Noname manuscript No.
(will be inserted by the editor)

Learning to Plan for Constrained Manipulation from Demonstrations

Mike Phillips · Victor Hwang · Sachin Chitta · Maxim Likhachev

Received: date / Accepted: date

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.

M. Phillips
Carnegie Mellon University
E-mail: mlphilli@andrew.cmu.edu

V. Hwang
Carnegie Mellon University
E-mail: vchwang@andrew.cmu.edu

S. Chitta
Willow Garage
E-mail: sachinc@willowgarage.com

M. Likhachev
Carnegie Mellon University
E-mail: maxim@cs.cmu.edu

Keywords Motion Planning · Manipulation Planning · Experience Graphs

1 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.

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 [19] 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.

2 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 [15] [13] [18] [2] [27]. 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. Perhaps the most straight forward reuse approach is the PRM [12], which simply by begin a multi-query approach, reuses computation across trials. Bruce et. al. [5] 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 [9]. The selected path is then tuned to fit the current scenario using a local optimizer. In [10] 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). MP-RRT extends the RRT to replanning scenarios by maintaining a forest from the previously computed RRTs [31]. Another family of planning methods that reuses previous search effort are D* [26] and D* Lite [14]. Like E-Graphs, these are graph search methods, unlike E-Graphs, these methods require either the start or the goal to stay constant for any reuse to occur. Lightning [3], is a recent work that also attempts to repair paths from a database of past paths using sampling-based planners. We provide an experimental comparison against this state of the art method. 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 [19]. 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. The method has been extended to anytime and incremental planning scenarios as well [20].

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 [30]. 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, [4] [17]. Other approaches include offline computation of constraint manifolds [28] and constructing an atlas for the constraint manifold at runtime [23]. 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.

3 Experience Graphs

This section provides a description of how E-Graphs work (first introduced in [19]).

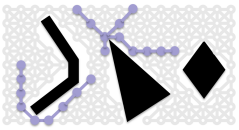
An Experience Graph G^E is a collection of previously planned paths (experiences). Planning with Experience Graphs uses weighted A* to search the original graph G (which represents the planning problem) but tries to reuse paths in G^E in order to minimize the exploration of the original graph G (as G^E is dramatically smaller than 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.

Weighted A* maintains an $OPEN$ list of states (initialized with only the start state) which have been discovered but not yet *expanded*. When a state s is expanded, each neighbor s' (a state connected to it directly by an edge) may have its cost from the start, $g(s')$ updated if by reaching it through s was cheaper and if so, s' is put in $OPEN$ with a priority of $f(s') = g(s') + \varepsilon h(s')$. Where $h(s')$ is a heuristic estimate for the remaining cost to the goal and $\varepsilon \geq 1$. Weighted A* works by iteratively expanding the state in $OPEN$ with the minimum priority until the goal state has the minimum priority, at which point the algorithm terminates. If the heuristic is *admissible* (never overestimates the cost to the goal) then the solutions are guaranteed to be no larger than ε times the cost of an optimal solution [22]. If the heuristic function is also consistent, $h(s) \leq c(s, s') + h(s') \forall s, s'$ and $h(s_{goal}) = 0$, the same guarantee is maintained when not allowing a state to be expanded more than once [16].

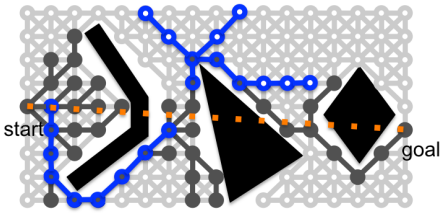
As stated eariler, the key to planning with Experience Graphs is creating a heuristic function h^E which is biased to follow edges in the E-Graph which lead to the goal. The E-Graph heuristic $h^E(s_0)$ for some state s_0 is computed by dynamic programming by finding the shortest path from the goal to s_0 through a graph which only contains E-Graph vertices, the goal, and s_0. There are two types of edges in this graph: pairs of states may be connected by an E-Graph edge, or connected by the original heuristic penalized by ε^E. Essentially, the path computation from the goal to the state in question penalizes traveling off of G^E but traveling on edges of G^E is not. Formally, for any state s_0 in the original graph:

$$h^E(s_0) = \min_{\pi} \sum_{i=0}^{\lambda-2} min\{\varepsilon^E h^G(s_i, s_{i+1}), c^E(s_i, s_{i+1})\} \tag{1}$$

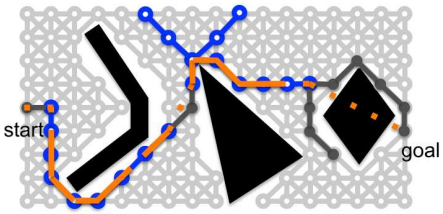
where π is a path $\langle s_0 \dots s_{\lambda-1} \rangle$ and $s_{\lambda-1} = s_{goal}$ and ε^E is a scalar ≥ 1. As shown later, the heuristic is ε^E-consistent and therefore guarantees that the solution cost will be no worse than ε^E times the cost of the optimal solution



(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.

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.

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. Note that the edges that use h^G allow h^E to guide the search to connect previous path segments and connect to the goal state, which might not lie on the E-Graph.

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

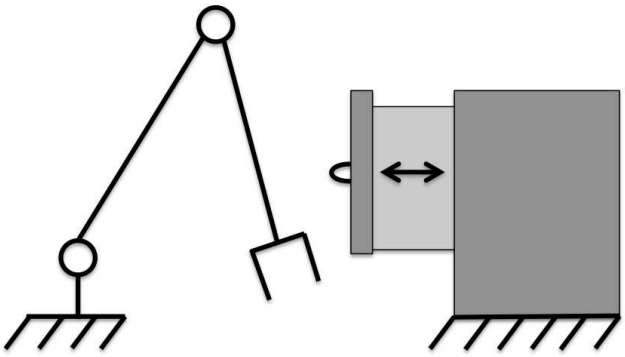


Fig. 2 A two link planar arm and a drawer that can be manipulated.

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.

4 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 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).

4.1 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. For simplicity, the algorithm will be described with only one possible contact point on the object, however, the algorithm works with an arbitrary number of demonstrations starting from any number of contact points. In our experimental analysis, we show how this works. During demonstration, we observe the movement of this contact point along a curve, which z parameterizes. As stated earlier, $coord(v)$ specifies the complete configuration of the robot. We then use a domain specific function $y = \varphi(coord(v))$ to compute the coordinate of the contact point on the robot, given the robot's configuration (i.e., forward kinematics). 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 [19]. 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.

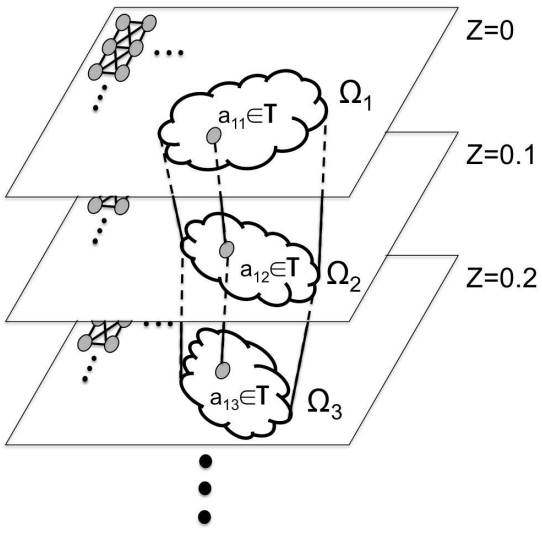


Fig. 3 The graph construction. The layered planes show how the original graph is duplicated for each value of $z \in \mathcal{Z}$. The a_{ij} elements are points on a demonstrated trajectory. During the demonstration the robot's state changes (movement within the plane) as well as the object's state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

4.2 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.

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

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.

4.3 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. We introduce two functions which are used in the definition of the new edge set E_{manip}. The $connectable(u, v)$ function returns true if two states can be connected in a simple collision free manner, such as by linear interpolation. The function is only applicable if the two states u and v are very close (such as when state u and v fall in the same "discretized bin"). The $withinError$ function determines if two contact poses are close enough to each other to be considered equivalent. The allowable error off the manifold is up to the user (not smaller than the discretization) but depends on the compliance of the object and robot.

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

The new edge set E_{manip} 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 two states must be *connectable* and 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 $\tilde{u}$, $withinError(\varphi(coord(u)), \varphi(coord(\tilde{u})))$, 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}.

4.4 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 of manipulating the object so that the variable z moves through all the required values to become z_{goal}. More formally,

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

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

This computation has 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 [7]. An advantage of this heuristic is that we do not 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. Specifically, the choices of v_k to search over becomes significantly smaller.

$$v_k \in \{ v \in V_{demo} | \wedge\, zcoord(v) = k \}$$

Therefore h^G_{env} can be computed using a Dijkstra search over the vertices from the demonstration and s, starting from the set of vertices from the demonstrations which have z-value of z_{goal}. As we are running a Dijkstra search backward, our edges will be directed from any demonstration state with z-value i to any with value $i - 1$. Finally, all demonstration states with value z connect to state s. We are therefore searching on a directed acyclic graph with the number of vertices on the order of the number of vertices in the demonstrations (which is typically very smaller), making this computation efficient. Additionally, the Dijkstra computes the minimum of the summation on the right first, which could be computed once at the start of planning and cached. When h^G_{env} is queried with some s, only the final edge to it must be chosen from the vertices with $zcoord(s)$.

4.5 Theoretical Properties

As we showed earlier, it is 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.*

We are searching the graph G_{manip} with Weighted A* (a complete planner), if a solution exists on this graph, our algorithm will find it.

Our planner provides a bound on the sub-optimality of the solution cost. The proof for this bound depends on our heuristic function h^E being ε^E-consistent.

Lemma 1 *If the original heuristic function $h^G(u, v)$ is consistent, then the heuristic function h^E is ε^E-consistent.*

From Equation 1, for any $s, s' \in V^G, (s, s') \in E^G$.

$$h^E(s) \leq min\{\varepsilon^E h^G(s, s'), c^E(s, s')\} + h^E(s')$$
$$h^E(s) \leq \varepsilon^E h^G(s, s') + h^E(s')$$
$$h^E(s) \leq \varepsilon^E c(s, s') + h^E(s')$$

The argument for the first line comes from Equation 1 by contradiction. Suppose the line is not true. Then, during the computation of $h^E(s)$, a shorter path π could have been found by traveling to s' and connecting to s with $min\{\varepsilon^E h^G(s, s'), c^E(s, s')\}$. The last step follows from h^G being admissible. Therefore, h^E is ε^E-consistent.

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}.*

Consider $h'(s) = h^E(s)/\varepsilon^E$. $h'(s)$ is clearly consistent. Then, $\varepsilon h^E(s) = \varepsilon \cdot \varepsilon^E h'(s)$. The proof that $\varepsilon \cdot \varepsilon^E h'(s)$ leads to Weighted A* (without re-expansions) returning paths bounded by $\varepsilon \cdot \varepsilon^E$ times the optimal solution cost follows from [16].

It is interesting to note that 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 according to the planner's cost function) than the user demonstrations.

5 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 the arm moving 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 in a 9 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. A possible goal for the planner would be to move the joint of an object to a specified z-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.

The planner can support arbitrary cost functions (as long as no motion results in negative cost), though the heuristic must be chosen to be consistent with respect to the cost function. The cost function we used for the planner is a weighted sum of end effector linear and angular motion, base linear and angular motion, motion of the spine, and motion of the arm's redundant joint.

Kinesthetic demonstration, where the user manually moves the robot along a desired path to execute the task, was used to record 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. The 3D sensor's trace (represented as a temporal series of occupancy grids in 3D) represents 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.

5.1 Robot Results

Our planner was implemented in five different scenarios with the PR2 robot: opening a cabinet (45cm wide door), opening a drawer with an external handle attached (extends 30cm), opening an overhead kitchen cabinet (45cm wide door), opening a freezer (61cm door), and opening a bread box (handle is 18cm offset from hinge). 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/box. 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 objects. Again, note that the robot could be in a different start state at the beginning of its motion for opening the objects 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. It should be noted that for the bread box trial, roll and pitch of the gripper were added as additional degrees of freedom (bringing the search space to 11 dimensions).

Table 1 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 two cases, 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 1 Planning times in seconds for opening a file drawer, Ikea cabinet, overhead kitchen cabinet, freezer, and bread box.

	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
Bread Box	1.04	(unable to plan)

5.2 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.

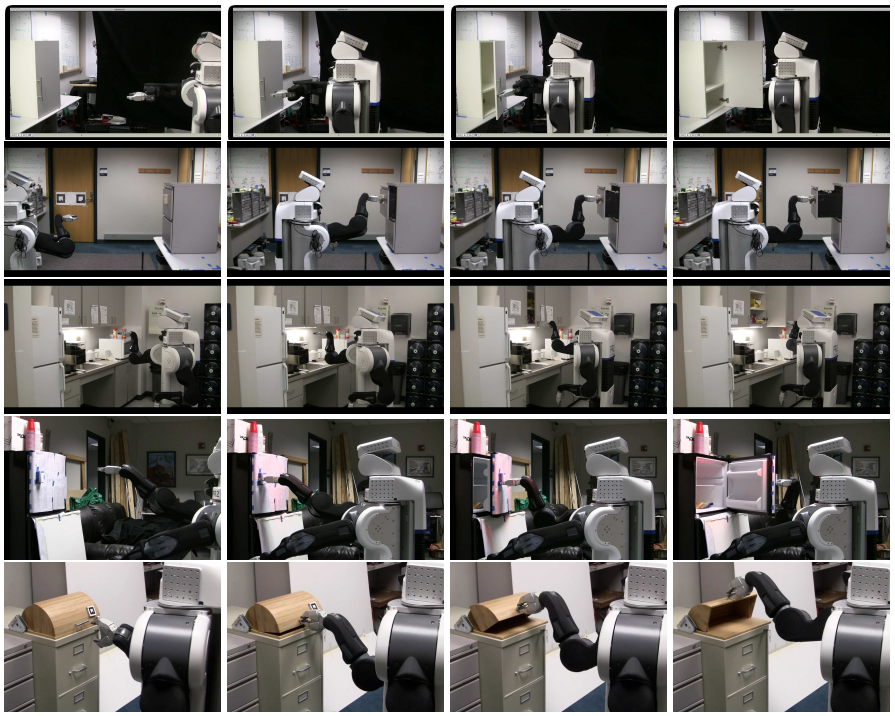


Fig. 4 PR2 opening an Ikea cabinet, metal drawer, overhead kitchen cabinet, freezer door, and bread box, respectively. Each sequence shows the execution of the completely planned full-body motion which approaches, grasps, and opens each object. The numerical results of these real robot experiments are shown in Table 1.

Table 2 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.

Table 2 A comparison between E-Graphs and weighed A* over 35 simulations

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

We also compared against Constrained Bi-directional Rapidly-Exploring Random Tree (CBiRRT), which 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 [4]. Like most RRT algorithms this method 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

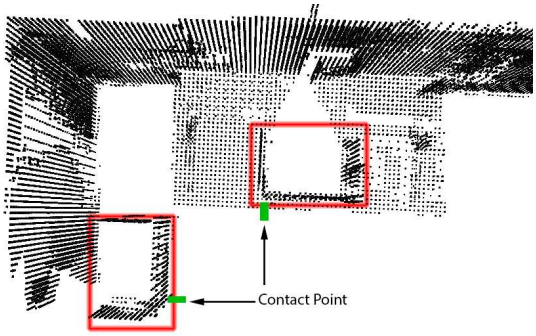


Fig. 5 The simulation environment. The red boxes represent example locations of the target object to be manipulated. The green boxes represent the contact point that the robot gripper should attempt to grasp.

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 does not 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 3 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	

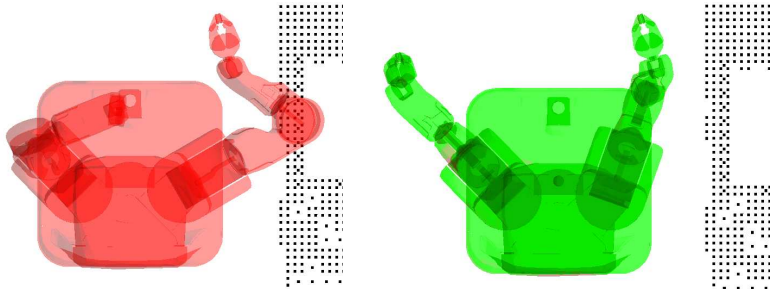
Table 3 compares E-Graphs to CBiRRT. In the first section of the table we can see that the planning times for the two approaches are similar for simpler scenarios though E-Graphs perform better on others (across 35 trials). We expect E-Graphs to continue to perform better than sampling planners as tasks 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 [24] 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.

5.3 Using a Partially Valid Demonstration

This scenario demonstrates the capability of using a partial E-Graph. An 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 4 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).



(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

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.

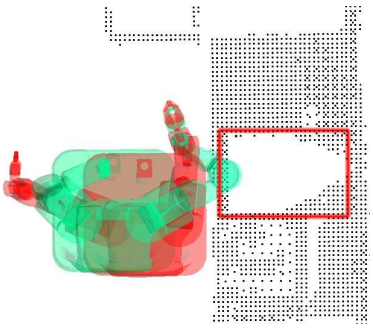
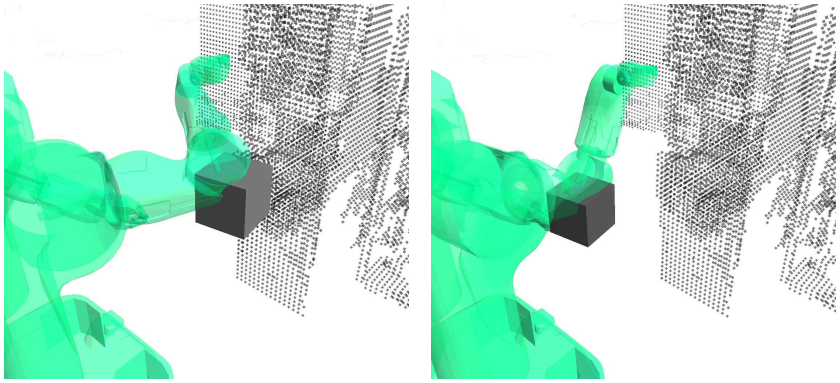


Fig. 7 A similar solution is found when planning from scratch.

Table 4 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



(a) A demonstration with the elbow to the right. The optional "Right Obstacle" is shown also.

(b) A demonstration with the elbow down. The optional "Under Obstacle" is shown also.

Fig. 8 Two different demonstrations for opening a drawer. The gray cube in each picture is an obstacle used in some of the experiments which was chosen to block part of only its corresponding demonstration.

5.4 Multiple Demonstrations

In this experiment we show how several demonstrations can be used to teach the planner to manipulate an object. Our results indicate that this leads to an increased level of robustness to additional obstacles and clutter. Specifically, in this experiment we provide two different demonstrations for opening a drawer. Figure 8 shows the last configuration in each of the two demonstrations. Figure 8a corresponds to a demonstration where the elbow is held out to the right while pulling open the drawer. This figure also shows the optional "Right Obstacle" which is used in some of our trials. When this obstacle is used, only this demonstration is blocked. On the other hand, Figure 8b depicts a demonstration where the elbow is held downward. The figure also shows the optional "Under Obstacle" which is used in some of our trials. This obstacle only blocks this elbow down demonstration.

We ran experiments in 3 different environments (no added obstacle, adding the under obstacle, and adding the right obstacle). Additionally, for each of these we tried all combinations of providing demonstrations (elbow to the right, elbow down, both demonstrations, and no demonstrations). In the case with no demonstration, the planner was still provided the trajectory that the contact point needs to follow, but specific configuration space trajectories were not added to the E-Graph.

Table 5 shows the results of our experiments. We can see that opening the drawer in this scenario was sufficiently difficult that without any demonstration (the "None" row) the planner was unable to find a solution within 60 seconds, which was our chosen timeout. The elbow right demonstration allows us to plan except when an obstacle is added on the right to invalidate part of the E-Graph. Similarly, the elbow down demonstration works well unless the under

obstacle is added. By providing the planner with both demonstrations, the E-Graph heuristic guides the search to follow the demonstration that allows it to avoid following the less informative original heuristic as much as possible. In this case, that results in the planner avoiding the demonstration where some poses have been invalidated by the newly introduced obstacle. Therefore, by providing multiple demonstrations, the planner can become more robust to added clutter.

Table 5 Performance with multiple demonstrations

Demonstrations	No obstacle	Right Obstacle	Under Obstacle
Elbow right	1.37	(unable to plan)	9.35
Elbow down	1.52	3.69	(unable to plan)
Both	1.16	2.88	9.81
None	(unable to plan)	(unable to plan)	(unable to plan)

5.5 Comparison against another reuse planner

In this section we compare planning with Experience Graphs to another motion planner that leverages reuse, Lightning [3]. Lightning, is a portfolio method which runs two planners in parallel, a planning from scratch (PFS) method and a retrieve-and-repair method (RR). Lightning terminates when either of the methods succeed and the resulting path is put back into a path database which the RR module can draw upon during future planning episodes. The PFS can use any planner but in the authors' implementation they use RRT-Connect [11], which is considered to be one of the faster sampling-based planners. The RR method first selects a small number of paths from a database which may solve the problem quickly (paths that have endpoints near the start and goal). It augments the paths by adding segments which connect the start and the goal to the path (e.g., by linear interpolation). It then collision checks the small set of augmented paths and selects the one with the least invalid portions. This path will be repaired by using a motion planner to reconnect broken sections. Again, while any planner could be used, RRT-Connect is chosen for the authors' implementation.

We compare against Lightning on single arm manipulation planning for the PR2 robot (7 degree of freedom), for which, the authors provide an implementation. We used a simulated kitchen environment with 100 randomly chosen starts and goals in difficult areas as shown in Figure 9. We used a weighted A* based arm planner [7] and ran it both with and without using E-Graphs. Finally, we made a planning portfolio more similar to Lightning. We ran the weighted A* planner (without E-Graphs) in parallel with the same planner using E-Graphs. Our planner uses a fast collision checker that represents the robot as a set of spheres. We set up Lightning to use the same fast

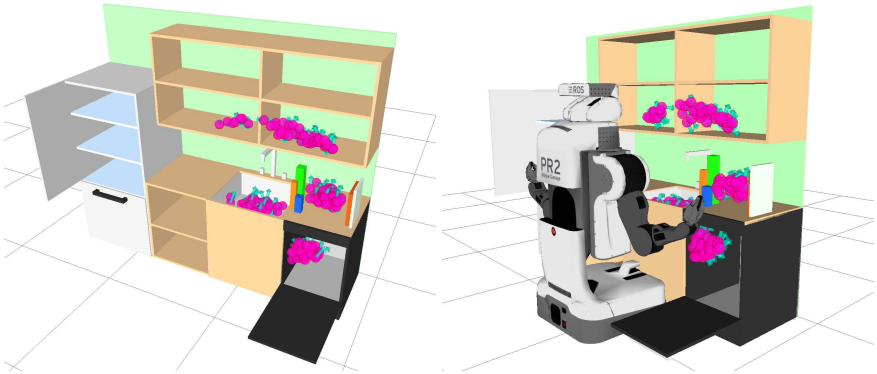


Fig. 9 The pink spheres show the location of the gripper for the start and goals states of our experiments. Planning is performed with the right arm.

collision checker. Both E-Graphs and Lightning start out with no prior experience but both are allowed to remember every path they generate (so by the end, both methods have around 100 paths). We give methods 10 seconds to plan, after which it is considered a failure. A shortcutter is run on both methods after planning is finished (included in planning times). Additionally, the environment does not change between queries and therefore previous paths don't need to be collision checked (though additions to those paths to solve new queries do). Lightning collision checks previous paths before using them so we disabled this in order to not have it do unnecessary computation. We used the default parameters for Lightning which include using 1 thread for the PFS planner and 4 threads to collision check potential paths for the RR planner.

The results of the experiments are shown in Table 6. We can see that E-Graphs has the fastest median planning time among Lightning, and weighted A*, though there are a number of outliers which give it a worse mean time than Lightning. However, by using a simple portfolio with weighted A* and E-Graphs, (much like Lightning with PFS and RR), many of the outliers are eliminated. Our portfolio has slightly lower mean planning time than Lightning, substantially lower median planning time and uses only 2 threads compared to Lightning's 5.

Lightning has slightly better success rate that E-Graphs and better path quality, while weighted A* has the best path quality. The weighted A* planner is minimizing the L2 norm in joint space, which is essentially what the shortcutting does as well. The E-Graph path quality is partially worse due to the planner going out of its way to reuse previous paths. Additionally, the chosen heuristic is a 3D grid search which guides the end effector to the end effector location on previous paths. This however is not enough to get the arm on the E-Graph. After reaching the gripper position from a previous path the planner needs to fix the gripper orientation and arm redundancy before it can

Table 6 Comparing E-Graphs, weighted A* and Lightning

	mean time(s)	std dev time(s)	median time(s)	mean arm motion (rad)	success of 100	threads used
Weighted A*	0.84	1.31	0.4	6.62	87	1
E-Graphs	0.76	1.39	0.17	11.40	95	1
Portfolio	0.34	0.77	0.13	10.39	95	2
Lightning	0.36	0.33	0.33	7.64	99	5
	Consistency(m)					
Weighted A*	4.85					
E-Graphs	4.82					
Lightning	14.30					

follow the previous path. This results in a relatively short end effector path while not getting a short path in joint space. This could be resolved by using a different heuristic h^G, such as euclidean distance in joint space.

We also ran a consistency experiment. As discussed earlier, consistency measures how similar paths from a planner are for similar inputs (start, goal, environment). Planners that create similar paths for similar inputs are more predictable, which is desirable when robot operate near people. We chose 100 random starts and goals from two regions (the start and goal are never from the same region).

Again we used the dynamic time warping (DTW) similarity metric [24] to compare the methods. DTW computes a similarity score for a pair of paths. To do this, it first aligns the two paths by finding corresponding waypoints between the two paths. Each waypoint will have at least 1 correspondence in the other path (but they can have several). After correspondences have been determined, the distances from each correspondence are summed. The similarity score is therefore, the sum of the distances between the waypoints on the paths after they have been aligned. Having a value closer to 0 means the paths are more similar as the distances between corresponding waypoints are small. Since this method is for comparing pairs of paths, we computed the average pair-wise similarity (we computed DTW on all pairs of paths and then averaged the scores). The waypoints in the paths used for the DTW computation are the location of the end-effector, and therefore, the scores reported are in meters. Both E-Graphs and weighted A* exhibit much more consistent behavior than Lightning. In fact we can see that they are about 3 times more consistent, meaning that Lightning tends to have 3 times as much distance between pairs of paths. Figure 10 visually shows the paths produced by E-Graphs and Lightning. The images clearly show that E-Graphs has significantly less variance.

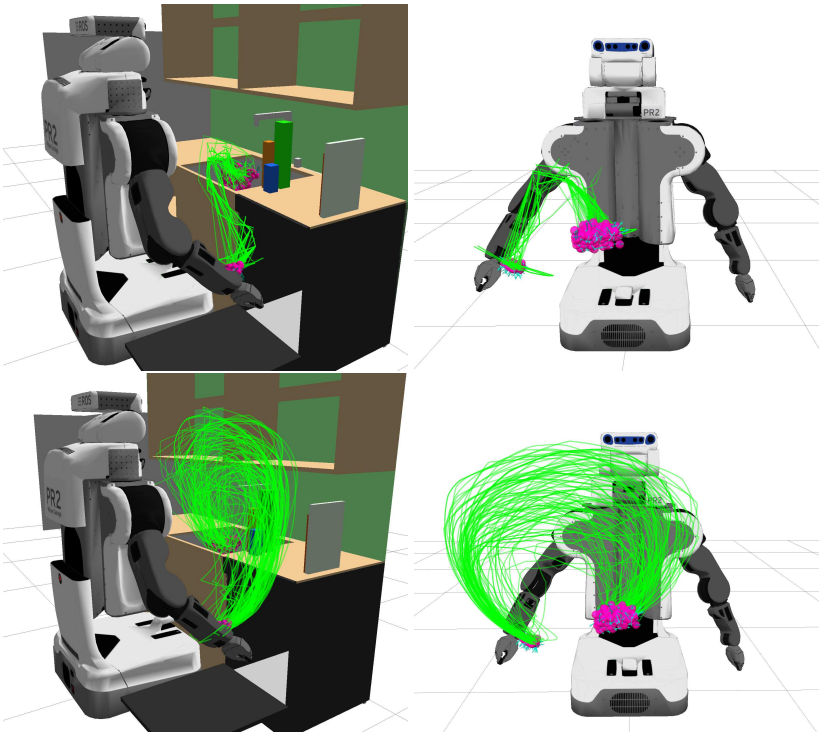


Fig. 10 The top row shows all the paths produced by E-Graphs in the consistency experiment. The bottom row shows those produced by Lightning. The pink spheres show the start and goal locations of the gripper. The green lines are motion that the gripper traced during each path.

6 Discussion

This method provides a novel framework to teach robots how to manipulate constrained objects by demonstration and then incorporate those demonstrations into a planner using Experience Graphs. However, there are several ways this method could be improved and extended.

In this work, we deal with objects which can be manipulated on a one dimensional manifold. We chose to represent this as a discretized curve that the contact point on the robot must follow to manipulate the object. This representation is simple, yet very expressive. While in our examples we used objects that have commonly modeled joints (e.g., revolute and prismatic), the representation supports arbitrary joints that exist on one dimensional manifolds, including those that are more difficult to capture with a few model parameters, such as a garage door or moving a toy train along tracks. The chosen representation also offers the ability to demonstrate how to use new objects very easily. A non-expert can give a demonstration as no programming or modeling needs to be done (the motion of the contact point and the z di-

mension is automatically computed and recorded). This being said, there are drawbacks. One of them is that the minimim and maximum extent to which the user demonstrated moving the object also defines the limits that planner has. For example, if the demonstration only opens the cabinet halfway, the planner will never be able to generate a plan that opens it all the way as it is not represented. Whereas, a parametric method may be able to hypothesize about how to continue moving the object beyond where the demonstration ended.

Additionally, the approach assumes that the objects we manipulate exist on one dimensional manifolds. We made this simplification because many objects fall into this category, but also because the entire range of motion of the object can be shown with a single demonstration. However, some objects in constrained manipulation exist on multi-dimensional manifolds. For instance, a robot sweeping a pile of dirt out of a room, where the dirt is constrained to stay on the floor. Our planner can handle these kinds of objects but it would always have to follow one of the contact trajectories provided. The planner would really be viewing the object as having a set of one dimensional manifolds which can all accomplish the task. The planner could be extended to truly manipulate multi-dimensional manifolds, but it would require a generalization step that takes a set of demonstrations and infers what manifold they are drawn from. This potentially could cause the planner to produce invalid paths when this generalization is incorrect and therefore, might require feedback and potentially more examples from the teacher.

Along similar lines, it would be interesting to see if demonstrations for a particular object can be generalized to other objects. For instance, if a demonstration is given on one cabinet, can we make use of that demonstration to open a door that is a little wider? When given a novel object that the user has not provided a kinesthetic demonstration for, we are missing two things: the contact point trajectory for how to operate the object and a robot configuration space trajectory which can be used in an Experience Graph. The second of these, is not actually required but as shown in our experiments, can greatly accelerate the planning process. However, given a contact point for the object, it may be possible to adapt a similar robot trajectory using a projection method like Jacobian pseudo-inverse [25]. When projecting a prior demonstration to a new (but hopefully similar) object, some parts may fail due to obstacles or joint limits, and therefore, the E-Graph will be seeded with a partial robot trajectory. However, as was shown in our experiments, even this can be beneficial in accelerating the planner.

In terms of scalability, providing a kinesthetic demonstration for how to operate every object in a home is limiting. However, it may be possible to extract a contact point trajectory for an object by observing how a human performs tasks in the home (by tracking the motion of their hand). This would be less time consuming for the human.

In our multiple demonstration experiment we show that having more demonstrations can increase robustness. Specifically, we would block part of one demonstration at a time and showed how we were still able to solve the prob-

lem by using a different (completely valid) demonstration. However, there may be cases where every demonstration is partially invalidated by added clutter. In these cases, each demonstration could look equally helpful and in weighted A* type planner, what generally happens is that each route is fully explored before moving on to another. It may be easier to find a path to the goal from some demonstrations than others (e.g., perhaps some demonstrations are easier to modify while keeping the same contact point trajectory). Therefore, the time it takes to plan could vary wildly depending on if the planner chooses (somewhat arbitrarily) a difficult or easy demonstration to explore first. One way to alleviate this might be to employ a version of A* which searches multiple "branches" of the search space at the same time. One way to do this might be to use a different heuristic for each demonstration [8, 1]. Multiple demonstrations could also be searched using parallel planners [29]. There are also parallel versions of weighted A* which could be applied directly to the algorithm used in this article [6, 21].

While our method allows the contact point on the object to be different with each demonstration, we make the assumption that the contact point on the robot stays constant across all demonstrations (the point we compute forward kinematics for). This may be limiting in scenarios where the robot must alternate between using its two grippers to operate the object or in the case of opening a spring-loaded door, when people often use contact with their body to push it open. We would like to relax this restriction in the future to support these kinds of tasks.

It would be interesting to see how much solution quality improves (at the expense of planning time) when using a smaller value of ε^E. The anytime version of Experience Graphs could be applied to reach a compromise between planning time and quality [20].

Finally, while plans produced by the planner are valid with respect to the input they are given, the final execution of the path is not always right. This can happen because the location of the object we are going to manipulate has some error due to sensor noise or during execution, the localization of the robot drifts. Grabbing a small handle on the order of centimeters is difficult after driving distances on the scale of meters. To combat this error we've attached a visual fiducial (AR marker) to the objects that we use to correct pose error as we get close to grasping the object. However, there are situaitons where this could lead to a sub-optimal execution or even collision. A more principled approach might be to replan regularly during execution to account for drift in the path following. This can be done efficiently using Experience Graphs, as we have shown in prior work [20].

7 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 need to be useful). Experimentally we provide results on high dimensional mobile manipulation tasks using the PR2 robot to open cabinets, freezers, bread boxes, 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.

Acknowledgements 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. Aine, S., Swaminathan, S., Narayanan, V., Hwang, V., Likhachev, M.: Multi-heuristic A*. In: Proceedings of Robotics: Science and Systems. Berkeley, USA (2014)
2. Argall, B., Chernova, S., Veloso, M.M., Browning, B.: A survey of robot learning from demonstration. Robotics and Autonomous Systems **57**(5), 469–483 (2009)
3. Berenson, D., Abbeel, P., Goldberg, K.: A robot path planning framework that learns from experience. In: ICRA (2012)
4. Berenson, D., Srinivasa, S., Ferguson, D., Kuffner, J.: Manipulation planning on constraint manifolds. In: IEEE International Conference on Robotics and Automation (ICRA '09) (2009)
5. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2002)
6. Burns, E., Lemons, S., Ruml, W., Zhou, R.: Best-first heuristic search for multicore machines. Journal of Artificial Intelligence Research **39**, 689–743 (2010)
7. Cohen, B.J., Chitta, S., Likhachev, M.: Single- and dual-arm motion planning with heuristic search. IJRR **33**(2), 305–320 (2014)
8. Helmert, M.: The fast downward planning system. J. Artif. Intell. Res. (JAIR) **26**, 191–246 (2006)
9. Jetchev, N., Toussaint, M.: Trajectory prediction: Learning to map situations to robot trajectories. In: IEEE International Conference on Robotics and Automation (2010)
10. Jiang, X., Kallmann, M.: Learning humanoid reaching tasks in dynamic environments. In: IEEE International Conference on Intelligent Robots and Systems (2007)
11. Jr., J.J.K., LaValle, S.M.: Rrt-connect: An efficient approach to single-query path planning. In: ICRA (2000)
12. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation **12**(4), 566–580 (1996)
13. Kober, J., Peters, J.: policy search for motor primitives in robotics. In: advances in neural information processing systems 22 (nips 2008), cambridge, ma: mit press (2009)
14. Koenig, S., Likhachev, M.: D* lite. In: AAAI, pp. 476–483 (2002)
15. Kormushev, P., Calinon, S., Caldwell, D.G.: Robot motor skill coordination with EM-based reinforcement learning. In: Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS), pp. 3232–3237. Taipei, Taiwan (2010)
16. Likhachev, M., Gordon, G., Thrun, S.: ARA*: Anytime A* with provable bounds on sub-optimality. In: Advances in Neural Information Processing Systems (NIPS) 16. Cambridge, MA: MIT Press (2003)
17. Oriolo, G., Mongillo, C.: 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, pp. 2154–2160. IEEE (2005)
18. Pastor, P., Hoffmann, H., Asfour, T., Schaal, S.: learning and generalization of motor skills by learning from demonstration. In: international conference on robotics and automation (icra2009) (2009)

19. Phillips, M., Cohen, B.J., Chitta, S., Likhachev, M.: E-graphs: Bootstrapping planning with experience graphs. In: Robotics: Science and Systems (2012)
20. Phillips, M., Dornbush, A., Chitta, S., Likhachev, M.: Anytime incremental planning with e-graphs. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) (2013)
21. Phillips, M., Likhachev, M., Koenig, S.: Pa*se: Parallel A* for slow expansions. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2014)
22. Pohl, I.: First results on the effect of error in heuristic search. Machine Intelligence **5**, 219–236 (1970)
23. Porta Pleite, J.M., Jalliet, L., Bohigas Nadal, O.: Randomized path planning on manifolds based on higher-dimensional continuation **31**(2), 201–215 (2012)
24. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing **26** (1978)
25. Sciavicco, L., Siciliano, B., Sciavicco, B.: Modelling and Control of Robot Manipulators, 2nd edn. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2000)
26. Stentz, A.T.: The focussed d* algorithm for real-time replanning. In: Proceedings of the International Joint Conference on Artificial Intelligence (1995)
27. Stolle, M., Atkeson, C.: Policies based on trajectory libraries. In: IEEE International Conference on Robotics and Automation (2006)
28. Sucan, I.A., Chitta, S.: Motion planning with constraints using configuration space approximations. IEEE, Vilamoura, Algarve, Portugal (2012)
29. Valenzano, R., Sturtevant, N., Schaeffer, J., Buro, K.: Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In: International Conference on Automated Planning and Scheduling (2010)
30. Yang, Y., Brock, O.: Elastic roadmaps - motion generation for autonomous mobile manipulation. Auton. Robots **28**(1), 113–130 (2010)
31. Zucker, M., Kuffner, J., Branicky, M.: Multipartite rrts for rapid replanning in dynamic environments. In: IEEE International Conference on Robotics and Automation (2007)