A Hierarchical Approach to Manipulation with Diverse Actions

Jennifer Barry[1] Leslie Pack Kaelbling[1] Tomás Lozano-Pérez[1]

***Abstract*— We define the Diverse Action Manipulation (DAMA) problem in which we are given a mobile robot, a set of movable objects, and a set of diverse, possibly non-prehensile manipulation actions, and the goal is to find a sequence of actions that moves each of the objects to a goal configuration. We show that the DAMA problem can be framed as a multi-modal planning problem and describe a hierarchical algorithm that takes advantage of this multi-modal nature. We also extend our earlier forward search sampling algorithm to a bi-directional version. We give results on a complicated manipulation domain and demonstrate that both new algorithms are significantly more efficient than the original, and that the hierarchical algorithm is usually much more efficient than the forward or bi-directional searches.**

I. INTRODUCTION

Consider a personal robot performing chores in a home. To put a book away in a bookcase, it must consider the placement of the book when planning to pick it up, choosing a grasp that will later allow it to slide the book onto a shelf. To clear a table, the robot must manipulate plates and platters that are too flat for grippers to slide under. To remove a toy from under the couch, the robot must use a stick as a tool.

To behave well in the world, a robot must be able to reason about manipulation with diverse actions. These actions all constrain the robot differently, but a later action may depend on an earlier action. For instance, in order to grasp a plate, a robot might first push it to the edge of a table. The constraints on the grasping action had to inform the pushing action.

We give a general formulation of a manipulation problem with diverse actions. The key aspect of this problem is that multiple types of possibly non-prehensile manipulation are available for a single object. We show that this problem is naturally a *multi-modal problem* [1], in which the system moves between a set of modes as well as a set of configurations. A mode describes a set of configurations, usually a low-dimensional subspace of the full configuration space. We use the multi-modal characterization to automatically identify subgoals in a manipulation problems.

In this paper, we present our hierarchical algorithm, DARRTH, for the multi-modal manipulation problem. DARRTH uses as a sub-planner the previously presented DARRT algorithm [2] or a new planner, DARRTCONNECT, that solves the problem bi-directionally. We show that in a mobile manipulation domain, the bi-directional search is significantly faster than the forward search, and the hierarchical algorithm is often significantly faster than either search algorithm.

[1]MIT CSAIL, {jbarry, lpk, tlp}@csail.mit.edu

This work was supported in part by the NSF under Grants No. 1117325 and No. 1122374. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge support from ONR MURI grant N00014-09-1-1051, from AFOSR grant FA2386-10-1-4135 and from the Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center. We thank Willow Garage for the use of the PR2 robot as part of the PR2 Beta Program.

II. RELATED WORK

There is a large body of work on manipulation actions for us to draw upon. Mason [3] discusses the mechanics of pushing an object, while Brost [4] and Dogar and Srinivasa [5] propose to combine pushing and grasping in a push-grasp, and Huang and Mason investigate striking or tapping objects [6]. However, these papers focus on describing and simulating the dynamics and control of a specific action. When they address planning, they emphasize planning for this single action type. We take a different view; we build on this existing work to model a set of diverse action types and focus on combining them to generate complex plans.

A related problem is navigation among movable obstacles (NAMO) [7]–[11]. In this problem, there are multiple movable objects in the world and the robot can move each of them. Much of the work in this domain assumes that an object is only moved once and that moving a single object is an easy task. However, van den Berg et al. [11] relax these assumptions and solve for a set of paths for each object in the domain and then for the corresponding robot trajectory. To ensure that object paths are executable by the robot, they require that the object remain *manipulable* along the path. In their work, an object is manipulable if it has some point in the current connected component of the robot's configuration space. Deciding manipulability is computationally intractable for high dimensional state spaces. However, we will plan object-level paths with approximated manipulability before trying to plan in the full state space (Section V-B).

While the NAMO problem usually assumes rigid grasping and simple object dynamics, work on moving an object in clutter focuses on non-prehensile manipulation and the effects of objects on other objects. Dogar and Srinivasa [12] consider the problem of trying to move an object in clutter and have a library of manipulation actions, including non-prehensile actions, but assume each object or piece of clutter is moved only once using a single manipulation action. Cosgun et al. [13] discuss trying to place an object on a cluttered surface. They assume only the object to be placed is grasped, but that this object can push other objects out of the way. Multiple objects can be moved at once, but this still incorporates only a single type of manipulation action.

The re-grasping problem [14], [15] requires multiple manipulations for a single object. The framework proposed by Siméon et al. (2004) breaks the problem into finding *transits*

for the robot moving alone and *rigid-transfers* for the robot and object. They note that any switch between transit and rigid-transfer must occur in the part of the configuration space where the object is sitting stably and the robot is grasping the object, which we will refer to as PG. They prove the "reduction property" that any two points in the same connected component of PG can be connected by a finite number of transits and rigid-transfers. This allows them to plan for re-grasping tasks by first planning a path between the connected components of PG and then planning for each rigid-transfer or transit individually, a method we will call PG-Map. Although the reduction property relies on the grasped object being able to move instantaneously in any direction and does not hold for non-prehensile manipulation, we will build upon the idea of placing subgoals at the points where actions can change.

We frame the problem of planning with diverse manipulation actions as a multi-modal planning problem. Hauser [1] defines a multi-modal planning problem as one in which the system moves between configurations and also among a set of *modes*. The mode space is part of the problem description and each mode describes a set of configurations that all satisfy certain mode-specific constraints. In his initial work, Hauser focused on problems with discrete mode spaces, but low-dimensional mode transitions. He showed how to create a two-level roadmap of modes and configurations using interspersed mode and configuration sampling. Later Hauser and Ng-Throw-Hing [16] extended this work to domains like manipulation where the mode space is continuous and were able to find paths for a walking robot pushing an object on a table. However, that work required the implementation of complicated mode samplers and a number of heuristics, some of which took substantial pre-processing time. Hauser and Ng-Throw-Hing do not show how to generalize their problem-specific framework to other manipulation problems and, as we will show in Section IV, the algorithms proposed for multi-modal problems are unable to solve some manipulation problems of interest.

III. PROBLEM DEFINITON

We address problems in which we have a robot, a set of movable objects, and a set of manipulation primitives. We begin by formally defining a manipulation primitive.

A. DAMA Problem

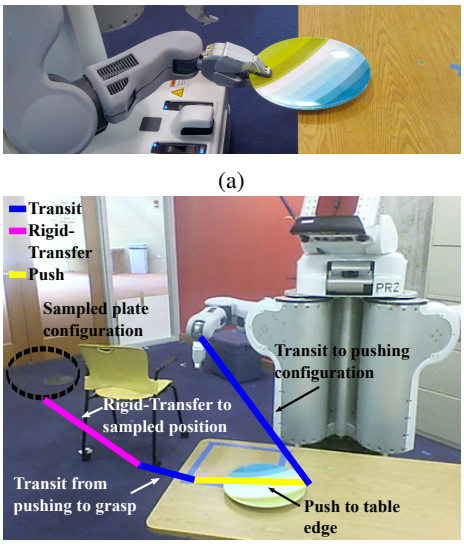
Definition: A *manipulation primitive* is a function that takes an initial configuration of the robot and movable objects and a displacement of the robot's configuration and returns a final configuration of the robot and objects. A primitive is *applicable* only to certain configurations and displacements and an *instance* of a primitive is an instantiation of the primitive with a specific set of arguments. Any primitive that moves an object is a *transfer primitive*.

Any type of manipulation can be represented as a manipulation primitive provided it is possible to describe the effect of the primitive on any given configuration of the robot and objects. For complicated primitives, it is possible that this would require computational integration of equations of motion, but we use simpler primitives. Throughout this paper we use the following primitives as examples:

- *Transit*: describes the robot moving alone. Transit is applicable to any configuration and displacement in which there is no collision between the robot and any objects or obstacles in the world.
- *Rigid-transfer*: describes the robot moving a rigidly attached object. Rigid-transfer is applicable to any configuration and displacement in which the robot is grasping the object and there is no collision between the robot with the attached object and any other objects or obstacles in the world.
- *Push*: describes the robot pushing an object. Push is applicable to any configuration in which there is two-point contact between the pushing end-effector and the pushed object and a collision-free displacement that moves the end-effector along the line connecting the end-effector with the center of the object.

A path using these primitives to move a plate off of a table is shown in Figure 1b. More examples of primitives will be given in Section VI.

The DAMA problem was informally introduced in Barry et al. [2], but here we give a formal definition.



(a)

(b)

Fig. 1: An example world with Transit, Rigid-Transfer and Push. (a) If the robot can only grasp the plate when it is at a single point on the edge of the table, this is a zero-measure subset of the configurations in which the plate is on the table. (b) An extension from the state shown in the photograph towards the sample shown with the dashed lines. Samples specify only partial states; here the sample specifies a configuration for the plate. The sequence of primitives shown for the wrist first transits the robot to a pushing configuration (blue), pushes the plate towards the edge of the table (yellow), transits the robot to a grasp (blue), and finally transfers the plate to its sampled configuration (magenta).

Definition: The *Diverse Action MAnipulation (DAMA)* problem is a tuple $\langle R,\ \{o_1, ..., o_n\},\ \{B_1, ..., B_q\},\ \{p_1, ..., p_m\},\ c_I,\ G\rangle$ in which R is a robot, $\{o_1, ..., o_n\}$ is a set of movable objects, $\{B_1, ..., B_q\}$ is a set of fixed obstacles, $\{p_1, ..., p_m\}$ is a set of manipulation primitives, c_I is an initial configuration, and G is a set of goal configurations. The goal set may be infinite in size. For example, in manipulation, goal configurations are often specified only for objects. The goal set is any configuration in the combined space in which the objects are in their goal configurations. The *movable components* are the robot plus the set of objects, $\{R, o_1, ..., o_n\}$. A solution is a sequence of primitive instances that takes c_I to a configuration in G.

In this paper, we assume that the environment is uncluttered enough that it is not necessary to focus on the order in which to move objects. The problem of rearranging objects in cluttered environments has been thoroughly addressed [7]–[11], [17], and is not the focus of this paper. It may be possible to combine the approaches to NAMO with our approach to solve a more general manipulation problem, but that is left for future work.

Having formally defined the DAMA problem, we can now define its multi-modal counterpart.

B. Multi-Modal DAMA Problem

A *multi-modal* problem is a tuple $\langle C, \Sigma\rangle$ where C is a configuration space and Σ is a mode space [1]. Each mode $\sigma \in \Sigma$ defines a set of mode-specific constraints that in turn define a set of configurations that satisfy those constraints. A *state*, (c, σ) in a multi-modal problem specifies both the current configuration c and the mode σ. In general this formulation is useful when some modes describe lower-dimensional subspaces of the full configuration space. For instance, in legged locomotion, the modes are a fixed set of footfalls. The footfalls constrain the feet to be on the ground, and walking must transition through these footfalls.

Multi-modal problems were originally formulated for systems with discrete modes. For adaptation to continuous-mode problems, such as the DAMA problem, we follow Hauser [16] and describe the set of continuous modes as a finite, discrete set of *mode families*. Mode families partition a continuous mode set using a co-parameter that varies to describe each of the different modes. Transitions between modes within a mode family are disallowed; modes must first transition out of the family. For instance, in the NAMO problem, there are n movable obstacles. This problem has $n + 1$ mode families: one for each obstacle and one for the robot moving alone. The modes of each family are the configurations of all non-moving obstacles.

Definition: Let the DAMA problem be $\langle R, \{o_1, ..., o_n\}, \{B_1, ..., B_q\}, \{p_1, ..., p_m\}, c_I, G\rangle$. Let primitive p_i be able to manipulate sets of objects $\{O_1, ..., O_j\}$. For each primitive p_i and each object set O_k, we define one *mode family* F_{ik} that corresponds to primitive p_i manipulating set O_k. A *mode* is a relative configuration of the robot and objects in O_k and a stationary configuration of the objects not in O_k. The *MM-DAMA* problem is the DAMA problem augmented by this mode space. An MM-DAMA problem instance can be automatically constructed from a DAMA problem instance.

Consider the example world shown in Figure 1, in which a robot manipulates a plate using the Transit, Rigid-Transfer and Push primitives. This world has three mode families, one for each of the primitives. Within each mode family, there is a continuum of modes with a set of parameters that we can vary. For instance, in Transit the robot moves and the plate remains stationary so the modes of the Transit family correspond to the different configurations of the plate. In Rigid-Transfer, the robot and plate both move but the plate must be grasped so the individual modes correspond to specific grasps. Similarly, the modes in Push correspond to different pushing configurations. If there was also a bowl in the world that could be grasped but not pushed, we would have four mode families: Transit, Rigid-Transfer-Plate, Rigid-Transfer-Bowl, and Push-Plate. Note that it is impossible to transition between modes within the same family. For instance, to change the grasp (Rigid-Transfer family), the robot must pass through Transit.

IV. MULTI-MODAL PLANNING

In this section, we first review the explicit multi-modal planning work [1], [16], showing that it cannot be used for some manipulation problems, and then discuss using the multi-modal structure to create hierarchies.

A. Explicit Multi-Modal Planning

Hauser [1] originally proposed the multi-modal framework and also several sampling-based algorithms for solving multi-modal problems. Those algorithms require an extra piece of information specifying not only the modes of the problem but also a high-level mode graph guiding the possible mode transitions. This graph may have transitions that cannot occur in actuality, but it must describe all possible transitions. The algorithms then rely on the following extension step:

1) Sample a configuration in the intersection of two modes from the mode-transition graph
2) Plan a collision-free, feasible path within a single mode to reach this transition

However, because these algorithms randomly sample mode transitions, they can never find solutions that involve moving through low dimensional subspaces within the mode transitions themselves. Manipulation problems often require moving through these subspaces. For example, consider the problem shown in Figure 1a. As discussed in Section III-B, the mode families in this problem are Transit, Push, and Rigid-Transfer. Transit modes can transition to and from Push and Rigid-Transfer modes, but Push and Rigid-Transfer cannot transition to each other because there is no configuration in which the robot is simultaneously in a pushing configuration and a grasp. Additionally, assume for the purposes of this example that there is only a single pose on the table where the plate can balance on the edge. In order to transition from Push to Rigid-Transfer, the robot must push the plate to this point and then Transit to a grasp as shown in Figure 1a. However, the intersection of Push

and Transit is any configuration in which the plate is on the table and the robot is pushing it. Therefore, there is zero probability that a configuration in which the plate is positioned at a single point on the table edge can be sampled at random from the intersection of Push and Transit, and any algorithm that only samples randomly from the mode-transition graph cannot solve the problem. Other examples include problems like placing a book on a shelf in which only one grasp of a continuum can be used or positioning a tool in a specific way for later use. More generally, manipulation problems require reasoning about long chains of related transfer primitives such as Tool-Use and Pick or Push and Rigid-Transfer. Because transfer primitives almost always transition only to and from transit primitives, sampling from the intersection of two modes alone does not capture this multi-step dependency. Since our focus is on manipulation problems, we do not use the algorithms initially proposed for multi-modal problems.

B. Multi-Modal Planning for Manipulation

However, many solutions to specific manipulation problems rely on the multi-modal nature. For example, in Section II, we discussed two manipulation problems: re-grasping and NAMO. Both of these problems are subsets of the MM-DAMA problem defined in Section III. For re-grasping, there are two mode families: Rigid-Transfer and Transit, and we discussed the modes and mode families of NAMO in Section III-B. The solutions to these problems discussed in Section II focus on identifying configurations in which the mode is likely to change, such as the intersection of grasp and place or the point at which to change the obstacle being moved. These are both examples of an idealized algorithm for solving the MM-DAMA problem:

1) Plan a sequence of modes.
2) Plan for each mode in the sequence.

However, only PG-map and the version of NAMO proposed by van den Berg et al. [11] are actually able to perform these steps exactly, and both algorithms rely on describing connected components of the robot's configuration space. PG-map is able to use the holonomic robot and rigid-grasp aspects of the problem to describe PG while van den Berg et al. assume it is possible to analytically describe all connected components of the robot's configuration space. Because we allow non-prehensile manipulation and have potentially high dimensional configuration spaces, we will not be able to apply the same leverage to the problem.

When the connected components of the robot's configuration space cannot be described, effective solution methods focus on finding a sequence of mode families in Step 1 rather than a sequence of modes. The difference is that of knowing the correct primitive to apply rather than knowing the exact configuration in which to apply it. For example, if we know that we must find a transition between Transit and Rigid-Transfer, we know the sequence of mode families. If, however, we know the specific grasp to use in the Rigid-Transfer, we know the sequence of modes. We also simplify the problem in this way, modifying the algorithm to:

1) Plan a sequence of mode families
2) Plan within each mode family

Moreover, in manipulation, transfer mode families are rarely able to transition to one another because this requires a configuration that could simultaneously be used for two different types of transfer. Therefore, we know that a transit must occur between every type of transfer and the interesting ordering is within the ordering of transfers. Thus, we simplify the algorithm even further:

1) Plan a sequence of transfer mode families
2) Plan each set of transfer and transit trajectories

Throughout the rest of this paper we assume a single object. If the domain is uncluttered, the algorithm can be repeated for each object. Otherwise, we can combine our algorithms for manipulation with the NAMO work to first find a candidate order in which to move objects and then plan the manipulations for each object individually.

In the next section, we describe a hierarchical algorithm for solving general DAMA problems.

V. DARRTH ALGORITHM

A common theme in manipulation planning is to plan a path for the object first and then use information from that plan to guide the search for a full path [7]–[11], [17], [18]. In our hierarchical approach to manipulation, we also first try to identify an object path. We then use that path to find a sequence of transfer mode families and adapt that into subgoals. Because we must solve smaller DAMA problems to achieve each subgoal, we first review the DARRT algorithm for solving DAMA problems. We also take this opportunity to present the DARRTCONNECT algorithm, the bi-directional version of DARRT.

A. DARRT *and* DARRTCONNECT *Algorithms*

The Diverse Action Rapidly exploring Random Tree (DARRT) algorithm has the structure of a rapidly exploring random tree (RRT) with controls [19]. However, we modify both the extension and sampling methods to work with manipulation. These modifications are shown in Algorithm 1. We summarize them briefly below, but for a full discussion see Barry et al. [2]. Full pseudo-code for DARRT and DARRTCONNECT is given in the appendix.

When sampling the space with DARRT, we sample partial configurations that specify only configurations for objects or configurations for the robot. This is shown in the SAMPLE method in Algorithm 1. When extending with DARRT, instead of using the more usual single-step extension, we instead try to extend the tree by finding a path all the way from the state in the tree to some state that satisfies the partially sampled state. This extension algorithm is shown in the EXTEND method in Algorithm 1. To make this tractable, the path found by EXTEND ignores any collisions. We truncate this path to its first collision and add it to the tree. Our implementation of the method for finding paths ignoring collisions is shown in PATH. An example of an extension is shown in Figure 1b.

Algorithm 1 Functions for DARRT and DARRTCONNECT. Parameters: M: movable components, robot R and objects $\{o_1, ..., o_n\}$, B: fixed obstacles, A: manipulation primitives

SAMPLE(M)
1 $S \leftarrow$ randomChoice$\big(\{R\}$, randomSubset$(\{o_1, ..., o_n\})\big)$
2 $r \leftarrow$ random configuration for each m_i in S
3 **return** r

EXTEND(c_1, c_2, M, B, A, F)
// c_1 is the state in the tree and c_2 is the sampled state.
*// $F =$ **True** when extending forwards, **False** backwards.*
1 **if** F: $e \leftarrow$ PATH(c_1, c_2, M, A)
2 **else** : $e \leftarrow$ PATH(c_2, c_1, M, A)
3 $\{e_1, ..., e_l\} \leftarrow$Discretize$(e)$
4 **if** $\neg F$: reverse$(\{e_1, ..., e_l\})$ *// Backwards from goal tree*
5 **for** e_i, **if** collision(e_i, M, B), **return** $\{e_1, ..., e_{i-1}\}$
6 **return** $\{e_1, ..., e_l\}$

PATH(c_1, c_2, M, A)
1 **if** $c_1 = c_2$: **return** $\{\}$
2 $p \leftarrow$ randomUsefulPrimitive(c_1, c_2, M, A)
3 $P \leftarrow$ propagate(p, c_1, c_2, M)
4 $c \leftarrow$ state after applying P to c_1
5 **return** $P \cup$ PATH(c, c_2, M, A)

Barry et al. presented only the RRT-based DARRT algorithm, but we can modify that into a bi-directional algorithm based on the RRT-Connect algorithm described by Kuffner and LaValle [20]. The extension method used for DARRT lends itself to a bi-directional algorithm as an extension from configuration c_1 towards c_2 involves computing a path from c_1 all the way to c_2. Therefore, we do not need to work out a system of inverse control for our primitives. Instead, when extending "backwards" from an ending state c_1 towards a starting state c_2, we simply reverse the order of the arguments to PATH. PATH returns a path from c_2 to c_1 that we discretize and then check for collisions in the reverse order. The result is a valid path backwards from c_1 towards c_2. This is shown in the EXTEND method when F is false.

There is one more subtlety in this modification, however, because we sample only partially specified states. Therefore EXTEND expects that c_2 may be partially specified but that c_1 is an exact state. When we exchange the order of the arguments, the reverse will be true. The primitives' USEFUL and PROPAGATE functions must be able to accommodate partial specifications in the source or the goal state. We have found in practice that this is an easy adaptation [21].

We show in Section VI that DARRTCONNECT is much more efficient than DARRT. We can use either to solve subproblems for DARRTH.

B. Finding an Object Path

van den Berg et al. [11] give a method for planning a valid object path using two criteria:

1) The path is collision-free for the object.

2) The object is manipulable at all points along the path.

van den Berg et al. define "manipulable" to mean that the object is adjacent to the robot's current connected component in configuration space. However, this definition relies on the robot being able to grasp the object at any point of contact and the object being able to sit stably everywhere in its configuration space. Since we do not make these assumptions, we must modify the definition of manipulable:

Definition: An object configuration c_o is *manipulable with displacement* d_o if there is some configuration c in the robot's connected component, displacement d and a primitive applicable at c and d that results in object displacement d_o.

Using this definition of manipulable, the above criteria for a valid path still applies.

Exactly calculating valid paths requires characterizing the connected components of the robot's configuration space and the possible applicable primitives at each configuration and displacement along the path. Even were this computationally feasible, it would be very likely harder than solving the original problem. However, we cannot ignore manipulability entirely because the object paths must obey the constraints of the manipulation primitives. For instance, in the world shown in Figure 1, while the plate is on the table, only Push is applicable. The plate cannot rise straight up off the table; it must first be pushed to the edge of the table and, since we are interested in the order of transfer primitives, we want the path for the object to reflect this constraint.

Thus, we plan object paths using DARRT(CONNECT) but only checking collisions between the object and the environment. These paths fulfill the first condition of a valid path, but only approximately satisfy manipulability. Although we ensure that there is some configuration for the robot that can move the object along the path, we do not ensure that this configuration is collision-free or that it is in the robot's current connected component.

C. Manipulation Primitive Subgoals

We use the object path to find a set of subgoals. Because we use DARRT(CONNECT) to solve for an object path, the object paths are annotated with the manipulation primitive used. Therefore, we can convert immediately from an object path to a sequence of manipulation primitives. Recall from Section IV that with only one object, the manipulation primitive used is the same as the mode family. Moreover, since only transfer primitives move an object, the object path will consist entirely of transfer primitives.

Therefore, the object path defines a sequence of subgoals $g_1, ..., g_S$ corresponding to the transfer primitives $p_1, ..., p_S$ used along the path. The subgoal g_i is the set of configurations for which the primitive p_i is applicable for some displacement. We refer to this as a *primitive subgoal*.

Analytically describing the set of configurations and displacements in which a primitive p_i is applicable is not necessarily tractable. However, our algorithms do not require an analytical description. For DARRT, we must just be able to decide whether or not a primitive is applicable at a configuration and displacement. Because we can label states

Algorithm 2 *Input*: M: movable components, robot R and object o, B: fixed obstacles, A: manipulation primitives, c_I: initial configuration, G: goal for o, N: Max flat tries
Output: A path from c_I to a configuration in G

DARRTH(CONNECT)(M, B, A, c_I, G)
1 **while** no solution has been found
  // Only check object collisions
2 $\omega \leftarrow$ DARRT(CONNECT)(M, B, A, c_I, G)
3 $\{\pi_1, ..., \pi_k\} \leftarrow$ Manipulation primitives on ω
4 $c \leftarrow c_I$
5 **for** $g \in \{\pi_1, ..., \pi_k, G\}$
6 **while** no solution and #attempts $< N$
7 $\tau_i \leftarrow$ DARRT(CONNECT)(M, B, A, c, g)
8 $c \leftarrow$ last configuration along τ_i
9 **return** $[\tau_1, ..., \tau_{k+1}]$

with the primitive used in the PATH method, when trying to achieve subgoal g_i, we simply check whether the primitive used in the state was p_i. When using DARRTCONNECT to achieve a primitive subgoal, we also need to sample from the goal set. To sample from the set of configurations and displacements for which primitive p_i is applicable, we use the PATH method to create paths from the starting state of the current subgoal to a random configuration until we find a path that uses p_i. We then return the state in which p_i is first used along the path.

D. Hierarchical Algorithm

Pseudo-code for DARRTH(CONNECT) is shown in Algorithm 2. We first generate an object path using DARRT(CONNECT) but only checking collisions for the object. We then identify the sequence of transfer primitives used along this path and, for each primitive, run DARRT(CONNECT) until we achieve a configuration in which that primitive is applicable. Lastly, we solve for the final goal set.

Because we only approximate the validity of object paths, we cannot guarantee that the sequence of transfer mode families found from the object path is correct. Therefore, if we are unable to find a solution for a subgoal, we do eventually restart the entire algorithm. Thus we must specify two restart conditions: The number of iterations after which to restart a DARRT(CONNECT) run and the number of DARRT(CONNECT) runs after which to restart the entire algorithm. We chose these numbers empirically for each problem, but found that we rarely required more than one iteration of DARRTH.

VI. RESULTS

We ran DARRT, DARRTH, DARRTCONNECT and DARRTHCONNECT on a set of problems in a challenging manipulation domain on the PR2 robot. We planned for the PR2's three degree of freedom holonomic base, one of its seven degree of freedom arms, and a rigid object for a total of 16 dimensions in our configuration space. In these problems, the PR2 had to maneuver to a table, push a plate to the edge of the table, grasp the plate and transfer it to somewhere else in the domain. The domains are shown in Figure 2 and the primitives implemented were:

- Base/Arm-Transit: Transit for the arm and base.
- Base/Arm-Rigid-Transfer: Rigid-transfer for the arm and the base. There were a finite number of grasps.
- Straight-Line-Arm-Transit: Moves the gripper in a straight line in Cartesian space. Used for approaching, retreating from and lifting objects.
- Push: The robot can push the plate when its gripper is in two-point contact with the plate. This primitive is non-prehensile.

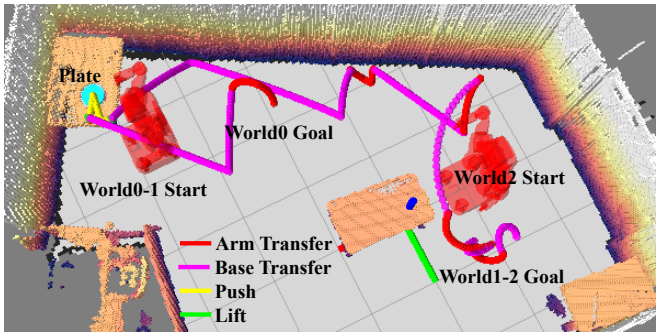
Sample trajectories are shown in Figure 2 and overall planner results are given in Table I. Videos of trajectories are on our website[2]. DARRTH(CONNECT) identified three subgoals in these domains: Push, Rigid-Transfer and achieving the actual goal pose. Intuitively, these subgoals correspond to how a human might break down the problem: 1) Move to the table, 2) Push the plate to the edge of the table, and 3) Grasp the plate and move it to the goal. The time taken for each subgoal, as well as the time required for planning the object path, is shown for the hierarchical planners in Table II.

For comparison's sake, to show that the planning times are within the realm of reason, we also found an approximate lower bound on planning time using a planner for which we specified the exact modes by hand. For the mode-specified planner, we give the robot base and arm positions at every mode switch along the path and then use out-of-the-box planners to find a plan for the base or arm alone. We gave the mode-specified planner at least 11 waypoints in each problem: 1) base position at table, 2) arm position at approach to pushing, 3) arm position while pushing, 4) arm position after pushing, 5) arm position at retreat from pushing, 6) base position for picking up the object, 7) arm position for approaching the grasp, 8) arm position in the grasp, 9) arm position after lifting the object, 10) base position at the goal pose, and 11) arm position at the goal pose. Note that the base and arm planners are planning in three and seven dimensional spaces respectively while the DARRT and DARRTH variants make an entire plan in sixteen dimensional space. This mode-specified planner is shown in the last column of Table I. We did not run it on World 4 as that was a world designed to show a weakness of the hierarchical planner.

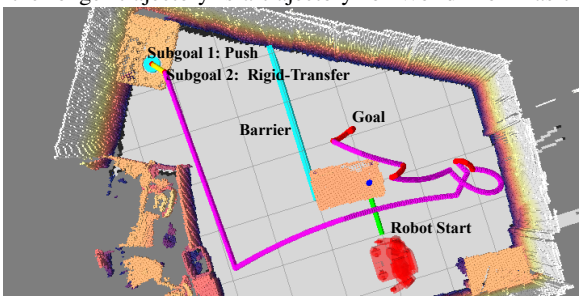
VII. DISCUSSION

The bi-directional planner is always faster than the forward planner, as expected, and in all but two domains the hierarchical planners are faster than their flat counterparts. Except in World 4, which we will discuss in detail, DARRTHCONNECT is not unreasonably slower than the mode-specified planner, even though the latter has much more information.

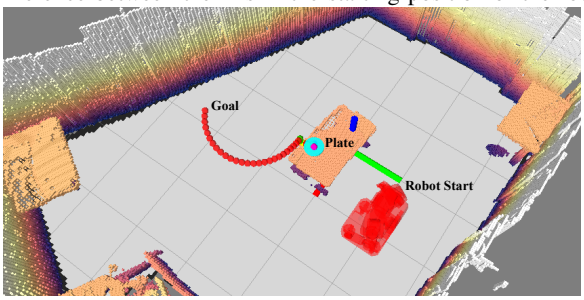
[2]http://people.csail.mit.edu/jbarry/pr2/darrt



(a) Worlds 0, 1, and 2 all began or ended in similar places so are grouped in this figure. The shorter trajectory is a trajectory in World 0 while the longer trajectory is a trajectory for World 1 or 2 as the only difference between them is in the starting position of the robot.



(b) World 3. The robot cannot cross the cyan barrier.



(c) World 4

Fig. 2: The domains in which we ran the algorithms. There is a plate (cyan cylinder) on a table and the goal is to move it to somewhere else in the environment. The robot starting state is shown in red and the final trajectory is shown color-coded by the primitive used. The trajectory is only shown for the plate for visual clarity, but the plans were for robot and object. The subgoals are labeled in World 3 and were the same for all five problems. Videos of trajectories are on our website[2].

Domain	DARRT		DARRTConnect		Mode-Specified
	DARRT (s)	DARRTH (s)	DARRTConnect (s)	DARRTHConnect (s)	(Lower Bound)
World 0	12	14	**11**	19	13
World 1	42	**25**	34	28	14
World 2	142	65	98	**36**	19
World 3	1004	171	436	**61**	36
World 4	411	218	**165**	240	–

TABLE I: Overall planning time (wall time in seconds) averaged over 50 runs. DARRTH is the hierarchical algorithm using DARRT as a flat planner and DARRTHCONNECT is the hierarchical algorithm using DARRTCONNECT as a flat planner.

Domain	DARRTH				DARRTHConnect			
	Object	S1 (Transit)	S2 (Push)	S3 (Rigid-Transfer)	Object	S1	S2	S3
World 0	5	1	4	5	3	1	12	3
World 1	6	1	4	15	3	1	12	12
World 2	8	9	8	41	4	7	16	9
World 3	7	15	10	140	3	9	8	42
World 4	8	19	59	132	4	32	201	3

TABLE II: Time taken to compute each subgoal. The Object column is the time taken to plan the object path while S1, S2 and S3 are the times taken to plan the intermediate subgoals using the flat planner. (Times are rounded to the nearest second so the sum of the results may not exactly equal the total time reported in Table I.)

World 0 is a trivial domain for which a single goal sample can solve the problem depending on the path chosen. Although the four sub-problems are also easy, solving them individually takes more time than solving the single problem.

World 4 was chosen to illustrate a weakness of the hierarchical bi-directional planner. In this domain it is possible to reach a Push configuration from the wrong side of the table. In this case, the robot's starting configuration when solving the second subgoal (pick up the plate) is actually worse than the original starting configuration. It is clear from the amount of time taken by DARRTHCONNECT on the second subgoal (pick up the plate) that it fell into this trap often. Its time for solving that subgoal alone is greater than DARRTCONNECT took to solve the entire problem. We expect that situations like this are rare in practice or could be alleviated by better goal sampling in DARRTHCONNECT.

From the subgoal breakdown, it is also clear that DARRTH did not face this problem. This is because the feet

of the table make the configurations in which the robot is on the wrong side of the table difficult to reach. Therefore, DARRTH does not often find such a configuration. However, for DARRTHCONNECT, when adding states to the goal tree, we continue to sample until we find a collision-free state. Therefore we were actively adding bad pushing configurations to the goal tree. Moreover, the bi-directionality helped the planner achieve these bad configurations.

Except in the world designed to be difficult for them, the hierarchical algorithms consistently out-perform their flat counterparts and DARRTHCONNECT plans only a factor of about two slower than the mode-specified planner. The leverage is owing primarily to two factors:

- Nearest Neighbor Calculation: The distance function for DARRT is costly because it must approximate manipulation paths. By resetting the tree every time a subgoal is achieved, DARRTH makes many fewer calls to the distance function.
- Targeted Restarts: As is common with sampling-based planners, we restart DARRT(H)(CONNECT) after a specified time period. With DARRT(CONNECT), we can do no better than restarting from the starting state. With DARRTH(CONNECT), however, we restart from the most recent subgoal. For example, lifting the plate off of the table is a difficult problem. If the move from the table to the goal configuration is also difficult, DARRT(CONNECT) might restart after having solved for lifting up the plate but before being able to transfer it to its goal location. DARRTH(CONNECT), however, by its automatic choice of good subgoals, restarts from the state in which the plate has been lifted.

The time taken for nearest neighbors is dependent on the distance function used. It has since been shown that a simpler distance function than the one used in this work is valid [21]. In this case, the impact of the fewer distance calculations would be lessened, bringing the hierarchical and the flat planning times closer together.

VIII. CONCLUSION

In this paper we formally defined the DAMA problem and its multi-modal counterpart, MM-DAMA. We gave three new algorithms for this problem (DARRTH, DARRTCONNECT, DARRTHCONNECT) and showed that all three are significantly more efficient than the original DARRT algorithm in a mobile manipulation domain.

REFERENCES

[1] K. Hauser, "Motion Planning for Legged and Humanoid Robots," Ph.D. dissertation, Stanford University, 2008.

[2] J. Barry, K. Hsiao, L. Kaelbling, and T. Lozano-Pérez, "Manipulation with Multiple Action Types," in *ISER*, 2012.

[3] M. T. Mason, *Mechanics of Robotic Manipulation*. Cambridge, MA: MIT Press, August 2001.

[4] R. C. Brost, "Automatic Grasp Planning in the Presence of Uncertainty," *IJRR*, vol. 7, no. 1, 1988.

[5] M. Dogar and S. Srinivasa, "Push-Grasping with Dexterous Hands: Mechanics and a Method," in *IROS*, 2010.

[6] W. Huang and M. T. Mason, "Experiments in Impulsive Manipulation," in *ICRA*, vol. 2, 1998.

[7] K. Okada, A. Haneda, H. Nakai, M. Inaba, and H. Inoue, "Environment Manipulation Planner for Humanoid Robots Using Task Graph That Generates Action Sequence," in *IROS*, 2004.

[8] M. Stilman and J. Kuffner, "Navigation Among Movable Obstacles: Real-Time Reasoning in Complex Environments," in *HUMANOIDS*, 2004.

[9] ——, "Planning Among Movable Obstacles with Artificial Constraints," *IJRR*, vol. 27, no. 11-12, 2008.

[10] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation Planning Among Movable Obstactions," in *ICRA*, 2007.

[11] J. van den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha, "Path Planning among Movable Obstacles: A Probabilistically Complete Approach," in *WAFR*, 2008.

[12] M. R. Dogar and S. S. Srinivasa, "A Framework for Push-Grasping in Clutter," in *RSS*, 2011.

[13] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push Planning for Object Placement on Cluttered Table Surfaces," in *IROS*, 2011.

[14] T. Lozano-Pérez, J. L. Jones, E. Mazer, and P. A. O'Donnell, *Handey: A Robot Task Planner*. Cambridge, MA: MIT Press, 1992.

[15] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation Planning with Probabilistic Roadmaps," *IJRR*, vol. 23, no. 7-8, 2004.

[16] K. Hauser and V. Ng-Throw-Hing, "Randomized Multi-Modal Motion Planning for a Humanoid Robot Manipulation Task," *IJRR*, vol. 30, no. 6, 2011.

[17] J. Ota, "Rearrangement of Multiple Movable Objects: Integration of global and Local Planning Methodology," in *ICRA*, vol. 2, 2004.

[18] S. S. Srinivasa, C. R. Baker, E. Sacks, G. B. Reshko, M. T. Mason, and M. A. Erdmann, "Experiments with Non-Holonomic Manipulation," in *ICRA*, 2002.

[19] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[20] J. James J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *ICRA*, 2000.

[21] J. Barry, "Manipulation with Diverse Actions," Ph.D. dissertation, Massachusetts Institute of Technology, 2013.

[22] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *RAM*, vol. 19, no. 4, 2012.

APPENDIX

For reference the full pseudo-code for DARRT and DARRTCONNECT is reproduced here. This top-level code is very similar to the RRT [19] and the RRT-Connect [20] algorithms respectively. The implementation uses the Open Motion Planning Library [22].

DARRT(M, B, A, c_I, G)

$T \leftarrow \{c_I\}$
while $T \cap G = \emptyset$
  $s \leftarrow$ SAMPLE(M)
  $t \leftarrow \arg\min_{v \in T}$Distance$(v, s, M, A,$ **True**$)$
  $T \leftarrow T \cup$ EXTEND$(t, s, M, B, A,$ **True**$)$
return ExtractPath(T)

DARRTCONNECT(M, B, A, c_I, G)

$T_a \leftarrow \{c_I\}$, $T_b \leftarrow \{$randomState$(G)\}$
$F \leftarrow$ **True** // *True when extending forwards*
while True
  if F: $T_b \leftarrow T_b \cup \{$randomState$(G)\}$
  $s \leftarrow$ SAMPLE(M)
  $t \leftarrow \arg\min_{v \in T_a}$Distance$(v, s, M, A, F)$
  $\{c_1, ..., c_l\} \leftarrow$ EXTEND(t, s, M, B, A, F)
  $T_a \leftarrow T_a \cup \{c_1, ..., c_l\}$
  if $l > 0$ // *Extend T_b towards T_a*
    $t \leftarrow \arg\min_{v \in T_b}$Distance$(v, c_l, M, A, \neg F)$
    $\{b_1, ..., b_q\} \leftarrow$ EXTEND$(t, c_l, M, B, A, \neg F)$
    if $b_q = c_l$: **return** ExtractPath(T_a, T_b)
  swap(T_a, T_b), $F \leftarrow \neg F$