Learning to Plan for Constrained Manipulation from Demonstrations

Mike Phillips
mlphilli@andrew.cmu.edu
Carnegie Mellon University

Victor Hwang
vchwang@andrew.cmu.edu
Carnegie Mellon University

Sachin Chitta
sachinc@willowgarage.com
Willow Garage

Maxim Likhachev
maxim@cs.cmu.edu
Carnegie Mellon University

***Abstract*—Motion planning in high dimensional state spaces, such as for mobile manipulation, is a challenging problem. Constrained manipulation, e.g. opening articulated objects like doors or drawers, is also hard since sampling states on the constrained manifold is expensive. Further, planning for such tasks requires a combination of planning in free space for reaching a desired grasp or contact location followed by planning for the constrained manipulation motion, often necessitating a slow two step process in traditional approaches. In this work, we show that combined planning for such tasks can be dramatically accelerated by providing user demonstrations of the constrained manipulation motions. In particular, we show how such demonstrations can be incorporated into a recently developed framework of planning with experience graphs which encode and reuse previous experiences. We focus on tasks involving articulation constraints, e.g. door opening or drawer opening, where the motion of the object itself involves only a single degree of freedom. We provide experimental results with the PR2 robot opening a variety of such articulated objects using our approach, using full-body manipulation (after receiving kinesthetic demonstrations). We also provide simulated results highlighting the benefits of our approach for constrained manipulation tasks.**

I. Introduction

In order to perform useful tasks robots must not only be able to move safely through their environments but must also be able to manipulate objects in them. Motion planners can be used to solve manipulation problems though planning times suffer for more complex tasks. An example of such tasks is constrained manipulation, e.g. opening doors or drawers. The motion required for such tasks occurs on a constrained manifold, e.g. the motion of the gripper is constrained to stay on the handle of the door with a firm grip, which cannot be directly sampled in joint space.

Tasks such as opening doors or drawers are often addressed using two stages of planning: a first stage where a motion planner is used to plan the initial path to a contact or grasp location followed by a second stage where a constrained plan is computed. This two stage approach can be slow since the goal state of one stage needs to be fed as the start state for the next. In particular, traditional approaches that plan from scratch every time are unable to exploit previous experiences which is a huge disadvantage for tasks like door or drawing opening which are essentially repetitive.

In this work we augment an existing Experience Graph planner [12] with user generated demonstations in order to obtain fast planning times in such challenging constrained problems. Experience Graphs (E-Graphs) are formed from a collection of paths. These could be previous paths that the planner generated or, as we show in this work, come from user demonstrations. Planning on E-Graphs is done with an A* based algorithm and therefore, the state space is represented as a discretized graph. When planning with E-Graphs the search is focused toward reusing parts of paths that look like they help find a solution quickly. The planner guarantees completeness and a bound on the sub-optimality of the solution cost with respect to the graph representing the problem.

We show that by using demonstrations with Experience Graphs, motion planning can be significantly sped up. This approach is flexible as we are still running a complete planner which is focused on reuse when useful, but is not forced or hard-coded to make a previous path work where it is not helpful.

II. Related Work

There has been a large amount of work within the field of "learning from demonstration" which incorporates teacher examples to generate policies for the robot [9, 8, 11, 1, 15]. Our work also uses demonstrations but differs from these approaches. In learning from demonstration literature, the provided examples show the desired behavior and therefore are goal (or reward) directed. This means that the demonstrations are provided with the goal or reward of the task already in mind. In our problem, demonstrations are given before knowing the goal or task. Some or all of the demonstrated movements may be irrelevant for a given query and the planner determines this during the search. The demonstrations are purely used to help search the state space more efficiently.

In our approach the demonstrations are used to guide the planner to a solution more quickly and avoid unnecessary exploration. There has been quite a bit of research on incorporating prior information from prior searches into the planning process. Bruce et. al. [4] extended RRTs to reuse cached plans and bias the search towards waypoints from old paths. A feature-based approach involves selecting a trajectory from a database from a similar scenario based on the positions of the start, goal, and obstacles relative to the robot [6]. The selected path is then tuned to fit the current scenario using a local optimizer. In [7] a bi-directional RRT is used to draw the search toward a path from a database which is most similar to the new motion planning problem (based on distances to

the start, goal and obstacles). Some more recent work [3] also attempts to repair paths from a database of past paths using sampling-based planners. Planning on Experience Graphs is an A* based method for reusing paths in new queries by guiding the search toward parts of old paths if they appear as though they will help the planner find the goal faster [12]. This method provides guarantees on completeness and solution quality which the other methods we refered to lack. E-Graphs are able to do this regardless of the quality of the paths put into the Experience Graph. We use this method in our work.

This work is focused on planning to manipulate objects in the environment. In particular, we deal with objects in the environment that inherently have constraints enforced on them. For instance, a cabinet door is constrained to swing about its hinge. Planning with constraints has been addressed in the recent past. Past approaches include local methods that provide fast smooth trajectory generation while maintaining workspace constraints [17]. However, this lacks global exploration of the state space and therefore is not guaranteed to find a valid solution even if one exists. Sampling-based planners on constraint manifolds allow for probabilistic completeness, [2, 10]. Other approaches include offline computation of constraint manifolds [16] and constructing an atlas for the constraint manifold at runtime [13]. Reusing demonstrations can help in improving the performance of planning for constraint tasks, something that no existing approach exploits. We aim to show that our approach can significantly improve its performance by reusing demonstrations while at the same time dealing robustly with changes in the environment, and gracefully planning from scratch when necessary.

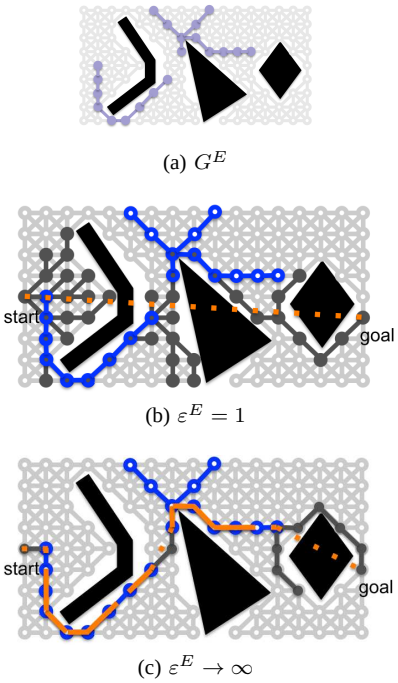
III. Experience Graphs

This section provides a brief description of how E-Graphs work. For more details see the prior work where Experience Graphs were introduced [12].

An Experience Graph G^E is a collection of previously planned paths (experiences). Planning with Experience Graphs uses weighted A* to search the original graph G (which represents the planning problem) but tries to reuse paths in G^E in order to minimize the exploration of the original graph G. This is done by modifying the heuristic computation of weighted A* to drive the search toward paths in G^E, that appear to lead towards the goal. Essentially, this comes down to computing a new heuristic distance from the state in question to the goal where traveling off of G^E is penalized but traveling on edges of G^E is not. The new heuristic h^E is defined in terms of the original heuristic h^G and edges in G^E for all states s_0 in the original graph.

$$h^E(s_0) = \min_{\pi} \sum_{i=0}^{N-1} min\{\varepsilon^E h^G(s_i, s_{i+1}), c^E(s_i, s_{i+1})\} \quad (1)$$

where π is a path $\langle s_0 \dots s_{N-1} \rangle$ and $s_{N-1} = s_{goal}$ and ε^E is a scalar ≥ 1. As shown in [12], the heuristic is ε^E-consistent and therefore guarantees that the solution cost will be no worse than ε^E times the cost of the optimal solution when running A* search to find a path. More generally, planning with Experience Graphs using weighted A* search inflates the entire h^E heuristic by ε. Consequently, the cost of the solution is bounded by $\varepsilon \cdot \varepsilon^E$ times the optimal solution cost.



(a) G^E

(b) $\varepsilon^E = 1$

(c) $\varepsilon^E \to \infty$

Fig. 1. Effect of ε^E. The light gray circles and lines show the original graph. The darkened states and edges in (a) show the E-Graph. In (b) and (c) the dark gray circles show states explored by the planner in order to find a solution. The light dashed line shows the heuristic path from the start state. Notice that when ε^E is large, this path travels along the E-Graph and avoids most obstacles (there are few explored states). On the other hand when ε^E is small, the heuristic (in this case euclidean distance) drives the search into several obstacles and causes many more expansions. It should be noted that $\varepsilon > 1$ is used in these examples.

Equation 1 is computed by finding the shortest path from s_0 to the goal in a simplified version of the planning problem where there are two kinds of edges. The first set are edges that represent the same connectivity as h^G in the original planning problem but their cost is inflated by ε^E. The second set of edges are from G^E with their costs c^E (∞ if the edge is not in G^E). As ε^E increases, the heuristic computation goes farther out of its way to use helpful E-Graph edges.

Figure 1 shows the effect of varying the parameter ε^E. As it gets large, the heuristic is more focused toward E-Graph edges. It draws the search directly to the E-Graph, connects prior path segments, and only searches off of the E-Graph when there aren't any useful experiences (such as around the last obstacle). There are very few expansions and much of the exploration of the space is avoided. As ε^E approaches 1 (optimality) it ignores G^E and expands far more states.

IV. Demonstration-based Experiences

The main contribution of our paper is in showing how demonstrations can be integrated into planning with Experience Graphs. The use of demonstrations in conjunction with Experience Graphs is not as simple as just adding the

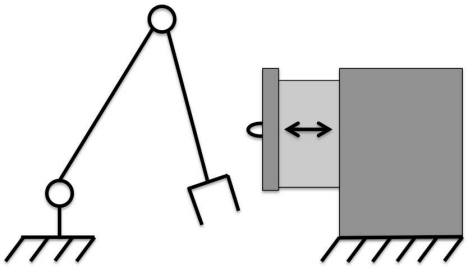


Fig. 2. A two link planar arm and a drawer that can be manipulated.

demonstrations into the graph as additional experiences for several reasons. First, demonstrations may not lie on the original graph. Second, since the demonstrations show how to manipulate an object (e.g. how to open a door), they require adding a new dimension to the state-space, the dimension along which the object is being manipulated. Consequently, the underlying graph as well as Experience Graph must be updated, to include this dimension. Finally, the heuristic used in the graph search will need to be improved to guide the search towards the object that needs to be manipulated as well guide it in how to manipulate the object. We describe how to address these challenges in this section. We will use a running example of a 2 link planar manipulator opening a drawer to make the explanation clearer (Figure 2).

A. Notations and Overall Framework

First we'll go through some definitions and notations and briefly describe the overall framework.

The original graph representing the planning problem is $G = \langle V, E \rangle$. Each vertex $v \in V$ represents a robot state: $coord(v) \in \mathbb{R}^n$. We also assume a database of demonstrations $\mathcal{D} = \langle \mathcal{T}_1 \dots \mathcal{T}_m \rangle$. Each $\mathcal{T}_i$ is a set of discretized trajectories corresponding to the i^{th} object in the environment that can be manipulated. $\mathcal{T}_b = \{\langle a^b_{11} \dots a^b_{1k_1} \rangle \dots \langle a^b_{\ell 1} \dots a^b_{\ell k_\ell} \rangle\}$ where $a^b_{ij} \in \mathcal{T}_b$ is the j^{th} point in the i^{th} trajectory for object b. $a^b_{ij} \in \mathbb{R}^{n+1}$. The extra dimension corresponds to the state of the manipulated object, which we will term z. In Figure 2 this would be how far the drawer is pulled open. We will use $zcoord(a^b_{ij})$ to represent the value of the state of the object b at a^b_{ij}. For every object b, we also use $\mathcal{Z}_b$ to represent the set of all values of z that are present in $\mathcal{T}_b$. Formally, $\mathcal{Z}_b = \{z | \exists a_{ij} \in \mathcal{T}_b \, s.t. \, z = zcoord(a_{ij})\}$.

Finally, we assume that the objects we are manipulating lie on one-dimensional manifolds in a higher dimensional space. For instance, when opening a cabinet, the door is constrained to move on a one dimensional manifold. The planner infers how to operate the manipulated objects automatically from one or more demonstrations. There is no prior model of any of the objects the robot interacts with. Instead we assume there is a stationary point of contact on the object that the robot's end-effector comes into contact with during manipulation. During demonstration, we observe the movement of this contact point along a curve, which z parameterizes. The domain specific function $y = \varphi(coord(v))$ computes the coordinates of the contact point on the robot. This function is many-to-one. In Figure 2 this corresponds to the pose of the end-effector and would be computed from $coord(v)$ using forward kinematics. Note that in our simple example there are two states x that could produce the same y (corresponding to an elbow up or down configuration). The drawer handle's constraint manifold is the small line segment which would be traced by the handle while opening the drawer.

---

planToManipulate($G, \mathcal{D}, s_{start}, z_{goal}, obj$)

1: $\mathcal{T} = \mathcal{T}_{obj} \in \mathcal{D}$
2: $G_{manip} = buildGraph(G, \mathcal{T})$
3: $G^E = createEGraph(\mathcal{T})$
4: $\pi = findPath(G_{manip}, G^E, \mathcal{T}, s_{start}, z_{goal})$
5: $return \, \pi$

The $planToManipulate$ algorithm shows the high-level framework. First it selects the demonstrations from $\mathcal{D}$ that correspond to object obj. Then it constructs a new graph G_{manip} to represent the planning problem. This graph represents the robot's own motion (as before), contact with the object, and manipulation of the object by the robot. The $createEGraph$ function uses the demonstration to create the Experience Graph G^E as well as to augment the graph with a new dimension. Finally, a planner is run on the two graphs as described in [12]. The following sections describe the construction of the graph $G_{manip} = \langle V_{manip}, E_{manip} \rangle$ and a new heuristic to guide search for motions that manipulate the objects.

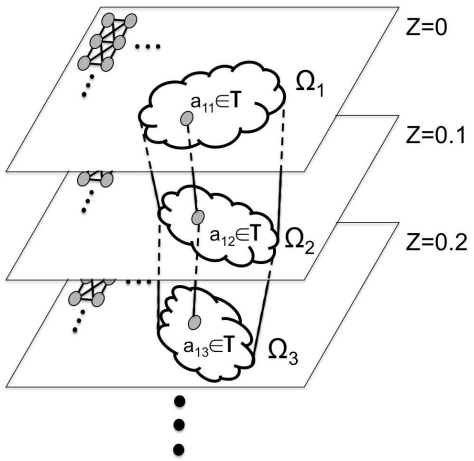


Fig. 3. The graph construction. The layered planes show how the original graph is duplicated for each value of $z \in \mathcal{Z}$. The a_{ij} elements are points on a demonstrated trajectory. During the demonstration the robot's state changes (movement within the plane) as well as the object's state (movement between planes). Each a_{ij} element is in a set of states Ω_j. In addition to this state Ω_j contains $s \, s.t. \, withinError(\varphi(coord(s)), \varphi(coord(a_{ij}))) \wedge zcoord(s) = zcoord(a_{ij})$.

B. Task-based Redefinition of States

The provided demonstrations change the state space in two significant ways. First, the manipulated object adds a new

dimension to the graph. Secondly, the demonstration may contain robot states that do not fall on any state in the original graph. In order to handle this, we construct a new vertex set V_{manip} as shown below.

$$\begin{aligned} V_{manip} =& V_{orig} \cup V_{demo}, \; where \\ V_{orig} =& \{v | \langle coord(v), zcoord(v) \rangle = \langle coord(u), z \rangle \\ & \forall u \in V, z \in \mathcal{Z}\} \\ V_{demo} =& \{v | \langle coord(v), zcoord(v) \rangle = a_{ij} \in \mathcal{T}\} \end{aligned}$$

The new vertex set is a combination of the old vertices and the vertices from the demonstrations. The vertex set V_{orig} contains the vertices in the original graph but repeated for each possible value of the new variable z. The set V_{demo} is the set of vertices that exist in the demonstration trajectories. In Figure 3 the planes show the layers of the original graph repeated for each value of z. Additionally, we can see the states that come from the demonstration.

C. Task-based Redefinition of Transitions

The demonstrations not only change the state space, but also affect the connectivity of the graph due to the additional dimension as well as motions in the demonstration that are not used in the original graph.

The new edge set E_{manip} is defined below.

$$\begin{aligned} E_{manip} =& E_{orig} \cup E_{demo} \cup E_{bridge} \cup E_z \; where \\ E_{orig} =& \{(u,v) | \exists \tilde{u}, \tilde{v} \in V s.t. \, coord(\tilde{u}) = coord(u) \wedge \\ & coord(\tilde{v}) = coord(v) \wedge \\ & (\tilde{u}, \tilde{v}) \in E \wedge \\ & zcoord(u) = zcoord(v)\} \\ E_{demo} =& \{(u,v) | \langle coord(u), zcoord(u) \rangle = a_{i,j} \in \mathcal{T} \wedge \\ & \langle coord(v), zcoord(v) \rangle = a_{i,j+1} \in \mathcal{T} \\ E_{bridge} =& \{(u,v) | \langle coord(u), zcoord(u) \rangle \in \mathcal{T} \wedge \\ & \exists \tilde{v} \in V s.t. \, coord(\tilde{v}) = coord(v) \wedge \\ & connectable(u,v) \wedge \\ & zcoord(u) = zcoord(v)\} \\ E_z =& \{(u,v) | \exists (\tilde{u}, \tilde{v}) \in E_{demo} s.t. \\ & withinError(\varphi(coord(u)), \varphi(coord(\tilde{u}))) \wedge \\ & zcoord(u) = zcoord(\tilde{u}) \wedge \\ & withinError(\varphi(coord(v)), \varphi(doord(\tilde{v}))) \wedge \\ & zcoord(v) = zcoord(\tilde{v})\} \end{aligned}$$

The new edge set is a combination of edges from the original graph E_{orig} (replicated for each value of z), edges that come from demonstrations E_{demo}, "bridge edges" E_{bridge}, and Z edges E_z.

Bridge edges connect demonstration states to states in the discretized original graph. The "connectable" function used to define them should typically be used if the two states u and v are very close (such as when the demonstration state u falls within the discretized "bin" of the original graph state v). The two states must also share the same z-value (the manipulated object must be in the same state). For example, in Figure 2 a bridge edge may be added whenever the euclidean distance between the two joint angles of demonstration state and an original graph state are within a small distance of each other and the drawer is pulled open the same amount.

Z edges generalize the demonstrations in order to create edges on the object's constraint manifold that may not have existed in the demonstrations. This means that if the contact point of the robot at state u is very close to that of state $\tilde{u}$ ($\varphi(coord(u)) \approx \varphi(coord(\tilde{u}))$), the object is in the same state ($zcoord(u) = zcoord(\tilde{u})$), these conditions are also true for v and $\tilde{v}$, and $\tilde{u}$ is connected to $\tilde{v}$ in the demonstrations, then we will connect u to v (provided the action is collision free, as with any edge in the graph). These edges allow the planner to find ways to manipulate the object other than exactly how it was done in demonstrations. This is especially important if part or all of the specific demonstration is invalid (due to collision) but it may still be possible to manipulate the object. Figure 3 shows this using the cloud-shaped Ω. Any of the states that fall in Ω_i can connect to states in Ω_{i+1} or Ω_{i-1}.

D. Task-based Heuristic

Since the goal is to manipulate an object to a particular state (for instance, open a drawer), the search will be slow unless the heuristic guides the planner to modify the object toward the goal configuration. With that in mind we outline a heuristic that takes into account the motion of the robot required to reach contact with the object as well the manipulation of that object.

We introduce a two part heuristic h^G_{env} built on top of the original heuristic for the environment h^G. The E-Graph heuristic h^E described in section III will now use h^G_{env} instead of h^G. For any state $s = \langle x, z \rangle$ we are trying to provide an admissible (underestimating) guess for the remaining cost to get to the goal (have $z = z_{goal}$). The general idea is that $h^G_{env}(s)$ estimates the cost of getting the robot in contact with the object plus the cost it takes to manipulate the object so that the variable z moves through all the required values to become z_{goal}. More formally,

$$h^G_{env}(s) = \min_{v_z \ldots v_{z_{goal}}} h^G(s, v_z) + \sum_{k=z}^{z_{goal}-1} h^G(v_k, v_{k+1})$$

$$\begin{aligned} v_k \in & \{v \in V_{manip} | \exists a_{ij} \in \mathcal{T}, s.t. \\ & withinError(\varphi(coord(a_{ij})), \varphi(coord(v))) \\ & \wedge \, zcoord(a_{ij}) = k\} \end{aligned}$$

We can see that we are having the contact point pass through all the poses shown in the demonstration (between the z of state s and the goal z). There may be many robot configurations to choose from for each of these contact poses in order to get a minimum sequence. In our experiments, we chose a

heuristic $h^G(a, b)$ that computes the linear distance that the contact point travels between the two robot configurations [5]. An advantage of this heuristic is that we don't need to consider the set of all robot configurations. Since all the robot configurations in a set (e.g. all possible states to choose for some v_k) have the same contact point, they are equivalent inputs to this heuristic function (so any state with that contact point will do). Therefore, the sequence of $v_z \dots v_{z_{goal}}$ can just be that segment of a demonstration. This makes h^G_{env} easy to compute.

E. Theoretical Properties

As we showed earlier, it's possible for edges (motions) in the demonstration to not exist in the original graph. These extra edges can help the planner find cheaper solutions than what it would have been able to achieve without them. It also may be able to solve queries for which there was no solution in the original graph G alone.

An important thing to note is that while the quality of the demonstration can dramatically affect the planning times and the solution cost, the planner always has a theoretical upper bound on the solution cost with respect to the optimal cost in graph G_{manip}.

Theorem 1: For a finite graph G and finite Experience Graph G^E, our planner terminates and finds a path in G_{manip} that connects s_{start} to a state s with $zcoord(s) = z_{goal}$ if one exists.

Theorem 2: For a finite graph G and finite Experience Graph G^E, our planner terminates and the solution it returns is guaranteed to be no worse than $\varepsilon \cdot \varepsilon^E$ times the optimal solution cost in G_{manip}.

The proofs follow by applying the theorems in [12] to the graph G_{manip}. Also, since we are running a full planner in the original state space, for a low enough solution bound, the planner can find ways to manipulate the environment objects more efficiently (cheaper) than the user demonstrations.

V. Experimental Results

We tested our approach by performing a series of mobile manipulation tasks with the PR2 robot including opening drawers and doors. All the tasks involve manipulation with a single arm, coupled with motion of the base. The end-effector of the right arm of the PR2 is restricted to be level i.e. its roll and pitch are restricted to a fixed value (zero). This results in the motion of the arm happening in a 5 dimensional space parameterized by the position of the right end-effector (x,y,z), the yaw of the end-effector and an additional degree of freedom corresponding to the shoulder roll of the right arm. We consider the overall motion of the robot to be happening in a nine dimensional state space: the 5 degrees of freedom mentioned above for the arm, the three degrees of freedom for the base and the an additional degree of freedom for the vertical motion of the torso (spine).

When performing a task, an additional degree of freedom is added to the state space corresponding to the articulated object, bringing the dimensionality of the state space to a total of ten. An illustrative goal for the planner would be to change an articulated object to a specified value e.g., moving a cabinet door from the closed to open position. This requires the creation of plans for the combined task of grasping the handle of the cabinet door by coordinated motion of the base, spine and right arm of the robot, followed by moving the gripper appropriately (again using coordinated motions of the base, spine and right arm) along the circular arc required to open the cabinet door.

Kinesthetic demonstration, where the users manually moves the robot along a desired path to execute the task, was used to record the desired paths for different tasks. The values of the state space variables were recorded along the desired paths. Once the demonstrations have been recorded, the robot replays the demonstrated trajectories to execute the given task. As it executes the demonstrations, it uses its 3D sensors to record information about the changing environment. This 3D sensors trace (represented as a temporal series of occupancy grids in 3D) represent the motion of the target object (e.g. the cabinet door) throughout the demonstration.

The stored temporal sensor information provides information about the evolution of the changing environment, particularly for use in collision checking. Forward kinematics is used to determine the demonstrated workspace trajectory for the contact point of the gripper and the articulated object. This information, along with the recorded state data, represents data that can be fed back into the E-Graph for later use.

A. Robot Results

Our planner was implemented in four different scenarios with the PR2 robot: opening a cabinet, opening a drawer with an external handle attached, opening an overhead kitchen cabinet, and opening a freezer. The overall goal for each task is for the robot to start from an arbitrary position, move to the desired task location, grasp the handle and open the cabinet or drawer.

For each scenario, a full 3D map of the environment was first built using the stereo sensors on the robot. The opening part of the task was then demonstrated with the robot by a user. The robot then replayed the demonstrated motion on its own, recording the additional visual sensor data needed in the process to complete the demonstration. This data is available to the planner for incorporation into the E-Graph.

The planner was then tested using different start states. This required the planner to generate motions that would move the robot to a location where it could grasp the handle on the drawer/cabinet/freezer. Note that this part of the motion had not been demonstrated to the planner. The planner also had to generate the motion required to open the drawer/cabinet/freezer. Again, note that the robot could be in a different start state at the beginning of its motion for opening the drawer/cabinet/freezer as compared to the start state for the demonstrated motion. Further, there may be additional obstacles in the environment that the planner needs to deal with. Figure 4 shows still images of these trials.

Fig. 4. PR2 opening an Ikea cabinet, metal drawer, overhead kitchen cabinet, and freezer door, respectively.

Table I shows the planning times for these demonstrations. While the weighted A* planner solution time is shown, only the E-Graph planner result was executed on the robot. In one case, the weighted A* was unable to produce a plan in the allotted 60 seconds. Weighted A* was run with $\varepsilon = 20$, while our planner ran with $\varepsilon = 2$ and $\varepsilon^E = 10$ for an equivalent bound of 20.

TABLE I
PLANNING TIMES IN SECONDS FOR OPENING A FILE DRAWER, IKEA CABINET, OVERHEAD KITCHEN CABINET, AND FREEZER.

	E-Graph	Weighted A*
Drawer	2.06	2.96
Cabinet	1.83	12.87
Kitchen Cabinet	2.87	(unable to plan)
Freezer	1.52	7.81

B. Simulation Results

A separate set of simulated tests was conducted to measure the performance of the planner and compare it to weighted A* (without re-expansions) and a sampling-based approach. Weighted A* was run with $\varepsilon = 20$, while our planner ran with $\varepsilon = 2$ and $\varepsilon^E = 10$ for an equivalent bound of 20. The environments were generated by rigidly transforming two target objects (cabinet and drawer) to various locations in a room (the robot start pose was constant). Figure 5 shows a snapshot of the simulation environment.

Table II shows planning statistics of weighted A* versus planning with E-Graphs. These results show that using the Experience Graph allows us to find solutions with fewer expansions and therefore, in less time.

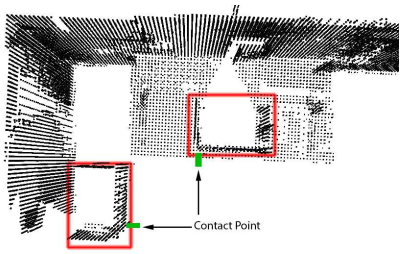


Fig. 5. The simulation environment. The red boxes represent example locations of the target object to be manipulated. The green boxes represent the contact point that the robot gripper should attempt to grasp.

We also compared against Constrained Bi-directional Rapidly-Exploring Random Tree (CBiRRT), wich is designed to help the RRT algorithm deal with planning on constraints which may be small compared to the state space and therefore difficult to sample [2]. Like most RRT algorithms this method

TABLE II
A COMPARISON BETWEEN E-GRAPHS AND WEIGHED A* OVER 35 SIMULATIONS

	E-Graph		Weighted A*	
	Mean	Std dev	Mean	Std dev
Drawer	2.75	1.73	7.25	16.62
Cabinet	1.74	0.70	54.69	43.49

repeatedly chooses a random sample, and tries to extend the nearest neighbor in the search tree toward it (since this is bi-directional, it grows both). The primary difference is that the extension is done by taking small unconstrained steps (like a linear step in c-space) followed by a projection step back on to the constaints. We use a state variable to represent how far the object has moved (like in our approach). If the object is "closed" then the projection does nothing (the robot doesn't have to be holding the contact point). If the object is in any other state then it has been moved, and we project configurations so the robot is holding the contact point. The constraint manifold learned in our approach is used for the projection step. For the goal state (the root of the backward tree) we provided the final configuration from the demonstration.

TABLE III
A COMPARISON BETWEEN E-GRAPHS AND CBIRRT

	E-Graph time (s)		CBiRRT time (s)	
	Mean	Std dev	Mean	Std dev
Drawer	2.76	1.88	44.31	28.39
Cabinet	1.94	0.76	1.72	1.60
	E-Graph base motion (m)		CBiRRT base motion (m)	
	Mean	Std dev	Mean	Std dev
Drawer	0.61	0.30	1.51	0.45
Cabinet	0.88	0.27	1.53	0.34
	E-Graph arm motion (rad)		CBiRRT arm motion (rad)	
	Mean	Std dev	Mean	Std dev
Drawer	5.37	2.74	5.50	2.04
Cabinet	6.83	2.00	4.42	0.86
	E-Graph consistency		CBiRRT consistency	
Drawer	0.33		10.70	
Cabinet	0.37		3.93	

Table III compares E-Graphs to CBiRRT. In the first section of the table we can see that the planning times for the two approaches are similar for simpler scenarios though E-Graphs perform better on others (across 35 trials). We expect E-Graphs to continue to perform better than sampling planners as tasks become more complicated since there is more room for reuse of prior experience. We also found the E-Graph solutions to be of similar or better quality (refer to the base and arm distance metrics in the middle of the table). At the bottom of the table, we see results from a consistency experiment. Consistency measures how similar output of a planner is, given similar inputs (start and goal). In many domains, this kind of path predictability is critical for people to be comfortable around robots. We tested this by choosing 5 similar start poses for the robot and 5 similar locations of the cabinet/drawer. We then plan between all pairs to get 25 paths. We used the dynamic time warping similarity metric [14] to compare the methods. Having a value closer to 0 means the paths are more similar. Since this method is for comparing pairs of paths, we performed an all-pairs comparison and then took the average path similarity. We can see that E-Graphs produce more consistent paths due to the deterministic nature of the planner.

The final simulation scenario demonstrates the capability of using a partial E-Graph. An artificial obstacle was intentionally placed to obstruct a portion of the provided experience. We show that the E-Graph planner derives as much of the solution as it can from the provided experience before doing a normal weighted A* search to replace the portion of the experience that is in collision.

Figure 6 shows the specific case. On the left we see the final configuration from the demonstration which is in significant collision with an obstacle. It is clear that simply playing back the demonstration to open this cabinet would fail (even small modifications on the joints would not be sufficient). In the image on the right we can see that the planner generates a valid final pose (and a valid path leading up to it) by lowering the elbow. The E-Graph planner actually uses the first half of the demonstration (which is valid) and then generates new motions for the rest. We can see from the final pose, that the motion is dramatically different from the demonstration in order to accomodate the new obstacle.

Table IV shows the time performance of this trial. The weighted A* performs worse because it must build the solution from scratch. The partial E-Graph solution completes in an order of magnitude less time. Figure 7 shows that when using none of the E-Graph (planning from scratch) a similar solution is found but it takes much longer. For comparison, the planning statistics for the provided demonstration without the obstacle is shown as well. We see that the partial E-Graph only took slightly more time than the case where the obstacle is removed (and the complete demonstration could be used).

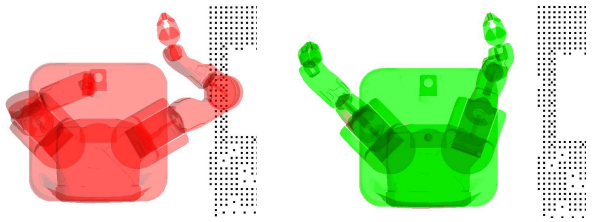
The end result of this simulation shows that the E-Graph planner can take full advantage of provided experiences, even when parts of the provided experience are invalid.

TABLE IV
PERFORMANCE STATISTICS FOR PARTIAL E-GRAPH PLANNING.

	Planning time	Expansions
Weighted A*	51.90	16402
Partial E-Graph	2.22	59
Complete E-Graph (without obstacle)	2.08	47

VI. CONCLUSION

In this work we presented a way to use Experience Graphs to improve the performance of planning for constrained manipulation by providing user demonstrations. The planner is able to find paths with bounded sub-optimality even though the demonstrations can be of arbitrary quality (and don't even



(a) The second half of the demonstration is in collision with an obstacle. The last pose is shown here.

(b) The planner reuses as much of the demonstration as it can and then generates the rest from scratch. The final pose in the path is shown. The elbow has been dropped to accomodate the obstace.

Fig. 6.

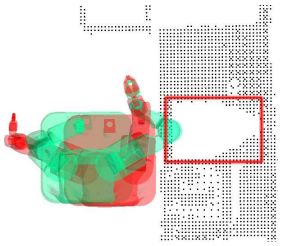


Fig. 7. A similar solution is found when planning from scratch.

need to be useful). Experimentally we provide results on high dimensional mobile manipulation tasks using the PR2 robot to open cabinets, freezers, and drawers both in simulation and on the real robot.

In future work we would like to look into the use of demonstrations for unconstrained manipulation and manipulation of objects that lie on multi-dimensional manifolds.

Acknowledgments

We thank Willow Garage for their support of this work. This research was also sponsored by ARL, under the Robotics CTA program grant W911NF-10-2-0016.

References

[1] Brenna Argall, Sonia Chernova, Manuela M. Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57 (5):469–483, 2009.

[2] Dmitry Berenson, Siddhartha Srinivasa, David Ferguson, and James Kuffner. Manipulation planning on constraint manifolds. In *IEEE International Conference on Robotics and Automation (ICRA '09)*, May 2009.

[3] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A robot path planning framework that learns from experience. In *ICRA*, 2012.

[4] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[5] B.J. Cohen, S. Chitta, and M. Likhachev. Search-based planning for manipulation with motion primitives. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010.

[6] Nikolay Jetchev and Marc Toussaint. Trajectory prediction: Learning to map situations to robot trajectories. In *IEEE International Conference on Robotics and Automation*, 2010.

[7] Xiaoxi Jiang and Marcelo Kallmann. Learning humanoid reaching tasks in dynamic environments. In *IEEE International Conference on Intelligent Robots and Systems*, 2007.

[8] J. Kober and J. Peters. policy search for motor primitives in robotics. In *advances in neural information processing systems 22 (nips 2008), cambridge, ma: mit press*, 2009.

[9] Petar Kormushev, S. Calinon, and D. G. Caldwell. Robot motor skill coordination with EM-based reinforcement learning. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 3232–3237, Taipei, Taiwan, October 2010.

[10] Giuseppe Oriolo and Christian Mongillo. Motion planning for mobile manipulators along given end-effector paths. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005, April 18-22, 2005, Barcelona, Spain*, pages 2154–2160. IEEE, 2005.

[11] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. learning and generalization of motor skills by learning from demonstration. In *international conference on robotics and automation (icra2009)*, 2009.

[12] Michael Phillips, Benjamin J. Cohen, Sachin Chitta, and Maxim Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, 2012.

[13] Josep Maria Porta Pleite, Léonard Jalliet, and Oriol Bohigas Nadal. Randomized path planning on manifolds based on higher-dimensional continuation. 31(2):201–215, 2012.

[14] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26, 1978.

[15] M. Stolle and C. Atkeson. Policies based on trajectory libraries. In *IEEE International Conference on Robotics and Automation*, 2006.

[16] Ioan A. Sucan and Sachin Chitta. Motion planning with constraints using configuration space approximations. Vilamoura, Algarve, Portugal, 2012. IEEE.

[17] Yuandong Yang and Oliver Brock. Elastic roadmaps - motion generation for autonomous mobile manipulation. *Auton. Robots*, 28(1):113–130, 2010.