memory.

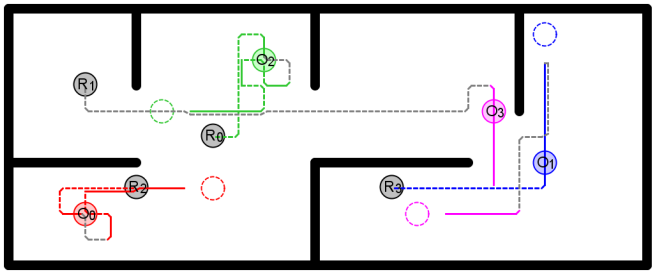


Fig. 8. Not solvable without Credit. The shown solution was constructed utilizing the basic + credit system algorithm. Robot paths are color coded based on the object assignments that triggered the displacement. Gray sections are displacements without actual assignments caused by the credit

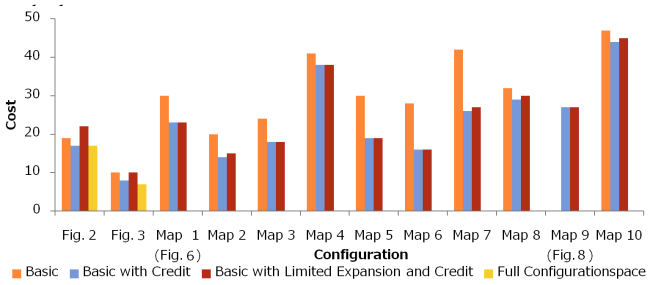


Fig. 9. Cost comparison for the randomized configurations.

and resulting plan quality.

A. Simulated Experiments

Fig. 6 shows an instance of our 2D simulation environment, which was discretized using a 26×16 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.

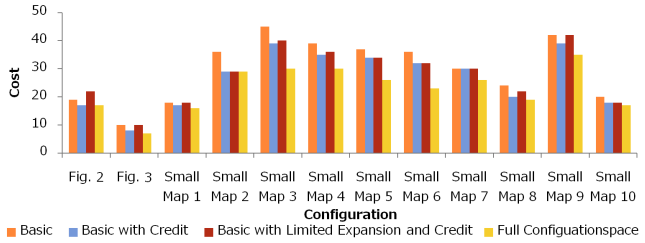


Fig. 10. Cost comparison for manually created configurations.

- A set of circular movable objects located (as the robots) on a tabletop
- A ceiling-mounted camera connected to a host computer

Using the camera data, the host computer tracked the location and orientation of the objects and robots through their attached visual markers. The environment is shown in Fig. 1.

The determined start configuration was passed to our planning system, which outputs separate queues of move and push primitives for each robot. The host computer then send wireless command signals to each robot according to its next primitive. To avoid collisions all queues were progressed simultaneously and only after every robot had successfully executed its current primitive. Since object drift can occur during the execution of a push primitive, we applied our Dipole Method [1] to overcome motion uncertainty. The dipole method computes a dipole field around the object which guides the robot. This ensures successful delivery.[2]

We found that this experimental environment enables the robots to successfully complete the rearrangement task according to our planner's output.

VIII. Discussion

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 domain. Next, we introduced the *credit method* to ensure that robots would be active in pursuing their next assignments instead of idling.

Our goals in future work are to guarantee completeness and improve algorithm performance. Currently, the algorithm does not guarantee completeness due to the hierarchical approach. For instance while primitive allocation requires 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.

[2]To ensure that the robot does not leave its assigned space during the use of the Dipole Method we slightly modified the method to allow for minor displacements of the goal position. This behaviour can also be observed in the accompanying video.

Acknowledgements

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

References

[1] T. Igarashi, Y. Kamiyama, and M. Inami, "A dipole field for object delivery by pushing on a flat surface," in *ICRA, 2010*, May 2010, pp. 5114 –5119.

[2] N. Fujii and J. Ota, "Territorial and effective task decomposition for rearrangement planning of multiple objects by multiple mobile robots," *Advanced Robotics*, vol. 25, pp. 47–74(28), January 2011.

[3] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[4] N. Fujii, R. Inoue, Y. Takebe, and J. Ota, "Multiple robot rearrangement planning using a territorial approach and an extended project scheduling problem solver," *Advanced Robotics*, vol. 24, pp. 103–122(20), January 2010.

[5] D. Rus, B. Donald, and J. Jennings, "Moving furniture with teams of autonomous robots," in *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, vol. 1. IEEE, 1995, pp. 235–242.

[6] Z. Wang and V. Kumar, "Object closure and manipulation by multiple cooperating mobile robots," in *ICRA 2002*, 2002, pp. 394–399.

[7] A. Yamashita, T. Arai, J. Ota, and H. Asama, "Motion planning of multiple mobile robots for cooperative manipulation and transportation," *IEEE Transactions on Robotics and Automation*, vol. 19, pp. 223–237, 2003.

[8] N. Oyama, Z. Liu, L. Gueta, and J. Ota, "Rearrangement task of multiple robots using task assignment applicable to different environments," in *ROBIO 2010*, December 2010, pp. 300 –305.

[9] N. Fujii, T. Chou, and J. Ota, "Rearrangement task realization by multiple mobile robots with efficient calculation of task constraints," in *ICRA 2007*, April 2007, pp. 8 –13.

[10] R. Inoue, N. Fujii, R. Takano, and J. Ota, "Realization of a multiple object rearrangement task with two multi-task functional robots," *Advanced Robotics, 25*, vol. 11, no. 12, pp. 1365–1383, 2011.

[11] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[12] P. Caloud, W. Choi, J.-C. Latombe, C. Le Pape, and M. Yim, "Indoor automation with many mobile robots," in *IEEE International Workshop on Intelligent Robots and Systems '90*, 1990, pp. 67 –72 vol.1.

[13] N. Kalra, R. Zlot, M. B. Dias, and A. Stentz, "Market-based multirobot coordination: A comprehensive survey and analysis," Tech. Rep., 2005.

[14] A. Yamashita, J. Sasaki, J. Ota, and T. Arai, "Cooperative manipulation of objects by multiple mobile robots with tools," in *Proceedings of the 4th Japan-France/2nd Asia-Europe Congress on Mechatronics*, 1998, pp. 310–315.

[15] M. Stilman and J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," in *Journal of Humanoid Robotics*, 2004, pp. 322–341.

[16] G. Wilfong, "Motion planning in the presence of movable obstacles," in *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*. New York, NY, USA: ACM, 1988, pp. 279–288.

[17] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," in *WAFR 2006*, 2006, pp. 119–135.

[18] E. Demaine, J. O'Rourke, and M. L. Demaine, "Pushpush and push-1 are np-hard in 2d," in *In Proceedings of the 12th Canadian Conference on Computational Geometry*, 2000, pp. 211–219.