Scheduling Pick-and-Place Tasks for Dual-arm Manipulators using Incremental Search on Coordination Diagrams

Andrew Kimmel and **Kostas E. Bekris**
Computer Science Department
Rutgers, the State University of New Jersey
110 Frelinghuysen Road, Piscataway NJ, USA
email:{andrew.kimmel,kostas.bekris}@cs.rutgers.edu

Abstract

A premise of dual-arm robots is increased efficiency relative to single-arm counterparts in manipulation challenges. Nevertheless, moving two high-dimensional arms simultaneously in the same space is challenging and care must be taken so that collisions are avoided. Given trajectories for two arms to pick two objects, velocity tuning over a coordination diagram can resolve collisions. When multiple objects need to be moved, a scheduling challenge also arises. It involves finding the order with which objects should be manipulated. This paper considers two ways to approach this combination of scheduling and coordination challenges: (i) a "batch" approach, where an ordering of objects is selected first; for the given ordering, velocity tuning is performed over a matrix of coordination diagrams that considers all pairs of pick-and-place trajectories; and (ii) an incremental approach, where the ordering of objects is discovered on the fly given the subset of coordination diagrams that arise depending on which object one of the arms is currently manipulating. Simulated experiments for a Baxter robot show that both methods return significantly more efficient trajectories relative to the naive "round-robin" schedule, where only one arm moves at a time. Furthermore, the incremental approach is computationally faster, it implicitly provides a good schedule for picking objects and can be used effectively when objects appear dynamically.

1 Introduction

One broad category of manipulation tasks, which appear frequently in manufacturing setups, correspond to pick and place challenges. For instance, a robot may need to grasp multiple products from a tabletop and place them in bins so that they are packaged. An example of such a scenario is shown in Figure 1. Alternatively, a robot could be tasked with stocking and retrieving items from shelves in a warehouse. It has been argued that dual-arm humanoid manipulators can be appropriate solutions for such tasks. The reasoning is that they can be easily used in facilities that have been constructed for human workers as they exhibit similar reachability to people and bi-manual skills.

This work focuses on pick-and-place tasks where multiple objects need to be manipulated and those objects

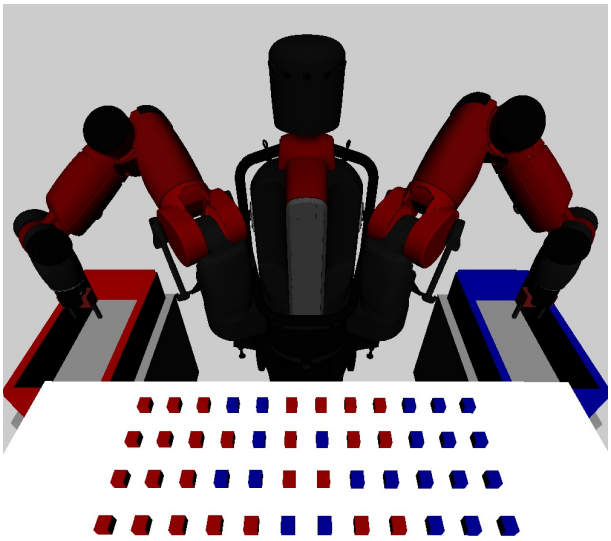

Figure 1: A dual-arm Baxter robot tasked with picking two types of objects from a tabletop and placing them into two distinct storage bins.

are separated in two groups, where a different arm needs to be used for each group. This can be useful in the case of different end-effectors, appropriate for different types of objects, and in general of separation tasks where two types of objects need to be moved to different target areas. Such tasks introduce a scheduling subproblem together with the dual-arm coordination challenge, where the ordering of the pick-and-place tasks affects the overall completion time and the conflicts arising between the two arms. For instance, the trajectory for a particular task could cause the arm to occupy the majority of the shared workspace, e.g., one of the arms reaching across the robot. Completing such a task could delay, or even obstruct, the other arm from carrying out its task depending on the placement of the objects. Consequently, scheduling the tasks for the two arms becomes a new aspect of dual-arm coordination in this context.

A naïve yet straightforward method for solving pick-and-place tasks with a humanoid robot corresponds to moving one arm at a time in a "round-robin" fashion, guaranteeing that only one of the arms operates in the workspace shared by both arms. This solution provides

fairness when different objects need to be grasped by different arms by alternating task completion. It is also simple to implement, since the coordination is minimal. The downside is that it doesn't take full advantage of the dual-arm capabilities of a humanoid robot relative to simultaneous arm motion.

Nevertheless, simultaneous arm motion corresponds to a high dimensional planning challenge, which for many popular dual-arm systems, such as a Baxter robot by Rethink Robotics, involves at least 14 DoFs. Given the complexity of motion planning, operating in the composite configuration space is computationally expensive, especially for high-quality paths. A practical alternative is to follow a decoupled methodology. For instance, paths for each arm can be computed independently and then coordination can be achieved by finding the velocity along the given paths that allows the arms to execute their task without collisions and deadlocks. This velocity tuning approach makes use of a representation known as a coordination diagram (O'Donnell and Lozano-Perez (1989); Siméon, Leroy, and Laumond (2002)). Although the coordination diagram approach is incomplete in the general case, it is computationally efficient to search, and in practice produces good quality paths for most realistic setups.

This paper first considers a "batch" scheduling approach, which is a straightforward extension of velocity tuning for multiple pick-and-place tasks. This involves a two-step process, where an ordering of the tasks for each arm is decided first. Given this ordering, the set of all coordination diagrams is implicitly searched so as to find the best coordination between the arms for that order. An alternative approach proposed here is an "incremental" scheduling method, which achieves online coordination of the two arms effectively. Tasks are assigned one at a time to each arm, and coordination diagrams are used to produce solution trajectories for a pairwise task assignment. The search method is defined so that a solution trajectory prioritizes the completion of a single task, thus "freeing" up one of the arms and allowing for a new task to be assigned.

To evaluate the effectiveness of both the "batch" and "incremental" methods, a series of simulated experiments on a Baxter robot were conducted. First, an experiment in a "tabletop" environment with a randomized distribution of objects in the scene showed that both methods produced solution trajectories that were nearly twice as fast compared to the "round robin" method, where only one arm moves at a time. This is close to the best that can be achieved by a simultaneous motion solution. The proposed "incremental" method, however, utilized much less computing resources - both processor time and memory - compared to the "batch" approach. If the coordination diagrams can be precomputed, the online computation time for both methods can be improved by a couple of orders of magnitude.

Another weakness of the "batch" scheduling approach beyond the high computational requirements is that the robot must compute the full schedule for both arms ahead of time. When a new task is added dynamically to the scene, this can result in additional cost for the "batch" approach. Simulations in a "shelf" environment, where objects appear dynamically in the scene, indicate that the "incremental" method produces even better results than the "batch" method in dynamic setups giving its capability to adapt to the scene.

Both the "batch" and "incremental" algorithms make a series of assumptions. First, the possible placements of the target objects are known during the precomputation step so as to allow the generation of grasping configurations and manipulation plans offline. Furthermore, the target objects are all geometrically similar, and are positioned in such a way so as to not prevent other objects from being grasped. Finally, the manipulator has access to perfect sensing, trajectory tracking, and grasping capabilities since the focus of the paper is on the combinatorial aspects of the problem.

2 Related Work

Manipulation Planning There are many ways to achieve manipulation planning, including tree sampling-based planners (Berenson, Srinivasa, and Kuffner (2012)), heuristic search (Cohen, Chitta, and Likhachev (2010)), constraint satisfaction (Lozano-Pérez and Kaelbling (2014)), and optimization approaches (King et al. (2013)). The underlying methodology for this work corresponds to searching solutions over a "manipulation graph" that contains "transit" and "transfer" paths (Alami, Siméon, and Laumond (1989); Siméon et al. (2004)). The graph itself can be built using sampling-based roadmaps (Kavraki et al. (1996)). One way to take advantage of multiple arms is through the use of handoffs (otherwise known as regrasps) (Vahrenkamp et al. (2009); Cohen, Phillips, and Likhachev (2014)), which involves passing an object from one arm to another. Grasp planning for multiple robots can be achieved with distributed constraint satisfiers (Panescu and Pascal (2014)).

A prototypical application of manipulation planning is in the area of bin-picking. Due to the structured nature of this domain, it is possible to take advantage of precomputed paths to speed up the online execution time. Such paths can be blended and further optimized (Ellekilde and Petersen (2013)) to account for noisy actuation and state uncertainty. The focus of such methodologies is on the optimality of trajectories generated for a single-arm and not on the effects of task ordering or dynamically appearing tasks. This work focuses on establishing a baseline framework which can effectively utilize dual-arm manipulators, and provides an adaptive online schedule which can be created in the event of dynamically appearing tasks.

Multi-Robot Coordination Multi-robot motion planning is a difficult problem due to the increased dimensionality. Coupled, complete approaches typically do not scale well with additional robots, despite the fact that there are methods which reduce the number of effective DOFs (Aronov et al. (1999)). A decoupled

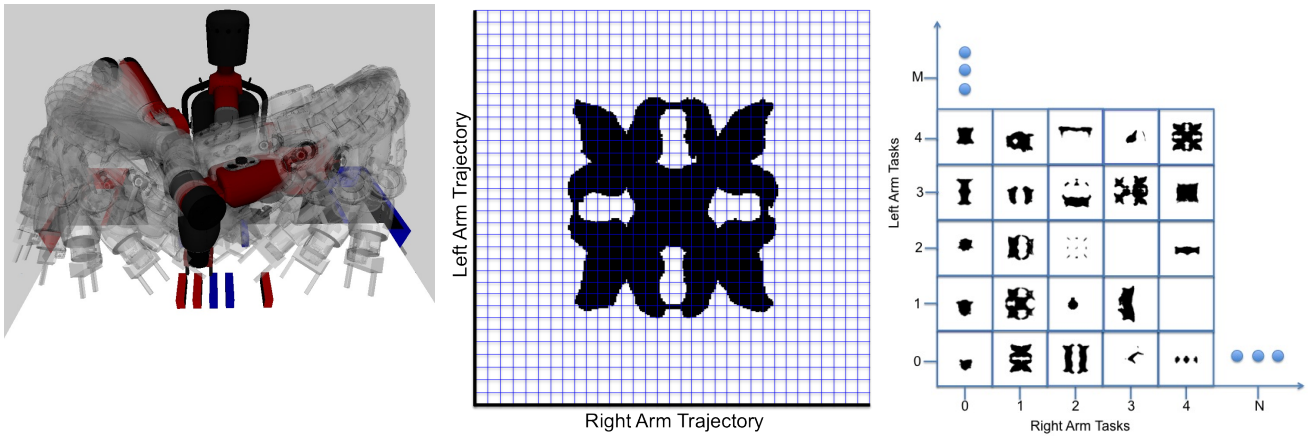

Figure 2: (Left) Given a pair of pick-and-place tasks for the two arms, the collisions between the two corresponding trajectories are computed. (Middle) The results of the pairwise collision checking define a *coordination diagram*. (Right) Building coordination diagrams for all pairs of trajectories defines a *coordination matrix*. Precomputing the entire matrix is useful in scheduling approaches as it is then possible to quickly query the relevant *coordination diagram* given a specific pair of designated tasks for each arm.

approach, which aims to best coordinate the individual paths of robots, can be used instead. Decoupling can be accomplished in several ways: by discretizing the common workspace and adopting a master-slave prioritization scheme (Zurawski and Phang (1992)), by using prespecified velocity profiles and varying the start times of the robots (Akella and Hutchinson (2002)), or by precomputing the collision volumes between the arms to form coordination diagrams (O'Donnell and Lozano-Perez (1989); Siméon, Leroy, and Laumond (2002)). This paper makes use of such coordination diagrams to achieve simultaneous, collision and deadlock-free movements of both arms.

Scheduling There is a very rich literature on scheduling challenges. One way to schedule a set of tasks is through the use of "batch" scheduling, which computes schedules over a set of assigned tasks (Maheswaran et al. (1999)). Genetic algorithms can be used to find the time-optimal batch schedules in the context of robotic manipulation (Xidias, Zacharia, and Aspragathos (2010)).

In contrast to batch scheduling, online scheduling methods are able to handle dynamic allocations of new tasks. Some work on online schedulers makes use of swarm optimization techniques (Xu, Hou, and Sun (2003); Yan et al. (2005); Chang, Chang, and Lin (2009)). While the proposed method here does not use such algorithms, it is also an online, incremental scheduling approach.

There has also been significant work towards addressing the issue of finding the optimal task assignment for multiple robots (Gerkey and Matarić (2004); Tang and Parker (2007); Zhang and Parker (2013)). This paper does not focus on the task assignment component of similar challenges. It instead focuses on the *ordering* of the tasks, given that they have already been assigned to a particular robot.

3 Problem Setup

Informally, a dual-arm manipulator is given a set of objects that can appear in known locations in the workspace to retrieve and place at specific goals. Each arm is assigned its own set of objects to be grasped and placed. The arms need to move the objects to their assigned goals as fast as possible while conflicts (i.e., collisions and deadlocks) need to be avoided. Thus, the problem involves finding both the order in which the objects are retrieved and placed as well as achieving collision-free and time efficient coordination. Consider a 3D workspace that contains obstacles and:

- **Two manipulators** M_{left} and M_{right} capable of picking and placing objects in the workspace.
- A set of **movable rigid-body objects** $\mathcal{O}_{left}$, where each object $o^i_{left} \in \mathcal{O}_{left}$ can acquire a **pose** in $SE(3)$ and can be grasped by the left arm M_{left}. Similarly for $\mathcal{O}_{right}$ and the right arm M_{right}.
- Two **goal regions**, which are the destinations for picked objects to be placed in. Each goal region corresponds to one of the two arms. Once an object is moved in a goal region, it is removed from the workspace.

An arm M_{arm} (i.e., either the left or the right one) is assigned an individual set of tasks $\mathcal{T}_{arm} = \{t^1_{arm}, \ldots, t^k_{arm}\}$, where a task $t^i_{arm} \in \mathcal{T}_{arm}$ corresponds to a single arm following a trajectory so as to pick a specific object $o^i_{arm} \in \mathcal{O}_{arm}$ from an initial pose and place it in the corresponding goal region.

Given an assignment of tasks $\{\mathcal{T}_{right}, \mathcal{T}_{left}\}$ to the two arms, a "manipulation coordination schedule" $\{S(\mathcal{T}_{right}), S(\mathcal{T}_{left})\}$ defines both the order of tasks with which each arm will execute the pick-and-place motions, as well as the corresponding *velocity profile* along the corresponding trajectories that the assigned arm will follow.

Overall, the problem is to find the solution schedules for both arms $S(T_{left})$ and $S(T_{right})$, which minimize *completion time* for all tasks, such that no *conflicts* arise between the arms. A *conflict* occurs both from collisions between the arms and deadlocks. A deadlock here corresponds to either arm reaching a configuration such that it is no longer possible for one of the arms (or, potentially both arms) to make progress towards completing assigned tasks.

This paper makes a few assumptions about the objects in the scene. The problem definition requires that each arm retrieves a different set of objects $\mathcal{O}_{left} \cap \mathcal{O}_{right} = \emptyset$. An example task would involve each arm equipped with a unique tool for manipulating objects. This assumption allows the results to focus on the effects of the selected schedule, rather than being an effect of different task assignments. Furthermore, each object's initial pose belongs in a set of poses that are known a priori to the robot. The rationale behind this is that the robot will be manipulating objects whose placements in the scene is predictable and in this way precomputation can be used to speed up the online solution times.

4 Method

Section 4.1 presents an overview of fundamental methodologies used throughout the rest of the paper. Then, Sections 4.2 and 4.3 describe the two different types of coordination modes, "batch" and "incremental" correspondingly, while Section 4.4 gives an overview of the precomputation utilized in this work so as to improve the running time of the proposed solutions.

4.1 Fundamentals

Given a pair of paths for the two arms, processing the collisions between all pairs of states along each trajectory can produce a *coordination diagram* (O'Donnell and Lozano-Perez (1989); Siméon, Leroy, and Laumond (2002)) as shown in Figure 2 (middle). Each axis corresponds to a path for one of the two arms, i.e., the range is [0,1], corresponding to where along each path each arm is located. Given the maximum velocity of each arm and the length of each path, each axis is discretized into intervals along which each arm is moving an equal distance.

A vertex in the resulting *coordination diagram* corresponds to a placement of both arms at configurations along the corresponding paths. The configurations which result in collisions between the arms are invalidated and marked as black in the diagrams. Searching such a diagram involves finding a collision-free path that starts from the bottom left vertex and reaches the top right vertex and does not go through black regions. Horizontal or vertical steps in the *coordination diagram* correspond to a single arm moving with maximum velocity, while a diagonal direction means that both arms are moving simultaneously with their maximum velocity. Extracting a solution from this search will produce a coupled trajectory that moves both arms to their respective goals without collisions.

4.2 Batch Coordination

This work considers the set of all coordination diagrams, which form a *coordination matrix* as shown in Figure 2 (right). The length of the trajectories does not have to be the same, in contrast to what is shown in the figure. This matrix can be constructed by computing and storing the *coordination diagrams* for all pairs of trajectories. Once this matrix is available, it allows for the framework to quickly check different orderings of tasks and find the corresponding solution trajectories for a particular schedule.

The *batch* mode coordination approach considers the orderings of tasks for both arms and then searches the resulting coordination matrix. In particular, the approach can be broken down into two main steps:

- `COMPUTE_TASK_ORDERING`(T_{left}, T_{right})
- `SOLVE_FULL_COORDINATION`$(S(T_{left}, S(T_{right}))$

The function `COMPUTE_TASK_ORDERING` takes as parameters the set of tasks for both arms. The tasks are then ordered to create the sequence of objects that each arm should aim to pickup. This step does not consider interactions between the arms. Instead, this ordering can be determined randomly, or could correspond to optimizing certain heuristic functions, such as minimizing execution time ignoring interactions between the arms. This ordering determines which pairs of coordination diagrams are tiled together and searched by the A*.

Once the ordering of the tasks has been fixed, the method calls `SOLVE_FULL_COORDINATION` to retrieve the solution trajectory for both arms. Given the order with which the tasks have been assigned, this method combines the corresponding pairwise coordination diagrams retrieved from the coordination matrix. Running an A* in this composed diagram, which will have a similar structure to the coordination matrix shown in Figure 2 (Right), will produce the coupled trajectory that moves both arms towards their goals.

For the A* search on the coordination matrix, a vertex in this space is valid as long as it does not occupy a cell in the diagram that is marked as a collision region (colored black in the figures). The cost of expanding a vertex is expressed in terms of time and corresponds to an additional "time step" for the arms to move. This cost is therefore equivalent in all directions of expansion. Since the tasks for both arms have already been ordered, the goal of the A* is to have each arm reach the end of their task schedule. Thus, the heuristic used in the search corresponds to the summation of each arm's distance to its goal. When both arms have finished all of their assigned tasks, the A* has successfully expanded the goal vertex and the solution trajectory is returned.

The benefit of using the *batch* coordination is that the A* returns the optimal path given the computed schedule. At the same time, there is a caveat. In particular,

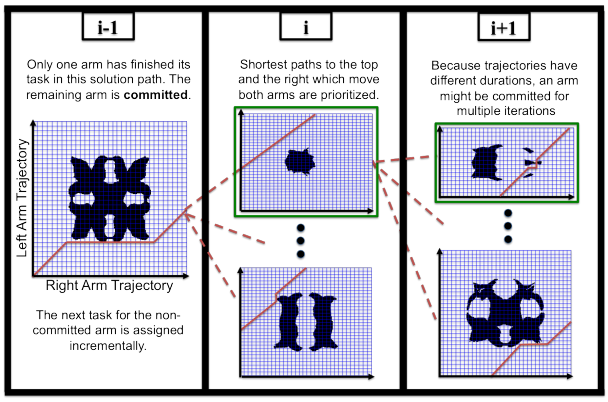

Figure 3: Initially, an arm might be committed to carrying out a pick-and-place task from a previous assignment. The non-committed arm, which has already finished its task, must be assigned a new task. This assignment is found by searching over all possible available tasks for the arm, and finding the assignment which produces the fastest solution time.

the schedule itself may not correspond to the optimal solution. Additionally, if any new tasks are introduced after the solution trajectory has been produced, there is no easy way to incorporate these tasks dynamically, i.e., until the robot has finished executing a trajectory. This would force an arm that has already completed its tasks to wait until the other arm is also finished before computing a new schedule.

4.3 Incremental Coordination

Rather than coordinating over the full coordination matrix, the *incremental* scheduling algorithm assigns a single task to each arm per iteration. An outline of an iteration for the approach is given in Algorithm 1. This method makes a distinction between a **committed** and a **non-committed** arm. If an arm has only partially completed its assigned task when a new iteration starts, it is **committed** to completing its current task, and is labeled as the *master* arm in the corresponding algorithmic. The remaining arm is labeled as the *slave* arm, which is the only arm that must be assigned a new task. This process continues as long as there are tasks remaining for either arm.

Line 1: The method `CHECK_PARTIAL_EXECUTION` determines if one of the arms is still in the middle of picking and placing an assigned object. This arm is set to be the master arm $M_{\texttt{MASTER}}$.

Lines 2-4: If both arms have finished their previously assigned tasks, a new master arm needs to be found. The method `ASSIGN_NEW_MASTER` searches the remaining tasks for both arms, and finds a suitable assignment. This can either be done by randomly selecting one of the arms to be the master, or it can be done by optimizing certain criteria. For this paper, random and *minimum conflict* assignments were tested. The *minimum conflict* assignment finds the pairwise task assignment which has the smallest amount of collision

Algorithm 1: Incremental Scheduling Algorithm

Data: $T_{\texttt{left}}$, $T_{\texttt{right}}$: left/right tasks
$\tau_{\texttt{left}}(t), \tau_{\texttt{right}}(t)$: current left/right trajectories
Result: Assigns a single task to each arm, returning the corresponding coupled trajectory to pick and place the two objects.

1 `CHECK_PARTIAL_EXECUTION`$(\tau_{\texttt{left}}(t), \tau_{\texttt{right}}(t), M_{\texttt{MASTER}})$
2 **if** $M_{\texttt{MASTER}} = \emptyset$ **then**
3 $\quad t_{\texttt{MASTER}} :=$ `ASSIGN_MASTER`$(T_{\texttt{left}}, T_{\texttt{right}}, M_{\texttt{MASTER}})$
4 $\quad$ `REMOVE_TASK`$(t_{\texttt{MASTER}}, T_{\texttt{MASTER}})$
5 $M_{\texttt{SLAVE}} := \{M_{\texttt{left}}, M_{\texttt{right}}\} - \{M_{\texttt{MASTER}}\}$
6 **if** $T_{\texttt{SLAVE}} \neq \emptyset$ **then**
7 $\quad t_{\texttt{SLAVE}} :=$ `ASSIGN_SLAVE`$(T_{\texttt{left}}, T_{\texttt{right}}, M_{\texttt{SLAVE}})$
8 $\quad$ `REMOVE_TASK`$(t_{\texttt{SLAVE}}, T_{\texttt{SLAVE}})$
9 $\quad$ `SOLVE_PARTIAL_COORDINATION`$(t_{\texttt{MASTER}}, t_{\texttt{SLAVE}})$

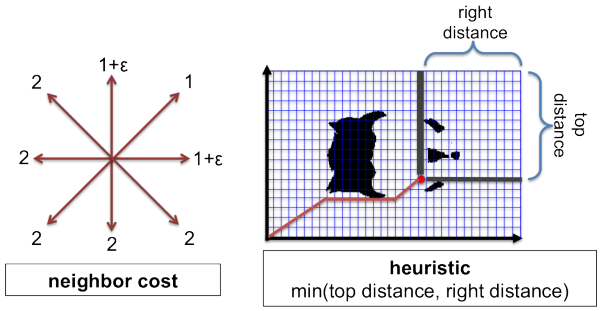

Figure 4: The neighbor cost and heuristic used by the A* in the incremental search is shown. Motions which involve one or both arms moving backwards along their trajectory are penalized during vertex expansion. The heuristic used here guides the search to allow one of the arms to finish as quickly possible.

area as determined by the *coordination diagrams*, and uses this as a heuristic to select the master arm.

Lines 5-6: The arm which is not the master arm is assigned to be the slave $M_{\texttt{SLAVE}}$. If the slave arm $M_{\texttt{SLAVE}}$ has any tasks remaining in its task set $T_{\texttt{SLAVE}}$, then an appropriate assignment must be made.

Lines 7-8: This assignment is computed by searching the remaining set of tasks for the slave arm and finding which pairwise task assignment for both arms produces the fastest solution time. An illustration of this is shown in Figure 3. The assigned task is then removed from the remaining set of tasks.

Line 9: The method `SOLVE_PARTIAL_COORDINATION` runs an A* on the coordination diagram corresponding to the pairwise task assignment of the arms. Since the *incremental* method does not make a complete ordering of the objects, and is therefore only operating on a single coordination diagram, it is no longer possible to guarantee that an optimal path will be returned by the A*. To promote trajectories that allow both arms to move simultaneously, the A* used in the *incremental*

approach was altered as shown in Figure 4. The heuristic used in the A* was the minimum distance for either arm to finish, while the goal condition was that either arm finished their currently assigned task.

The path returned by the A* is constructed such that the solution trajectory promotes the movement of both arms towards their goals, while penalizing any regressive movements. Single arm movements where the other arm is stationary is slightly penalized with a cost of $1+\epsilon$, with $\epsilon << 1$.

4.4 Precomputation

Computing pick and place motions for an individual object can be accomplished by building and querying a *manipulation graph* (Alami, Laumond, and Siméon (1997)). The graph contains roadmaps for transfer and transit paths, which are connected at grasping configurations. Querying the roadmap for a pick-and-place task corresponding to an individual object and arm will produce trajectories similar to the ones shown in Figure 2 (Left).

In many industrial settings, such as a factory or warehouse, it is reasonable to assume that a) the robot has knowledge of the static and immovable workspace, and b) movable objects can appear in poses out of a set of known ones.

To take advantage of such structured setups, the proposed framework performed the majority of the computationally expensive functions offline. This includes building the manipulation roadmap ahead of time for the known static scene and the known poses where objects can appear. In particular, the following data structures were precomputed and stored:

1. The *manipulation roadmap* corresponding to the static workspace.
2. The trajectories for each arm to pick and place objects that can appear in the predetermined poses, which are computed with the aid of the manipulation roadmap.
3. The *coordination diagrams* storing the collision checking information between pairs of trajectories for the two arms, as shown Figure 2 (middle).
4. The *coordination matrix* storing each computed *coordination diagram*, as shown Figure 2 (right).

5 Experimental Evaluation

Simulated experiments were performed, using a model of the dual-arm Baxter robot, where the robot needs to pick two sets of objects and place them into different bins, one on each side of the robot. Two separate problem scenarios, shown in Figure 5, were considered, where the objects are placed either on a tabletop or inside a shelving unit. The following methods were evaluated:

- **Round Robin (RR)**: This base case comparison point corresponds to only a single arm operating in the workspace at any given time. The other arm stays stationary at a pre-specified safe position. Each arm then alternates taking turns completing their tasks, until all tasks have been completed.
- **Batch (BA)**: This method corresponds to the *batch* approach described in Section 4.2. The ordering of the tasks is determined randomly at the beginning of each experiment.
- **Incremental (IN)**: This method corresponds to the incremental approach as described in Section 4.3. The initial assigned task is determined randomly.

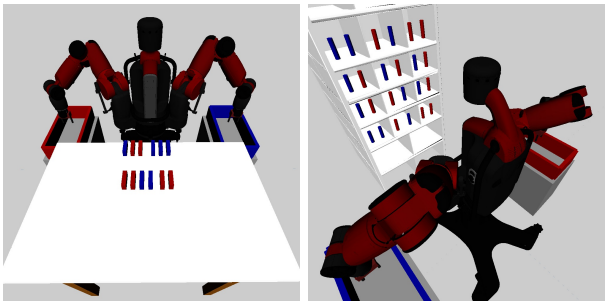

Figure 5: The objects colored red must be deposited in the red goal bin on the robot's right side. Similarly, the blue objects must be deposited in the blue bin on the left side. (Left) Table scenario with an example arrangement of objects. (Right) Shelf scenario with an example arrangement of objects.

The methods are then evaluated in terms of:

1) efficiency: measured as average online computation time and memory requirements, as well as
2) solution quality: measured as average duration of execution for solution trajectories.

Each simulated experiment was run on a single computer with an Intel Xeon E5-4650 2.8 gHz processor and 8GB of RAM.

Tabletop Results: The "tabletop" scenario, as shown in Figure 5 (Left), had 50 different randomized task assignments, i.e., potential object placements and corresponding arm assignments. A comparison of the average solution and computation time for all methods is shown in Figure 6. Since the schedule for the *batch* method was randomized, and since the initial task selection for the *incremental* method is also randomized, the numbers presented are averaged over a total of 1000 runs. The *round robin* method indicates both an upper and a lower limit for the performance of the methods considered in this paper. In particular, the best possible execution time that *batch* and *incremental* can return corresponds to half the duration of the round robin solution. And they cannot possibly exceed the time of the round robin solution.

The results show that in terms of execution time, both the batch and the incremental approach achieve

	4 objects		8 objects		12 objects	
	Batch	Incremental	Batch	Incremental	Batch	Incremental
# collision checks	655,306	409,566	2,544,669	1,431,376	5,686,158	3,080,021
computation time (s) without precomputation	114.2	71.2	443	261.65	1,000.6	532.2
computation time (s) with precomputation	0.08	0.05	0.35	0.18	0.7	0.25
execution time (s)	21.5	20.9	40.5	39.75	59.4	59

Table 1: Cost of Online Collision Checking in the Table Scenario

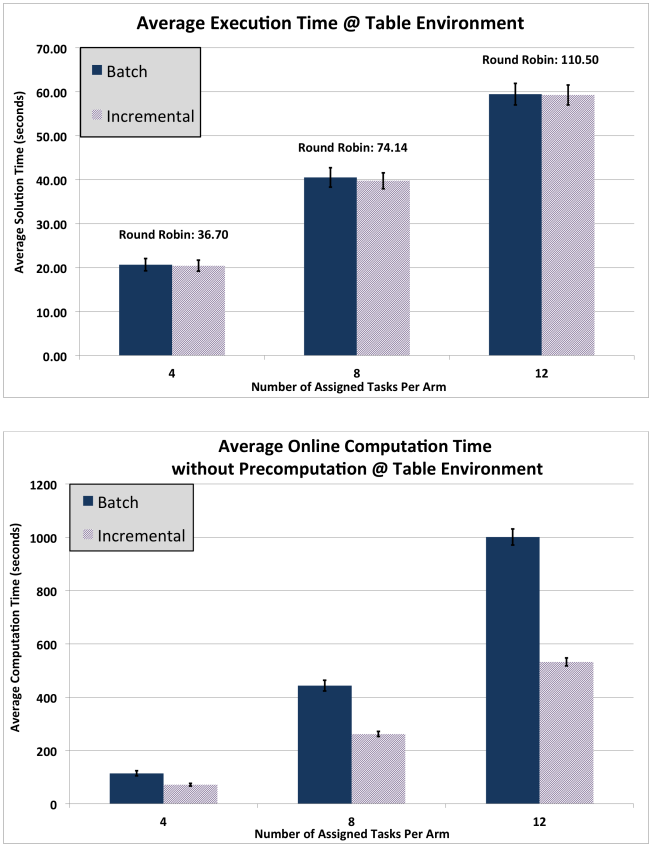

Figure 6: Comparison of average execution and computation times for 4, 8, and 12 tasks assignments per arm in the Table environment. Each bar is averaged over 50 different object placements, with the randomized approaches averaged over 1000 randomized orders per object placement. The 95% confidence interval for the randomized approaches is displayed.

times close to half of that of the round robin schedule. In terms of computation time, the *incremental* approach is faster than the *batch* one. In the results of Figure 6, the coordination diagrams of pairs of arm trajectories were not assumed precomputed. Thus, it is necessary to spend online computation time to identify the collision-free solution in the coordination space. Nevertheless, it is possible to achieve orders of magnitude faster solutions if the coordination diagrams are precomputed ahead of time. This can be done when the possible locations of objects which can appear are known ahead of time.

Table 1 provides a more detailed comparison of the computation cost of the two alternatives. An interesting statistic is how often the *batch* and *incremental* approaches queried the coordination diagram. In the case that a trajectory needs to be verified online, these calls correspond to collision checks, which are computationally expensive. The *batch* method makes significantly more collision checks than the *incremental* method. When the coordination diagrams are not precomputed, this is directly reflected in the online computation time of the two approaches. With precomputation, both methods can be computed orders of magnitude faster. In this case, the incremental method remains the faster out of the two.

Obviously preprocessing helps but at the cost of significant memory requirements in the case of the batch approach. In the above results, the number of objects considered was up to 12. It is interesting to evaluate what happens when a significantly higher number of objects is involved, which can happen in some industries where many small objects (i.e., electronic components) may need to be picked up by a robot. Figure 1 shows an example placement of a larger number of objects, while Table 2 shows the resulting RAM usage by all three methods for increasing numbers of objects in this scene. The *batch* method showed an enormous difference in the amount of memory used, and at 64 objects it was no longer able to solve the problem given a maximum of 6 GB of RAM. The memory requirements of the *batch* approach arise from the need to store the frontier nodes of the A* search over a significantly larger space.

	30 objects	**50 objects**	**64 objects**
RR	30.5 MB	41 MB	45 MB
BA	1.51 GB	4.21 GB	5.92+ GB
IN	43.7 MB	46 MB	47.1 MB

Table 2: RAM usage in the Table scenario for increasing numbers of objects and the three methods (RR: round robin, BA: batch, IN: incremental).

Shelf Results: The second setup, shown in Figure 5 (Right), considers an environment, which involves tight spaces. Rather than randomizing the placement of objects in the scene, the objects were instead placed in such a way so as to maximize the interactions between the two arms. To accomplish this, two objects were placed in the same bin side by side, and each was assigned to a different arm. A comparison of the average solution and computation time for all methods is shown in Figure 7. In this experiment, the online computation time considers the case where precomputed coordination diagrams are available. Examining these results shows that the *incremental* method provides improvements in both solution and computation time relative to the *batch* method and for precomputed coordination diagrams it is possible to achieve very fast solutions.

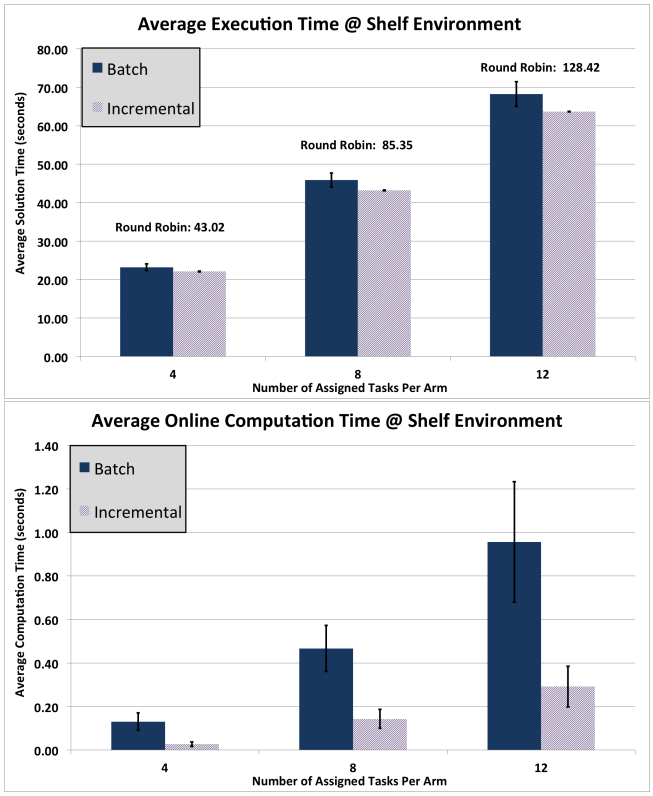

Figure 7: Comparison of average execution and computation times for 4, 8, and 12 tasks assignments per arm in the Shelf environment. To promote interaction between the arms, two objects are placed in the same bin. The numbers are averaged over 1000 different orderings. The 95% confidence interval for the approaches is displayed.

Dynamically Appearing Objects: Another important aspect is the performance of the methods when objects are allowed to appear dynamically during the execution of a schedule. To analyze this, the Shelf scenario was utilized so that once an object was placed at its goal bin, another object assigned to the same arm would appear in one of the empty shelves. The *batch* method was altered in the following way so as to support dynamically appearing objects: every time a new object appeared in the scene, both arms would finish placing their currently assigned objects and the schedule for the arms would be recomputed for the new set of objects. The *incremental* method did not have to be altered as it directly accommodates the presence of new objects. The results shown in Figure 8 measure total time, which takes into account both execution time and the amount of time each method takes to compute a schedule once a new object appears. From these results, the *incremental* method shows better performance in total time compared to the *batch* method. This is due to the fact that the *incremental* method computes a schedule on the fly, and can directly accommodate the dynamically appearing tasks.

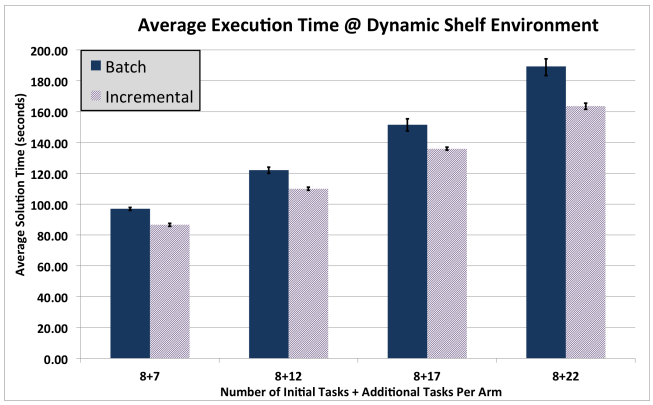

Figure 8: Comparison of average execution times in the Shelf scenario with increasing total number of dynamic objects appearing in the scene. The 95% confidence interval for each approach is displayed.

Precomputation Statistics: In the "tabletop" scenario, a roadmap with 4,600 vertices and 103,466 edges was generated. In the "shelf" scenario, a roadmap with 5,800 vertices and 104,138 edges was generated. The amount of time spent computing the trajectories and coordination diagrams is given in Table 3. Since the *round robin* method did not utilize the coordination graphs, this precomputation cost can be seen as the trade-off for achieving two orders of magnitude faster online computation times for *batch* and *incremental* scheduling.

	computation time (seconds)
12 Shelf Trajectories	15.7
24 Table Trajectories	23.4
"shelf" coord. graphs	2,176.7
"table" coord. graphs	5,024.2

Table 3: Trajectory and Coordination Matrix Computation Time

For picking tasks that appear infrequently, or only once, precomputation is not warranted and instead the summation of computation and execution time provides the best measure of effort for solving the tasks. When tasks appear repeatedly, whether in a static scene or in a dynamically appearing scene, precomputation can be taken advantage of to reduce the amount of time spent online. Once precomputation is utilized, the online computation time becomes significantly smaller than the execution time, and has a smaller impact on the overall time spent solving the task schedule. Nevertheless, in dynamic scenes, the batch method will need to recompute the schedule, which significantly increases its online execution time, whereas the incremental method can directly incorporate the newly added task into its existing search.

6 Discussion

This work examines the coordination and scheduling of dual-arm manipulators in the context of pick-and-place tasks. Three different methods for accomplishing this were evaluated. First, a naíve round-robin approach was examined that defines bounds for achieving coordination between the arms by ensuring that only one arm moves at any given time. Then, two approaches for allowing simultaneous arm movements are considered. A *batch* method, which searches over all pairs of *coordination diagrams* given an ordering of pick-and-place tasks. And an *incremental* approach, which dynamically assigns tasks to each arm, producing solutions of similar quality to the *batch* method for static scenes and improved performance in the presence of dynamically appearing objects. The *incremental* approach exhibits significant benefits, in terms of memory requirements as well as computation time, especially when precomputation cannot be taken advantage of.

In the current work none of the objects occludes any other object in the scene. Thus, the task ordering can be freely defined by the scheduler so as to improve performance. An interesting direction is to extend the method for solving problems where objects occlude one another, where some form of object rearrangement will also be required. This would define an implicit, potentially partial, ordering of the tasks, such that certain tasks must be completed before other tasks would even be feasible. Similarly, it is interesting to consider challenges where the arms need to perform handoffs in order to solve the problem (Cohen, Phillips, and Likhachev (2014)), when objects and their target placements cannot be reached by both arms.

Another interesting direction is scheduling in the context of multiple humanoid robots with overlapping tasks. In this case, an object might be graspable by more than one robot. Solving such a challenge would involve a task assignment sub-challenge for multiple robots and reasoning over a *generalized coordination diagram* (Siméon, Leroy, and Laumond (2002)). It should still be possible to apply the methodologies presented here to obtain an incremental schedule in this context.

Accounting for uncertainty due to sensing and noisy actuation are essential steps to consider when implementing the method on real robots. In this context, it is important to consider in terms of schedules that provide robust solutions. One potential way to adopt the coordination diagrams to account for noisy actuation is to artificially expand the collision region, thereby producing more conservative trajectories.

References

Akella, S., and Hutchinson, S. 2002. Coordinating the motions of multiple robots with specified trajectories. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, 624–631. IEEE.

Alami, R.; Laumond, J.-P.; and Siméon, T. 1997. Two Manipulation Planning Algorithms. In Laumond, J.-P., and Overmars, M., eds., *Algorithms for Robotic Motion and Manipulation*. Wellesley, MA: A. K. Peters.

Alami, R.; Siméon, T.; and Laumond, J.-P. 1989. A Geometrical Approach to Planning Manipulation Tasks. In *Proc. of International Symposium on Robotics Research*, 113–119.

Aronov, B.; de Berg, M.; van den Stappen, A. F.; Svestka, P.; and Vleugels, J. 1999. Motion Planning for Multiple Robots. *Discrete and Computational Geometry* 22(4):505–525.

Berenson, D.; Srinivasa, S. S.; and Kuffner, J. J. 2012. Task Space Regions: A Framework for Pose-Constrained Manipulation Planning. *The International Journal of Robotics Research (IJRR)* 30(12):1435–1460.

Chang, R.-S.; Chang, J.-S.; and Lin, P.-S. 2009. An ant algorithm for balanced job scheduling in grids. *Future Generation Computer Systems* 25(1):20–27.

Cohen, J. B.; Chitta, S.; and Likhachev, M. 2010. Search-based Planning for Manipulation with Motion Primitives. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Cohen, B.; Phillips, M.; and Likhachev, M. 2014. Planning single-arm manipulations with n-arm robots. In *Proceedings of Robotics: Science and Systems*.

Ellekilde, L.-P., and Petersen, H. G. 2013. Motion planning efficient trajectories for industrial bin-picking. *The International Journal of Robotics Research* 32(9-10):991–1004.

Gerkey, B. P., and Matarić, M. J. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research* 23(9):939–954.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. 1996. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

King, J.; Klingensmith, M.; Dellin, C.; Dogar, M.; Velagapudi, P.; Pollard, N.; and Srinivasa, S. S. 2013. Pregrasp Manipulation as Trajectory Optimization. In *Robotics: Science and Systems (RSS)*.

Lozano-Pérez, T., and Kaelbling, L. P. 2014. A constraint-based method for solving sequential manipulation planning problems. In *Intelligent Robots and Systems (IROS*

2014), 2014 IEEE/RSJ International Conference on, 3684–3691. IEEE.

Maheswaran, M.; Ali, S.; Siegal, H.; Hensgen, D.; and Freund, R. F. 1999. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, 30–44. IEEE.

O'Donnell, P. A., and Lozano-Perez, T. 1989. Deadlock-free and collision-free coordination of two robot manipulators. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, 484–489. IEEE.

Panescu, D., and Pascal, C. 2014. A constraint satisfaction approach for planning of multi-robot systems. In *System Theory, Control and Computing (ICSTCC), 2014 18th International Conference*, 157–162. IEEE.

Siméon, T.; Laumond, J.-P.; Cortés, J.; and Sahbani, A. 2004. Manipulation Planning with Probabilistic Roadmaps. *International Journal of Robotics Research (IJRR)* (23).

Siméon, T.; Leroy, S.; and Laumond, J.-P. 2002. Path coordination for multiple mobile robots: A resolution-complete algorithm. *Robotics and Automation, IEEE Transactions on* 18(1):42–49.

Tang, F., and Parker, L. E. 2007. A complete methodology for generating multi-robot task solutions using asymtre-d and market-based task allocation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 3351–3358. IEEE.

Vahrenkamp, N.; Berenson, D.; Asfour, T.; Kuffner, J.; and Dillmann, R. 2009. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2464–2470. IEEE.

Xidias, E.; Zacharia, P. T.; and Aspragathos, N. 2010. Time-optimal task scheduling for two robotic manipulators operating in a three-dimensional environment. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 224(7):845–855.

Xu, Z.; Hou, X.; and Sun, J. 2003. Ant algorithm-based task scheduling in grid computing. In *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, volume 2, 1107–1110. IEEE.

Yan, H.; Shen, X.-Q.; Li, X.; and Wu, M.-H. 2005. An improved ant algorithm for job scheduling in grid computing. In *Proc. of 2005 Int. Conf. on Machine Learning and Cybernetics, 2005.*, volume 5, 2957–2961. IEEE.

Zhang, Y., and Parker, L. E. 2013. Multi-robot task scheduling. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2992–2998. IEEE.

Zurawski, R., and Phang, S. 1992. Path planning for robot arms operating in a common workspace. In *Industrial Electronics, Control, Instrumentation, and Automation, 1992. Power Electronics and Motion Control., Proceedings of the 1992 International Conference on*, 618–623. IEEE.