Anytime Incremental Planning with E-Graphs

Mike Phillips[1] and Andrew Dornbush[1] and Sachin Chitta[2] and Maxim Likhachev[1]

***Abstract*— Robots operating in real world environments need to find motion plans quickly. Robot motion should also be efficient and, when operating among people, predictable. Minimizing a cost function, e.g. path length, can produce short, reasonable paths. Anytime planners are ideal for this since they find an initial solution quickly and then improve solution quality as time permits. In previous work, we introduced the concept of Experience Graphs, which allow search-based planners to find paths with bounded sub-optimality quickly by reusing parts of previous paths where relevant. Here we extend planning with Experience Graphs to work in an anytime fashion so a first solution is found quickly using prior experience. As time allows, the dependence on this experience is reduced in order to produce closer to optimal solutions. We also demonstrate how Experience Graphs provide a new way of approaching incremental planning as they naturally reuse information when the environment, the starting configuration of the robot or the goal configuration change. Experimentally, we demonstrate the anytime and incremental properties of our algorithm on mobile manipulation tasks in both simulation and on a real PR2 robot.**

I. Introduction

Motion planning in real world human environments is a challenging task. Robots have to deal with changing environments, complex tasks, and account for the presence of humans. Mobile pick and place tasks can involve multiple steps: positioning the base of the robot, executing arm motions, grasping and navigation while carrying objects (Figure 1). Motion planning for these steps must be fast, to deal with human expectations for quick execution of tasks and also for efficient operation. Planning must also be accurate, avoiding collisions with the environment and executing tasks correctly.

Human environments, even seemingly unstructured environments like households, retain some amount of structure that can be exploited for fast motion planning. An example of this is a typical kitchen environment. The locations of individual objects will change a lot in such an environment. However, most objects will be found on counters, inside drawers, on tables, inside cabinets, etc., i.e. the underlying structure of the environment does not change. Motion planning in such environments is therefore often repetitive.

In previous work [1], we introduced the concept of Experience Graphs. Experience Graphs attempt to retain information from previous successful motion plans (or demonstrations of paths for a particular task). We showed how search-based planners can be modified to use Experience Graphs, using previous information when possible to speedup motion planning significantly while still gracefully falling back to planning from scratch when necessary. We demonstrated the results of this approach by applying it to problems of mobile pick and place in a kitchen environment.

Fig. 1: Motion planning is often used to compute motions for complex tasks such as dual-arm mobile manipulation.

In this work, we build on the concept of Experience Graphs and extend it to allow for anytime planning and incremental planning. Anytime planners are designed to return an initial solution quickly and then improve solution quality as time permits. Anytime planning allows robots to start moving quickly using an initial plan while giving the planner more time to refine the plan as time goes on. This allows robots to be more efficient by parallelizing and interleaving planning and execution. We present two approaches that allow for making Experience Graphs anytime and show how the approaches may be more suitable in different situations.

We also show how Experience Graphs can naturally be used for incremental planning. Incremental planning involves situations where the environment that the robot is operating in changes slightly or the goal or start configuration of the robot changes. In such cases, Experience Graphs naturally reuse as much information as they can while still dealing with the changes. We present both simulated and experimental results to show how our approach can be used for anytime motion planning with a mobile manipulation robot. We demonstrate the benefits of the incremental nature of the planner by showing how it can quickly replan when the goal configuration of the robot changes.

II. Related Work

Motion planning, especially for mobile manipulation, has seen considerable interest recently. Most of the approaches were initially focused on treating each new motion planning request as a fresh request for planning. There was little, if any, reuse of information from experience gained while carrying out a series of motion plans. However, recent work has seen more reuse of previous information. In [2], Lien et. al. constructed roadmaps for obstacles, stored them in a database and reused them during motion planning while Bruce et. al. [3] biased an RRT search towards waypoints remembered from previous motion planning attempts. Related work can also be found in [4], [5].

*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.

[1]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA
[2]Willow Garage Inc. Menlo Park, CA

Other approaches to exploiting experience has included the use of trajectory libraries. These libraries were used to adapt policies for control of underactuated systems and high-dimensional systems in [6], while in [7], new trajectories were generated by combining nearby trajectories. A different application of such techniques can also be found in [8] while transfer of policies across tasks was presented in [9]. The reuse of environment information, coupled with information about previous motion plans, was presented in [10] where machine learning methods were coupled with paths stored in a database to generate new motion plans based on the nature of the environment and the types of obstacles.

Jiang et. al. [11] present an approach to using a database of older motion plans to draw a bi-directional RRT search towards a path stored in the database that is most similar to the new motion plan request. Recent work [12] attempts to repair previous plans from a database using randomized planners. As mentioned in [1], the use of a database of motion plans is also core to Experience Graphs. Experience Graphs attempt to use all the information from previous experiences instead of attempting to find the nearest or closest motion plan in a database. They are thus capable of reusing many parts of the previous experiences when possible.

Anytime versions of several planners have also been explored in previous work. ARA* presented in [13] is an anytime version of the A* algorithm that iteratively improves its solution and and sub-optimality bound on solution quality. The AD* algorithm [14] is an anytime incremental search-based planner built on the D* planner and is capable of efficiently dealing with small changes in the environment and replanning without having to plan from scratch. The RRT* planner [15] is an incremental anytime randomized planner that has been shown to work well for manipulation and navigation tasks.

Search-based planning with Experience Graphs offers several advantages over these approaches. It does not rely on a particular representation of the environment. It also builds the graph using paths from prior tasks in contrast to approaches like Probabilistic Roadmaps [16] which rely on sampling the whole space. We refer the interested reader to [1] for more details on the merits of using Experience Graphs. The addition of anytime capabilities in this work allows us to handle realtime motion planning problems better.

III. Experience Graphs

A. Overview

This section gives an overview of our previous work on Experience Graphs (we refer the interested reader to [1] for more details). An *Experience Graph* or E-Graph is a graph formed from the solutions found by the planner for previous planning queries or from demonstrations. We will abbreviate this graph as G^E. The graph G^E is incomparably smaller than graph G used to represent the original planning problem. At the same time, it is representative of the connectivity of the space exercised by the previously found motions. The key idea of planning with G^E is to bias the search efforts, using a specially constructed heuristic function, towards finding a way to get onto the graph G^E and to remain searching G^E rather than G as much as possible. This avoids exploring large portions of the original graph G. In the following we explain how to do this in a way that guarantees completeness and bounds on solution quality with respect to the original graph G.

B. Definitions and Assumptions

First, we will list some definitions and notations that will help explain our algorithm. We assume the problem is represented as a graph where a start and goal state are provided (s_{start}, s_{goal}) and the desired output is a path (sequence of edges) that connect the start to the goal.

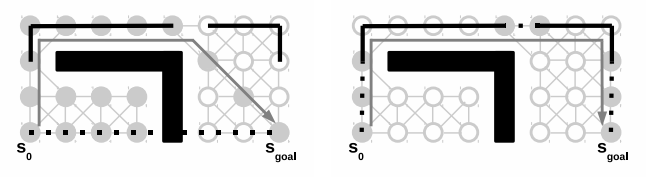
- $G(V^G, E^G)$ is a graph modeling the original motion planning problem, where V^G is the set of vertices and E^G is the set of edges connecting pairs of vertices in V^G.
- $G^E(V^E, E^E)$ is the E-Graph that our algorithm builds over time ($G^E \subseteq G$).
- $c(u, v)$ is the cost of the edge from vertex u to vertex v
- $c^E(u, v)$ is the cost of the edge from vertex u to vertex v in graph G^E and is always equal to $c(u, v)$

Edge costs in the graph are allowed to change over time (including edges being removed and added which happens when new obstacles appear or old obstacles disappear). The algorithm is based on heuristic search and is therefore assumed to take in a heuristic function $h^G(u, v)$ estimating the cost from u to v ($u, v \in V^G$). We assume $h^G(u, v)$ is admissible and consistent for any pair of states $u, v \in V^G$. An admissible heuristic never overestimates the minimum cost from any state to any other state. A consistent heuristic $h^G(u, v)$ is one that satisfies the triangle inequality, $h^G(u, v) \leq c(u, s) + h^G(s, v)$ and $h^G(u, u) = 0$, $\forall u, v, s \in V^G$ and $\forall (u, s) \in E^G$.

C. Experience Graphs

The planner maintains two graphs, G and G^E. Every time the planner receives a new planning request the *findPath* function (shown below) is called. It first updates G^E to account for edge cost changes due to differences between the current planning episode and previous planning episodes. If any edges are invalid (e.g. they are now blocked by obstacles) they are put into a disabled list. Conversely, if an edge in the disabled list now has finite cost it is re-enabled. At this point, the graph G^E should only contain edges with finite costs. Then it calls the *computePath* function, which produces a path π. This path is then added to G^E for future reuse.

findPath(s_{start}, s_{goal})
1: $updateChangedCosts()$
2: disable edges that are now invalid
3: re-enable disabled edges that are now valid
4: $precomputeShortcuts()$
5: $\pi = computePath(s_{start}, s_{goal})$
6: $G^E = G^E \cup \pi$



(a) $\varepsilon^E = 1$ (b) $\varepsilon^E \to \infty$

Fig. 2: Effect of ε^E. The dark solid lines are paths in G^E while the dark dotted lines are best paths from s_0 to s_{goal} according to h^E. Note as ε^E increases, the heuristic prefers to travel on G^E. The light gray circles and lines show the graph G and the filled in gray circles show the expanded states when planning with E-Graphs. The dark gray arrow shows the returned path.

The speed-up of planning with Experience Graphs comes from being able to reuse parts of old paths and avoid searching large portions of graph G. To accomplish this we introduce a heuristic which intelligently guides the search toward G^E when it looks like following parts of paths in G^E will help the search get close to the goal. We define a new heuristic h^E in terms of the given heuristic h^G and edges in G^E for any state $s_0 \in V^G$:

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

where π is a path $\langle s_0 \dots s_N \rangle$ and $s_N = s_{goal}$ and ε^E is a scalar ≥ 1.

Equation 1 returns the cost of the minimal path from the queried state s_0 to the goal where the path consists of an arbitrary number of two kinds of segments. The first type of segment corresponds to an instantaneous jump between between s_i and s_{i+1} at a cost equal to the original heuristic inflated by ε^E (this is equivalent to saying all states can reach all of the other states in G according to the original heuristic inflated by ε^E). The second kind of segment is an edge from G^E and it uses its actual cost c^E (∞ if the edge does not exist in G^E). As the penalty term ε^E increases, the heuristic path from s_0 to the goal will go farther out of its way to travel toward the goal using E-Graph edges.

The larger ε^E is, the more the actual search avoids exploring G and focuses on traveling on paths in G^E. Figure 2 demonstrates how this works. As ε^E increases, the heuristic guides the search to expand states along parts of G^E. In Figure 2a, the heuristic causes the search to ignore the graph G^E because without inflating h^G at all ($\varepsilon^E = 1$), the heuristics will never favor following edges in G^E. This figure also shows how during the search, by following G^E paths, we can avoid obstacles and have far fewer expansions. The expanded states are shown as filled in gray circles, which change based on ε^E.

IV. Anytime E-Graphs

In this section we describe two novel anytime extensions to planning with E-Graphs. The *computePath* function (shown below) runs a modified version of ARA* [13]. Anytime Repairing A* runs a series of weighted A* searches with decreasing sub-optimality bounds until it produces the optimal solution. The result is that an initial solution can be found fast, and the quality of the solution improved as time allows. Weighted A* uses an inflated heuristic to make an A* search more focused, generally finding solutions significantly faster. The solution cost is guaranteed to be no more than the inflation factor times the optimal solution cost. The individual weighted A* searches (*improvePath*) use our E-Graph heuristic h^E and in addition to using the edges that G already provides (getSuccessors), we add two additional types of successors: *shortcuts* and *snap motions* (line 4 in $findPath$). Snap successors are optional dynamically generated motions that help the search connect to the E-Graph [1]. Shortcut successors will be covered later.

computePath(s_{start}, s_{goal})
1: $OPEN = CLOSED = INCONS = \emptyset$
2: $g(s_{start}) = 0; f(s_{start}) = fvalue(s_{start})$
3: $g(s_{goal}) = \infty; f(s_{goal}) = \infty$
4: insert s_{start} into $OPEN$ with $f(s_{start})$
5: $improvePath(s_{goal})$
6: publish solution with sub-optimality bound $= getBound()$
7: **while** not isOptimal() **do**
8: improveAnytimeParams()
9: move states from $INCONS$ into $OPEN$
10: $f(s) = fvalue(s), \forall s \in OPEN$
11: update priorities for all $s \in OPEN$ according to $f(s)$
12: $CLOSED = \emptyset$
13: $improvePath(s_{goal})$
14: publish solution with sub-optimality bound $= getBound()$
15: **end while**

improvePath(s_{goal})
1: **while** $f(s_{goal}) > min_{s \in OPEN}(f(s))$ **do**
2: remove s with the smallest f-value from $OPEN$
3: insert s in $CLOSED$
4: $S = getSuccessors(s) \cup shortcuts(s) \cup snap(s)$
5: **for all** $s' \in S$ **do**
6: **if** s' was not visited before **then**
7: $f(s') = g(s') = \infty$
8: **end if**
9: **if** $g(s') > g(s) + c(s, s')$ **then**
10: $g(s') = g(s) + c(s, s')$
11: **if** $s' \notin CLOSED$ **then**
12: $f(s') = fvalue(s')$
13: insert s' into $OPEN$ with $f(s')$
14: **else**
15: insert s' into $INCONS$
16: **end if**
17: **end if**
18: **end for**
19: **end while**

The use of the ARA* algorithm usually makes it easy to make an A* search anytime. ARA* assumes a consistent heuristic $h(s)$ which it inflates with a scalar $\varepsilon \geq 1$. Initially, ε is large, so a highly weighted A* search is used to find a solution quickly. After each search iteration ε is reduced and the solution will be upper bounded more tightly until $\varepsilon = 1$ and a provably optimal solution is found.

Extending E-Graphs to be anytime is not this simple since the heuristic h^E is already partly inflated by ε^E. If the heuristic was entirely scaled up by ε^E we could just factor it out and run ARA*. However, the heuristic is computed as the sum of costs: some of which are inflated by ε^E

(original heuristic costs) and some which are not (E-Graph edges) so the ε^E can't be pulled out. One straightforward option that we will name H_1 is to recompute h^E after each search iteration with a newly reduced value of ε^E. In our experiments we found this method to work quite well as it reduces dependence on the E-Graph in a smooth way. In our mobile manipulation experiments the heuristic doesn't take long to compute compared to the difficulty of the actual search so it is worth the recomputation overhead to have a more informative heuristic at each search iteration. To use this method with ARA*, the following functions used in $computePath$ and $improvePath$ are implemented as follows.

- $fvalue(s)$: return $g(s) + \varepsilon h^E(s)$
- $isOptimal()$: return $[\varepsilon = 1 \wedge \varepsilon^E = 1]$
- $getBound()$: return $\varepsilon \cdot \varepsilon^E$
- $improveAnytimeParams()$: if $\varepsilon^E > 1$, decrease ε^E; otherwise, decrease ε

Essentially, we reduce the E-Graph inflation ε^E after each search iteration until it equals 1. At that point the heuristic function no longer has any dependence on the E-Graph. After that the overall heuristic inflation ε is reduced until it also equals 1. After this, the solution is provably optimal.

For some planning problems the heuristic computation is too expensive to be computed many times during the search. Moreover, in some cases, the heuristic computation is so expensive that it can only be done offline [17]. We provide a second anytime E-Graph method, H_2, for such domains. Unlike H_1 which has to recompute the heuristic for each search iteration performed by ARA*, H_2 only computes the h^E heuristic once. In H_2 we compute h^E only for the initial value of ε^E. For a given search iteration, the H_2 heuristic is given by $max\left(\frac{1}{\delta}h^E(s), h^G(s)\right)$. δ is initialized to 1 and is increased after each iteration until it is equal to ε^E. Essentially, after each iteration δ gets larger and reduces the magnitude of h^E. Eventually, the original heuristic h^G becomes the larger heuristic value for all states (this is guaranteed once $\delta = \varepsilon^E$). Once δ has been increased so that it equals ε^E, the anytime search starts reducing ε until it equals 1. Once a search is run with these values, the solution is optimal. To use this method with ARA* the following functions in $computePath$ and $improvePath$ are implemented as follows.

- $fvalue(s)$: return $g(s) + \varepsilon max\left(\frac{1}{\delta}h^E(s), h^G(s)\right)$
- $isOptimal()$: return $[\varepsilon = 1 \wedge \delta = \varepsilon^E]$
- $getBound()$: return $\varepsilon \cdot \frac{\varepsilon^E}{\delta}$
- $improveAnytimeParams()$: if $\delta < \varepsilon^E$, increase δ; otherwise, decrease ε

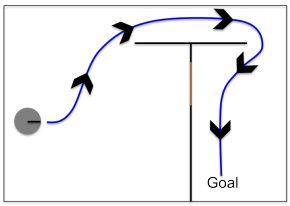
The H_2 method approximates H_1 while only having to compute h^E once. Our experiments show that the performance of H_2 is almost as good H_1 and therefore, may provide a suitable alternative in domains with expensive to compute heuristics.

In Experience Graphs, shortcut successors are generated when expanding a state $s \in G^E$. A shortcut successor uses G^E to jump to a place much closer to s_{goal} [1]. This shortcut may use many edges from various previous paths. The shortcuts allow the planner to quickly get near the goal without having to re-generate paths in G^E. Some of these optional shortcuts can be computed on line 4 of $findPath$ as shown in our prior work [1]. In our prior work, a shortcut for s generated the successor closest to s_{goal} according to the heuristic h^G on the same component of G^E as s. This works well for quickly generating a first solution as it gets the search as close as possible to the goal state. However, as the anytime algorithm improves solution quality, this shortcut quickly becomes too sub-optimal to be used. We therefore introduce new shortcut successors which are more likely to be within the asked sub-optimality bound by taking the heuristic for the current search iteration into account.

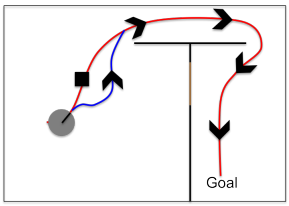
Ideally what we would like to see is as the sub-optimality bound comes down, shortcut successors are generated farther from the goal in places where continuing to follow the E-Graph is too out of the way (even if it does eventually get very close to the goal state). Conveniently, our heuristic h^E already encodes this information since the parameter ε^E represents how far out of the way the search is willing to go to use the E-Graph. We will use the heuristic to decide where to generate shortcuts. Since the new heuristic becomes less dependent on the E-Graph with each search iteration, shortcuts will automatically start getting generated in a way that travels less far on the E-Graph if it goes out of the way. To accomplish this, when a state s is expanded on the E-Graph, we perform a gradient descent on the E-Graph using the heuristic function H_1 or H_2 (depending on the method) until we reach a local minima. More specifically, we look at the states that s is connected to on the E-Graph, choose the neighbor with the smallest heuristic and continue the descent in that direction. When we reach a state s' where all the E-Graph neighbors have heuristic values greater or equal to that of s', we know we have reached a local minima. Then s' becomes the shortcut for s. For efficiency, we can cache s' as the shortcut for all the states we passed through during the gradient descent so each state is only passed through at most once for shortcut computations during a particular search iteration.

V. Incremental E-Graphs

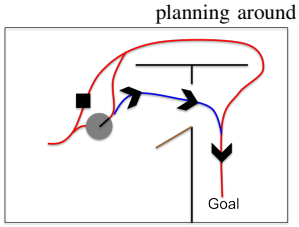
In this section, we will discuss the application of Experience Graphs to incremental planning. Incremental planners like D* [18] and D* Lite [19] are able to reuse computations from previous searches when the environment changes or when the starting state (the pose of the robot) changes. They do this by planning backward from the goal toward the start so that the root of the search tree is at the goal. When the start state changes the entire search tree is unaffected and the planner just needs to keep running until the new start state is connected (it may even be in the search tree already). On the other hand, if the goal state changes, the root of the search tree is moved and all previous computations are invalidated. When the environment changes, some parts of the tree are invalidated. Sometimes this can be quite significant and it can be faster to plan from scratch due



(a) An initial plan to the goal. Here the robot is shown at its start position as a circle.



(b) After encountering a new obstacle (square object in figure), the planner reconnects the robot to the rest of the E-Graph by planning around the obstacle.



(c) When the robot discovers an open door it thought was closed, h^E guides the search through the door instead of continuing to follow the old path, in order to stay within the sub-optimality bound. It is able to reconnect with part of the E-Graph on the other side of the door.

Fig. 3: An example of using E-Graph for incremental planning.

to the bookkeeping needed to update g-values of states that are affected by changed edges in the graph. Because of this bookkeeping, existing incremental graph searches are not well suited for planning problems that are higher than 3-4 dimensions. In contrast, E-Graphs are well-suited for high dimensional planning problems.

E-Graphs can be used to plan incrementally with minimal bookkeeping and can still replan using prior computations even when both the start and goal states change. Unlike D*, which plans optimal paths, E-Graphs are only beneficial when searching for paths within a given sub-optimality bound. E-Graphs handle incremental planning automatically just by feeding planned paths back into the E-Graph. Then if the start, goal, or both change, replanning is quick since the planner will just reconnect the states to parts of the E-Graph as long as it is within the bound on sub-optimality the user provides. An experiment showing this kind of scenario is in our real robot experiments. E-Graphs also handle changing environments. When the map is changed the algorithm runs through the E-Graph, disables edges that are now in collision and re-enables edges that are no longer in collision (the Experience Graph is incomparably smaller than the original graph especially since it is constructed in a task oriented way). For a sufficiently high bound, the planner will automatically repair the previous path if new obstacles broke the path. The planner will also shorten a previous path if an obstacle disappears, in order to stay within its sub-optimality bound. To demonstrate how E-Graphs can be used to handle a changing environment, we give a small example in Figure 3.

VI. Theoretical Analysis

Lemma 1: Heuristic H_1 is $\varepsilon \cdot \varepsilon^E$-consistent.
This heuristic was used in our prior work [1] where we proved the $\varepsilon \cdot \varepsilon^E$-consistency.

Theorem 1: For a finite graph G, the anytime planner using H_1 terminates, and the solution published after each iteration is guaranteed to be no worse than $\varepsilon \cdot \varepsilon^E$ times the optimal solution cost in graph G.
Given the $\varepsilon \cdot \varepsilon^E$-consistency Lemma 1, the upper bound provided after each iteration of the algorithm follows from the ARA* proofs since we are just running anytime repairing A* with a custom rule for decrementing the bound between iterations.

Lemma 2: Heuristic H_2 is $\varepsilon \cdot \frac{\varepsilon^E}{\delta}$-consistent.
From our previous work [1], we know the $\varepsilon \cdot h^E$ heuristic is $\varepsilon \cdot \varepsilon^E$-consistent. It follows then that $\frac{\varepsilon}{\delta} \cdot h^E$ heuristic is $\frac{\varepsilon}{\delta} \cdot \varepsilon^E$-consistent

Since h^G is actually consistent, it is trivially $\varepsilon \cdot \frac{\varepsilon^E}{\delta}$-consistent. Then since the max of two α-consistent heuristics is α-consistent, we can combine the two heuristics to show that H_2 is $\varepsilon \cdot \frac{\varepsilon^E}{\delta}$-consistent.

Theorem 2: For a finite graph G, our anytime planner using H_2 terminates, and the solution published after each iteration is guaranteed to be no worse than $\varepsilon \cdot \frac{\varepsilon^E}{\delta}$ times the optimal solution cost in graph G.
Due to Lemma 2, we can apply the same argument as for Theorem 1.

VII. Experimental Results

Search-based planners are challenged by having to plan in higher-dimensional spaces. Full-body planning for the PR2 robot involves planning with a high number of degrees of freedom. In our experiments, we tested the planners for mobile pick and place tasks with the PR2 robot where the robot is carrying objects in an upright orientation. We assume that the two end-effectors of the PR2 robot are rigidly attached to the object. The planning space is 10 dimensional: the robot's position and orientation (yaw) in a global frame constitute 3 degrees of freedom, the redundant degrees of freedom for the two arms constitute 2 more degrees of freedom, the 3D position and yaw of the object comprise 4 more degrees of freedom while the height of the telescoping spine of the robot constitutes the 10^{th} degree of freedom. Motions for the arms are computed in workspace of the object (they can vary the x, y, z position or the yaw). Inverse kinematics is used to check for feasibility.

h^G is represented by a 3D Dijkstra heuristic for a sphere inscribed in the object carried by the robot. The heuristic accounts for collisions between this object and obstacles in the environment but does not account for other constraints, e.g. the workspace of the arms, internal collisions, or collisions between the body of the robot and the environment. All experiments were carried out with $\varepsilon = 2$ and $\varepsilon^E = 10$. Thus, the initial sub-optimality bound is 20. For H_1, ε^E was decremented by 1 after each iteration and after it reached 1, ε was decremented by 0.2. For H_2, δ was incremented by 1 after each iteration and after it reached ε^E, ε was decremented in a similar manner to H_1.

The two anytime E-Graph methods (one with heuristics computed according to H_1 and the other one according to H_2) were compared to each other based on how quickly

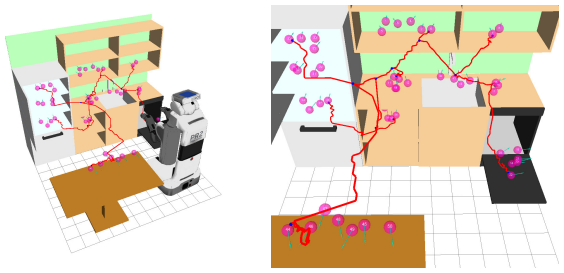
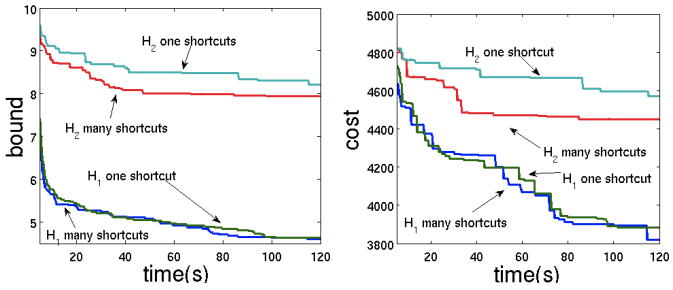


Fig. 4: Full-body planning in a kitchen scenario (G^E after bootstrap goals)



(a) Average bound over time (b) Average cost over time

Fig. 5: Anytime profiles for full-body planning in a kitchen scenario

they improve the path quality as they are given more time. We also compared anytime E-Graphs against ARA* (with an equivalent $\varepsilon = 20$) and RRT* [15] (an anytime sampling-based planner which reaches optimality in the limit) on time to first solution, and the initial and final path quality using various metrics. We used the implementation of RRT* in the OMPL [20] (Open Motion Planning Library). Its paths were post-processed using OMPL's shortcutter. All planners operate in the previously defined configuration space and each was given 2 minutes to find a solution and improve it.

A. Simulation

Tests were run in a simulated kitchen environment (the tests were similar to those in [1]). 50 goals were chosen in locations where objects are often found (e.g. tables, countertops, cabinets, refrigerator, dishwasher). 10 representative goals were chosen to bootstrap the planner, which was then tested against the remaining 40 goals. The bootstrap plans were done to the first solution so both anytime methods had the same E-Graph. Figure 4 shows G^E after bootstrapping. To ensure the two anytime methods had the same E-Graph throughout the tests, we did not allow paths from the test set to be fed back.

We will now compare the results from the two anytime approaches. Figure 5a shows the average sub-optimality bound (across the test goals) for the planners across the two minute planning time. Using a newly computed shortcut for each new iteration of the search (labeled as "many shortcuts") improves both anytime methods. Also, recomputing the heuristic for each iteration (H_1) performs better than interpolating between the initial and final heuristic (H_2). The better guidance provided to the search more than makes up for the overhead to recompute the heuristic for each iteration. Figure 5b shows the same plot but for the average cost of the solution instead of the bound. We can see that there is a reasonable correlation between the improvement of the bound and the actual reduction of the cost. In both plots, the time starts at 5 seconds so that most trials actually have a first solution.

For the comparisons against ARA* and RRT*, we used the anytime E-Graphs with H_1 and new shortcuts for each iteration, since this version performed the best in the previous comparison. Table I shows the planning times for E-Graphs. On average, 52% of the edges on the first path produced by the planner were recycled from G^E. After 2 minutes of improvement, on average, only 21% of the edges on the final path were reused from G^E. In Table II we can see that

TABLE I: Anytime E-Graph First Solution Planning Time

successes(of 40)	mean time(s)	std dev(s)	max(s)
40	2.12	6.92	43.90

E-Graphs provide a significant speed-up over all the other methods in finding a first solution, generally over 10 times. RRT* also fails to solve many of the queries within the 2 minute limit.

Table III shows how the path quality of the first E-Graph solution compares to other methods. Similarly, Table IV shows how the E-Graph final solution compares to other methods. By looking at either table it can be seen that both ARA* and RRT* improve from their first to final solution. However, the first solution returned by E-Graphs is already better than both the first and final solution produced by RRT* and this ratio only gets larger when comparing with the final E-Graph path. As expected, the ARA* path quality is better than the E-Graph first solution since the algorithm may take a non-direct route in order to reuse prior experience. Interestingly, after being given 2 minutes to plan, E-Graphs performs as well as ARA*. This demonstrates that the artifacts of reusing parts of prior paths drops away as the planner is given more time to plan.

B. Real Robot Experiments

A series of tests were also run on the PR2 robot (results from these tests can also be seen in the video accompanying this paper). To test the anytime properties of our algorithm, we required the robot to perform a task where it has to lift a tray off of a low platform onto a high table (Figure 6a). This task requires use of the arms as well as simultaneous navigation with the base. An advantage of this kind of coupled planning (over a hierarchical approach that plans

TABLE II: First Solution Planning Time Comparison

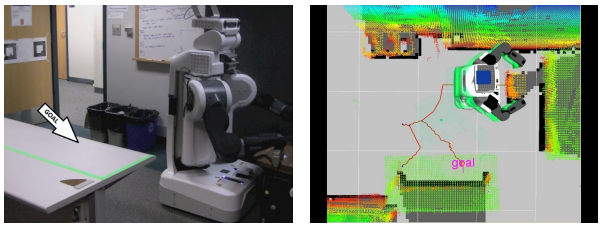
method	successes(of 40)	mean speed-up	std dev speed-up	max speed-up
ARA*	36	16.42	47.97	280.98
RRT*	25	12.36	33.90	165.33

TABLE III: Path Quality Comparison (Other Method to First E-Graph Solution Ratio)

method	object XYZ path length ratio	std dev ratio
ARA* (first solution)	0.86	0.22
ARA* (final solution)	0.82	0.25
RRT* (first solution)	1.18	0.78
RRT* (final solution)	1.11	0.51

TABLE IV: Path Quality Comparison (Other Method to Final E-Graph Solution Ratio)

method	object XYZ path length ratio	std dev ratio
ARA* (first solution)	1.07	0.28
ARA* (final solution)	1.01	0.26
RRT* (first solution)	1.46	0.93
RRT* (final solution)	1.40	0.68



(a) Experimental setup (b) Initial E-Graph

Fig. 6: Anytime E-Graph Experiment

a path for the base followed by a plan for the arms) is that it does not require explicitly planning for the positions of the base of the robot to reach the goal with its arms (the planner solves this problem automatically).

The planner generated paths to 3 goals (first solutions) to build a simple E-Graph in the environment before being given the test goal (Figure 6b). The first time that the robot was asked to find the first feasible solution for the test goal, it required a planning time of 35.98 seconds. However, this solution uses large parts of the E-Graph which in this case creates a highly sub-optimal path (Figure 7). The E-Graph was then re-initialized with paths for the 3 goals (but not the test goal) and the planner was then allowed to take 2 minutes. In Figure 8 we can see that the new planned path is significantly shorter as it cuts out most of the E-Graph paths.

Our second experiment demonstrates how E-Graphs can be used to do incremental planning. The robot's task is to bring a tray to a person. The initial planning time is 8.15 seconds (with an empty Experience Graph). Figure 9a shows this first path. After the PR2 starts executing the path, the person slides down to the left side of the table and the goal

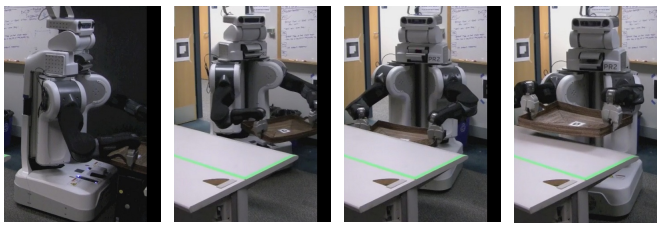


Fig. 7: The first, highly sub-optimal path

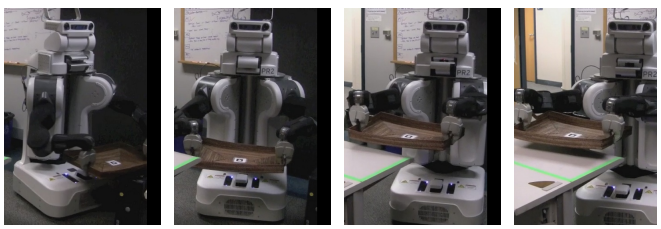
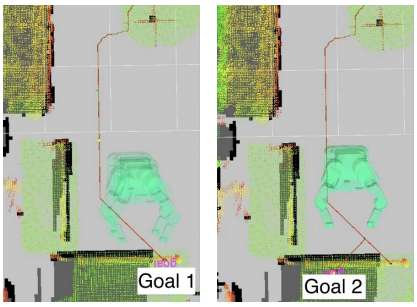


Fig. 8: The final path



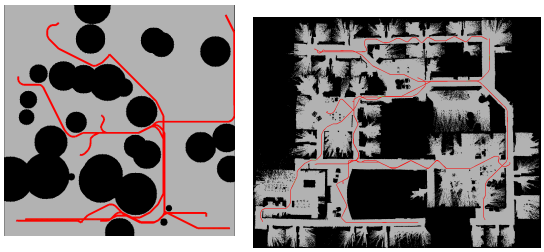
(a) Path to the first goal (b) Path to the goal after it moved

Fig. 9: Incremental E-Graph Experiment

state moves. The robot is told where the new goal is, stops, generates a new plan (Figure 9b) in 1.15 seconds and finishes the task. The second planning time is short because the first path is now in the E-Graph and most of this path can be reused (only the end of the path needs to be modified). An interesting thing to note is that both the start and goal states changed between the two planning requests. Conventional incremental planning methods (such as D* and its variants) are not able to reuse any previous information if the root of the search tree changes. If both the start and the goal change these methods must plan from scratch. On the other hand, E-Graphs allow prior computation to still be used in these cases.

C. Navigation Simulations

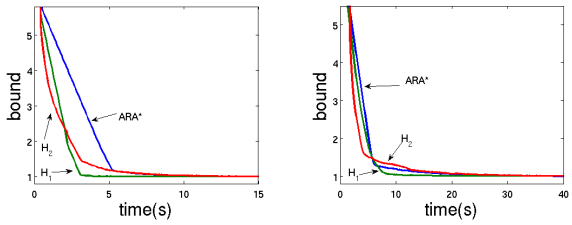
While E-Graphs are particularly well suited for high dimensional problems like full-body planning for the PR2, finding *optimal* solutions in such spaces is infeasible. Therefore, in order to get a better idea of the anytime profile of our algorithm as it approaches optimality, we ran experiments in an easier (x, y, θ) navigation domain. The θ dimension (heading) is useful in navigation planning for creating smooth paths especially for robots with non-holonomic constraints or non-circular footprints. We ran experiments on two maps: an indoor map of a real building (Figure 10b), and a randomly generated "outdoor" map (Figure 10a) with sparse obstacles and large areas of free space. The outdoor map is 500 by 500 cells while the indoor map is 2332 by 1825 cells. The heading dimension is discretized into 16 directions.



(a) E-Graph for an outdoor map (b) E-Graph for an indoor map

Fig. 10: Maps (x, y, θ) navigation experiments

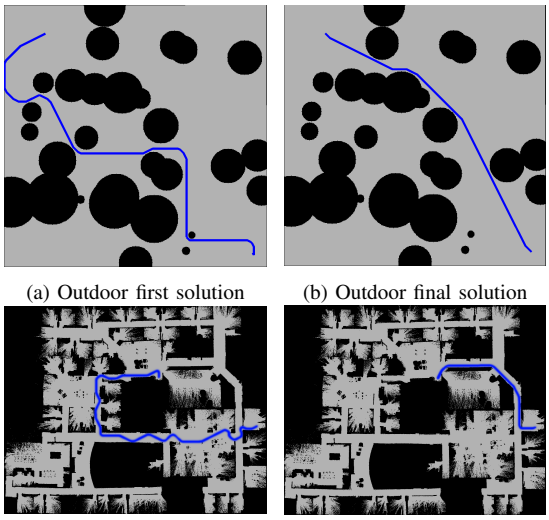
Our two methods were bootstrapped with 10 planning queries to build up an E-Graph with good coverage (Fig-



(a) Average bound over time on an outdoor map

(b) Average bound over time on an indoor map

Fig. 11: Anytime profiles for (x, y, θ) navigation



(a) Outdoor first solution

(b) Outdoor final solution

(c) Indoor first solution

(d) Indoor final solution

Fig. 12: Examples of anytime planning with the H_2 heuristic

ure 10). Then all methods were given the same 50 randomly generated test queries (E-Graph methods were not allowed to feed back the solutions to these queries so that they would have the same E-Graph throughout the tests). We compared our two methods against ARA*. The heuristic h^G is a 2D breadth first search starting from the goal (this was also the heuristic used for ARA*). All methods planned to a first solution with a bound of 6 and were given two minutes total to plan, though optimality was generally reached much earlier than that. Figure 11 shows the average bound (across the 50 trials) reached over time. The optimal solution is generally found in the first 10 seconds for the outdoor map and within 30 seconds for the indoor map. We can see that H_2 has on par performance with H_1 even though it only computes the first E-Graph heuristic. The anytime profiles of our methods are actually slightly better than ARA* especially on the outdoor map. Examples of a first and final solution path for method H_2 on each map are shown in $Figure$ 12.

VIII. Conclusion

In this paper we have presented an anytime extension to planning with Experience Graphs, a general search-based planning method for reusing parts of previous paths in order to speed up future planning requests. Our approach is able to do this while still providing theoretical guarantees on completeness and a bound on the solution quality with respect to the graph representing the planning problem. We demonstrated how E-Graphs provide a new way to approach the incremental planning problem. Unlike conventional incremental planners, E-Graphs can reuse previous computations even when both the start and goal configurations change. We provided experiments on full-body mobile manipulation tasks both in simulation and on a real PR2 which demonstrate the effectiveness of using E-Graphs in anytime and incremental fashion. Our planner is able to find initial solutions significantly faster than other anytime planners and provides solution quality that is on par or better.

References

[1] M. Phillips, B. Cohen, S. Chitta, and M. Likhachev, "E-Graphs: Bootstrapping Planning with Experience Graphs," in *Proceedings of the Robotics, Science and Systems Conference*, 2012.

[2] J. Lien and Y. Lu, "Planning motion in environments with similar obstacles," in *Proceedings of the Robotics, Science and Systems Conference*, 2005.

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

[4] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite rrts for rapid replanning in dynamic environments," in *IEEE International Conference on Robotics and Automation*, 2007.

[5] D. Ferguson, N. Kalra, and A. T. Stenz, "Replanning with rrts," in *IEEE International Conference on Robotics and Automation*, May 2006.

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

[7] C. Atkeson and J. Morimoto, "Nonparametric representation of policies and value functions: A trajectory-based approach," in *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[8] C. Liu and C. G. Atkeson, "Standing balance control using a trajectory library," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[9] M. Stolle, H. Tappeiner, J. Chestnutt, and C. Atkeson, "Transfer of policies based on trajectory libraries," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

[10] N. Jetchev and M. Toussaint, "Trajectory prediction: Learning to map situations to robot trajectories," in *IEEE International Conference on Robotics and Automation*, 2010.

[11] X. Jiang and M. Kallmann, "Learning humanoid reaching tasks in dynamic environments," in *IEEE International Conference on Intelligent Robots and Systems*, 2007.

[12] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *ICRA*, 2012.

[13] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems (NIPS) 16*. Cambridge, MA: MIT Press, 2003.

[14] M. Likhachev, D. Ferguson, G. Gordon, A. Stenz, and S. Thrun, "Anytime dynamic A*: An anytime replanning algorithm," in *International Conference on Automated Planning and Scheduling*. AAAI, 2005.

[15] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

[16] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[17] R. A. Knepper and A. Kelly, "High performance state lattice planning using heuristic look-up tables." in *IROS*. IEEE, 2006, pp. 3375–3380.

[18] A. T. Stentz, "The focussed d* algorithm for real-time replanning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, August 1995.

[19] S. Koenig and M. Likhachev, "D* lite," in *AAAI*, 2002, pp. 476–483.

[20] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, 2012, to appear. [Online]. Available: http://ompl.kavrakilab.org