Efficient Manipulation Task Planning via Reuse-Informed Optimization of Planning Effort

Christopher M. Dellin

April 7, 2015

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Siddhartha Srinivasa, CMU RI (Chair)
Anthony Stentz, CMU RI
Maxim Likhachev, CMU RI
Lydia Kavraki, Rice University

Abstract

In order to assist humans with dangerous or menial tasks, autonomous robots will need to act under significant time and energy constraints. At task time, the amount of effort a robot spends planning directly detracts from its total performance. Manipulation tasks, however, present challenges to efficient motion planning. They are often tightly coupled – while moving an object can be decomposed into steps (reach, grasp, transfer, release), each step requires choices (e.g. which grasp), and committing to a bad choice can render subsequent steps difficult; this encourages longer planning horizons. However, an articulated robot situated within a geometrically complex and dynamic environment induces a high-dimensional configuration space in which it is expensive to test for valid paths. And since multi-step plans require paths in changing valid subsets of configuration space, it is difficult to reuse computation across steps or maintain caches between tasks.

We focus on a motion planning approach for coupled multi-step manipulation problems that is efficient over the entire task (including both planning and execution). We contend that the problem's cost structure favors explicit handling of both graph representation and task effort optimization, and propose a graph search algorithm which captures these insights given a model of planning effort. We offer methods for roadmap construction which seek to balance completeness with efficiency at task time. We then unify previous work examining configuration space structure of related problems (e.g. multi-step manipulation) into a general set-theoretic formulation which suggests a planning effort model to be exploited by our roadmap search algorithm, yielding a motion planner which efficiently reuses computation between queries. We also present a task planner that maps a task decomposition into queries to our motion planner. Our insights yield complementary components which, taken together, constitute an efficient approach to planning manipulation tasks.

This thesis proposes a heavy emphasis on experimental evaluation of the individual constituent algorithms and the approach as a whole. We will compare against state-of-the-art task and motion planners on multiple robotic platforms, in applications from home table clearing to remote disaster response. We also provide open-source implementations of our algorithms.

Contents

1
Introduction

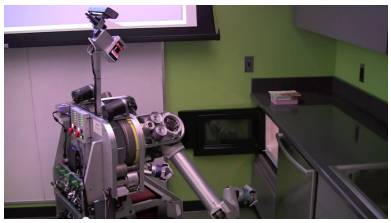
The steady advancement of technology has automated an increasing variety of menial or dangerous tasks previously performed by humans. Computer algorithms now trade our stocks, route our telephone calls and packages, and fly our planes, while simple machines clean our clothes and wash our dishes.

More complex tasks require complex robots with many degrees of freedom. Manipulation tasks, in particular, present challenges in many areas including perception, symbolic reasoning, and motion planning. Successful applications have so far been largely confined to manufacturing domains whose prescribed and structured environments allow these challenges to be overcome.

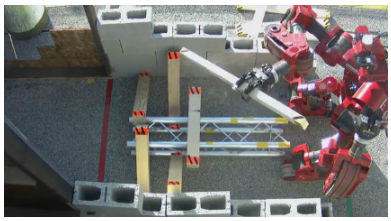
In contrast, consider the manipulation tasks in Figure 1.1. In the first application, the HERB home robot must retrieve a frozen meal from a countertop and transfer it into a microwave oven. In the second, the CHIMP distaster response robot must clear large pieces of debris from a blocked doorway in the recent DARPA Robotics Challenge competition. These multi-step manipulation tasks require finding motion plans in high-dimensional, dynamic, semi-structured spaces under significant resource constraints (e.g. time or energy). State-of-the-art approaches can not yet handle these planning problems quickly and reliably.

> *This thesis proposes an efficient motion planning approach well-suited to articulated robots performing recurring multi-step manipulation tasks in dynamic, semi-structured environments.*

To address this problem in depth, this thesis focuses on efficient kinematic planning approaches to coupled manipulation tasks consisting of several (2-5) steps as formuated in Chapter 2. We don't consider uncertainty or feedback to symbolic planners, and we focus primarily on sequential planning and execution. We consider our approach complementary to hybrid and interleaved planners, as well as approaches which learn heuristics.



(a) The HERB assistance robot moves a frozen meal into a microwave oven.



(b) The CHIMP distaster response robot clears debris from a blocked doorway.

Figure 1.1: This thesis addresses multi-step motion planning for manipulation tasks.

Challenges	Insights
Capturing Planning/Execution Tradeoff	Minimize Ensemble Effort on Explicit Graphs
Incongruent Sub-Problems Impede Reuse	Identify and Exploit Multi-Set Structure
Coupled Steps Require Long-Horizon Plans	Comprehensive Sub-Problem Planning

Table 1.1: Challenges and Insights

Challenges to Efficient Manipulation Planning

There are three principal challenges inherent in such human-scale manipulation tasks (Table 1.1) which render the planning problem difficult. We survey them here.

Challenge 1: Capturing the Planning vs. Execution Tradeoff

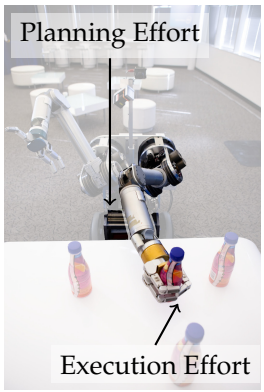
Manipulation robots must be resource-efficient. If a home robot takes thirty minutes to clear a table, or a disaster response robot exhausts its battery ten minutes into its mission, these robots will not see widespread use. A robot expends two types of effort. First, it must allocate computation to *plan* a sequence of motions that will acheive the task. Second, it must *execute* these motions using its actuators. Typically, there is a tradeoff between these two; spending more effort planning produces solutions that are cheaper to execute.

Consider the spectrum of planning domains in Figure 1.2. In many types of problems, such as planetary exploration, execution resources are most scarce, and therefore the optimality of the solution is most imporant. On the other hand, in domains such as automated theorem proving, the quality of the solution is less important; planners are instead designed to find a feasible solution quickly.

In manipulation tasks, both types of efficiency are equally important; metrics such as time or energy use are only meaningful when applied across the entire task (from assignment to completion). In

Figure 1.2: In many planning domains, either planning effort or execution effort is most important to minimize. For manipulation tasks, the HERB and CHIMP incur comparable cost (e.g. time or energy) during planning and execution. It is therefore important for planning approaches to accurately model and reason about this tradeoff.

(a) Planetary exploration: cost = execution effort



(b) Manipulation tasks on the HERB and and CHIMP robots: cost = planning effort + execution effort

Axioms:
- Axiom 1: $f(1,x) = x$
- Axiom 2: $u(x,y) = u(y,x)$
- Axiom 3: $u(1,a) = a$

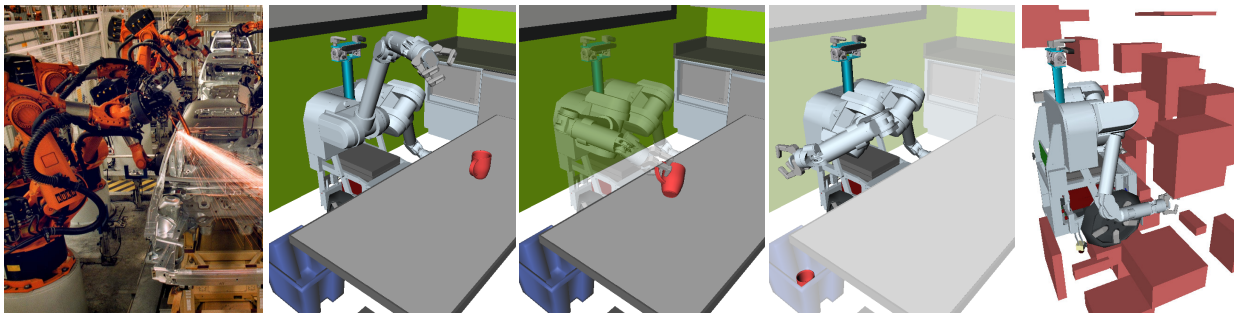
Goal Theorem:
- $u(a, f(a,b) = f(a,b)$

Proof:

Lemma 1:
- $u(z, f(a,z)) = f(a,z)$
$= u(f(a,z),z)$ by Axiom 2
$= u(f(a,z), f(1,z))$ by Axiom 1

Theorem 1:
...

(c) Automated theorem proving: cost = planning effort



(a) Fixed worlds (multi-query)

(b) A multi-step manipulation task.

(c) Random worlds (single-query)

Figure 1.3: The structure of multi-step manipulation tasks requires multiple queries, each in different but related subsets of configuration space.

fact, measured in either time or energy, robots tend to expend comparable effort on each. For example, at the DRC Trials, the CHIMP robot spent an average of 140.6 s planning, and 373.6 s executing those plans (with very slow execution speeds for safety). Therefore, motion planners for manipulation tasks must be able to reason about the important tradeoff between these two types of effort.

Challenge 2: Incongruent Sub-Problems Impede Reuse

Because planning effort is such an important factor in manipulation task efficiency, it's important to consider approaches which minimize computation across multiple planning queries. This is especially important in high-dimensional spaces, where planning costs are dominated by *validity checking* – e.g. checking whether configurations are free from collision. However, the structure of manipulation problems makes it difficult to apply common approaches.

Consider the motion planning problems in Figure 1.3. In the first case, an industrial robot repeatedly addresses the same scene; *multi-query* approaches such as the Probabalistic RoadMap[1] have proven effective at facilitating reuse in such cases. However, autonomous manipulation robots perform tasks in dynamic, semi-structured environments. Further, during each step of a manipulation plan, the valid subset of configuration space changes as objects are grasped and moved, making it difficult to apply roadmap approaches.

At the other extreme, a robot addressing a set of different random worlds are best served by *single-query* approaches which build a graph structure "from scratch" for each problem they solve. Applying such approaches to multi-step manipulation tasks leads to inefficiency in planning as computation is not reused between queries.

Manipulation sub-problems do have structure. Since they occupy this middle ground between fixed and random environments, it's essential that planning approaches find a way to reuse computation effectively between inconguent sub-problems.

[1] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, Aug. 1996

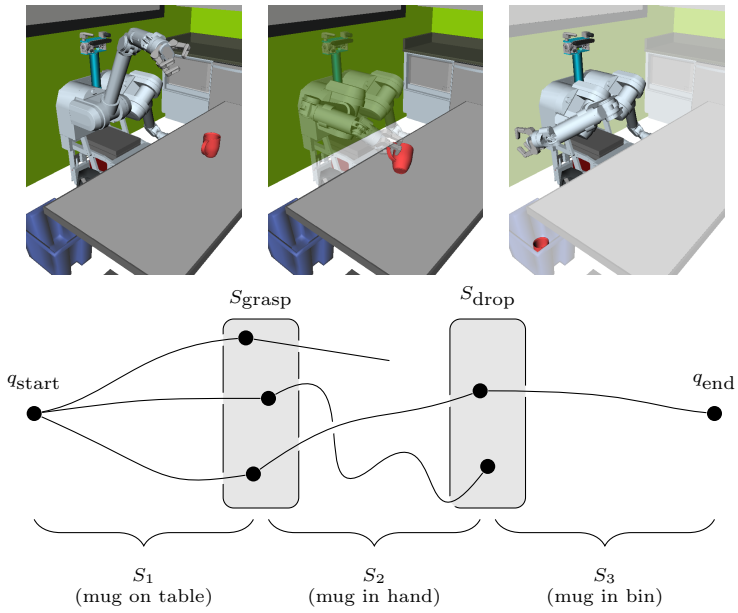


Figure 1.4: Manipulation sub-problems within incongruous free subsets of configuration space. HERB plans for a simple manipulation task to grasp, transfer, and drop a mug from a table into a bin before returning to an end configuration. Each sub-problem requires a path in a distinct free subset of configuration space.

Challenge 3: Coupled Steps Require Long-Horizon Plans

Not only does the structure of manipulation tasks impede planner reuse between sub-problems, but it also necessitates long-horizon plans to ensure robustness. For example, consider sequentially planning for the task in Figure 1.3. The interfaces between these three steps lie on continuous manifolds. A choice made by an early planning step – e.g. what arm configuration or object grasp to use – often renders a subsequent part either difficult to plan, costly to execute, or impossible altogether. We say that such sub-problems are *coupled*.

We therefore might endeavor to plan for all steps simultaneously before execution. Indeed, many motion planners are designed to take as input start and goal sets. We might hope that we can delegate each task sub-problem to separate such planner instances, and provide each with specifications for their corresponding root sets. However, a customary planning request takes an *any-to-any* form (i.e. from *any* start to *any* goal configuration). Clearly, without coordination, the juxtaposed solution paths will not be continuous.

Key Insights to Our Approach

While these three challenges make efficient manipulation task planning a difficult problem, we identify three corresponding and complementary key insights to address them. We apply our insights to graph- and roadmap-based planners as summarized in Table 1.2.

Ch.	Algorithm	Problem	Representation	Opt-Plan	Multi-Set
	A* Search [9]	Graph	Implicit	No	No
	Weighted A* Search	Graph	Implicit	Partial	No
	Experience Graphs [25]	Graph	Implicit	Partial	Partial
	BUGSY [27]	Graph	Implicit	Yes	No
2.1	**E^8 Search**	**Graph**	**Explicit**	**Yes**	**No**
	Lazy PRM [2]	$\mathcal{C}$-space	Roadmap (Explicit)	No	No
	Dynamic Planner [12]	$\mathcal{C}$-space	Roadmap (Explicit)	No	Partial
2.2	**E^8-PRM**	**$\mathcal{C}$-space**	**Roadmap (Explicit)**	**Yes**	**No**
2.4	**Multi-Set PRM**	**$\mathcal{C}$-space Families**	**Roadmap (Explicit)**	**Yes**	**Yes**
2.6	**Proteus**	**Multi-Step Tasks**	**Multi-Graph (Explicit)**	**Yes**	**Yes**

Table 1.2: Comparison of algorithms. The "Opt-Plan" column denotes planners which explicitly optimize an objective which includes a planning effort term. The "Multi-Set" column denotes planners which reason about common structure between related problems.

Insight 1: Minimize Ensemble Effort on Explicit Graphs

Because the tradeoff between planning and execution effort is so important for manipulation tasks, it is important to choose problem representations and design algorithms which accurately model and exploit this structure. We make two contributions to this end as described in Chapter 2.1.

First, while graph-based approaches have proven effective at solving many problems in high-dimensional spaces, we propose that the planning effort model assumed by traditional search algorithms (e.g. A*) may not be well-suited to motion planning problems for articulated robots. In particular, we propose that *explicit* graph representations allow the planner to more accurately capture the significant edge evaluation costs inherent in such problems.

Second, our planning approach explicitly optimizes for both planning and execution effort – what we call the task's *ensemble effort*. We submit that our approach handles this tradeoff more directly than "anytime" planning approaches. We apply this reasoning to explicit graphs with the E^8 search algorithm which determines an effort allocation between planning and execution in order to minimize total task cost.

In order to solve planning problems in continous configuration spaces, we borrow heavily from roadmap techniques for graph construction. The application of our ensemble effort algorithm to such roadmaps, the E^8-PRM (Chapter 2.2), maintains efficiency through an incremental densification approach motivated by approximating the probabalistic spatial correlation of $\mathcal{C}_{\text{free}}$. We propose to show that this planner can outperform state-of-the-art single-query motion planners for several manipulation tasks.

Insight 2: Identifying and Exploiting Multi-Set Structure

In order to address the important issue of planning computation reuse between the incongruent sub-problems inherent in manipulation tasks, we propose the *multi-set* planning formulation in Chapter 2.3 which bridges the gap between single-query and multi-query approaches. We explicitly represent the structure of such problems through a family of subsets of the robot's configuration space, as well as set relations between them. We further demonstrate instances of multi-set structure in many classes of manipulation problems, and use the formulation to unify several instances of prior work aimed at efficiently planning for such tasks.

Next, Chapter 2.4 shows how the multi-set formulation can be used as a planning effort model when solving planning queries in related subsets of configuration space. This planning model can then be inserted directly into the E^8-PRM to handle multi-step problems. The resulting algorithm, the Multi-Set PRM, uses propositional logic to represent the multi-set structure algorithmically.

Insight 3: Comprehensive Sub-Problem Planning

Our last insight addresses how to efficiently decompose a coupled multi-step manipulation task into individual sub-problems. For tightly coupled problems, it is especially important to plan for multiple possibilities. First, we introduce the *Comprehensive Multi-Root* (CMR) planner objective (Chapter 2.5). In constrast to the traditional any-to-any objective, CMR encourages a planner to discover paths between multiple pairs of roots. We also show that this objective can be applied greedily to traditional PRM planners yielding provably superior algorithms for CMR problems.

Second, we introduce the PROTEUS task planner (Chapter 2.6) which combines the three insights into a single planning approach. The task planner automatically discovers the multi-set structure in each task and instantiates a Multi-Set PRM to handle planning queries for each sub-problem. It then performs root sampling within each interface manifold and uses the CMR objective for each query to efficiently search for a full task plan.

Evaluation

Because this thesis presents an integrated set of algorithms for multi-step manipulation task planning, we endeavor to place significant emphasis on empirical evaluation in order to verify the performance of the proposed approach. See research question Q8 for more details about the metrics, baseline approaches, and test platforms that will

be used in these comparisons.

I refer the reader to Chapter 3 for a summary of the particular research questions that I propose to answer. Chief among these questions are the effects of the insights above on task performance as borne out by these experiments:

- The effect of the E^8 algorithm's planning vs. execution effort trade-off parameter (λ) compared to weighted/anytime approaches.
- The effect of the E^8-PRM's batching strategy on its efficacy as a single-query planner in high-dimensional planning problems.
- The performance of the Multi-Set PRM on problems with various amounts of shared structure (from fully fixed to fully random environments).
- The performance of the PROTEUS task planner compared to other baseline planners that address multi-step tasks.

Summary of Contributions

In summary, I propose the following contributions:

- A set of insights into the structure of the multi-step manipulation planning problem, including the multi-set and comprehensive multi-root (CMR) formulations.
- The E^8 graph search algorithm which explicitly minimizes both planning and execution effort over explicit graphs.
- The E^8-PRM, an application of E^8 to $\mathcal{C}$-space roadmaps.
- The Multi-Set PRM, an instantiation of the E^8-PRM using the multi-set formulation as a planning effort model.
- The PROTEUS task planner, a sequencing planner which submits queries across steps to an instance of the Multi-Set PRM using the CMR objective.
- The `ompl_multiset` package for the OMPL[34] sampling-based planning framework containing implementations of the E^8-PRM and Multi-Set PRM algorithms.
- The `or_proteus` package for the OpenRAVE[6] simulation framework which implements the PROTEUS task planner.
- An experimental evaluation of the above algorithms against state-of-the-art baseline approaches.

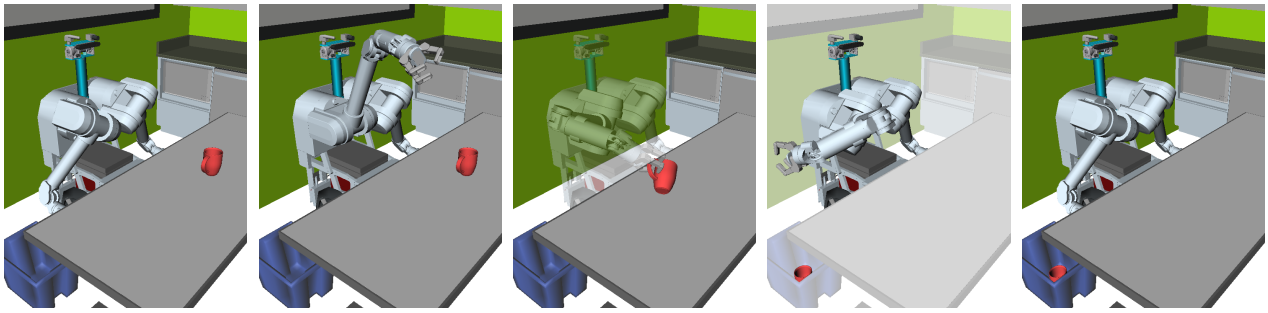
2
Framework for Efficient Manipulation Task Planning

This chapter lays out our proposed approach to planning for coupled manipulation tasks. The reader is directed to Figure 2.1 for an overview. Our approach consists generally of decomposing the task into multiple steps, each of which is solved by queries to an underlying motion planner in the robot's *configuration space*[1].

We suggest that a reasonable decomposition of a task into steps should be guided by changes to either the valid subset of the robot's configuration space (e.g. at grasp and release points of manipulated objects) or the set of constraints active on the robot's configuration. This facilitates simpler calls to each motion planner, and also allows separate planners with different capabilities (e.g. constrained planners) to be used for each step.

[1] T. Lozano-Pérez. Spatial planning: A configuration space approach. *Computers, IEEE Transactions on*, C-32(2):108–120, Feb. 1983

Outline. Loosely, the sections in this chapter are layed out as follows. Sections 2.1 and 2.2 describe our approach to planning *within* each step, on graphs and roadmaps respectively. Next, Sections 2.3 and 2.4 identify and exploit structure *between* steps in order to reuse planning computation across queries. Lastly, Sections 2.5 and 2.6 discuss how to specify queries across all steps in a higher-level sequencing task planner.



(a) Start config (b) Step 1 in S_1 (c) Step 2 in S_2 (d) Step 3 in S_3 (e) End config

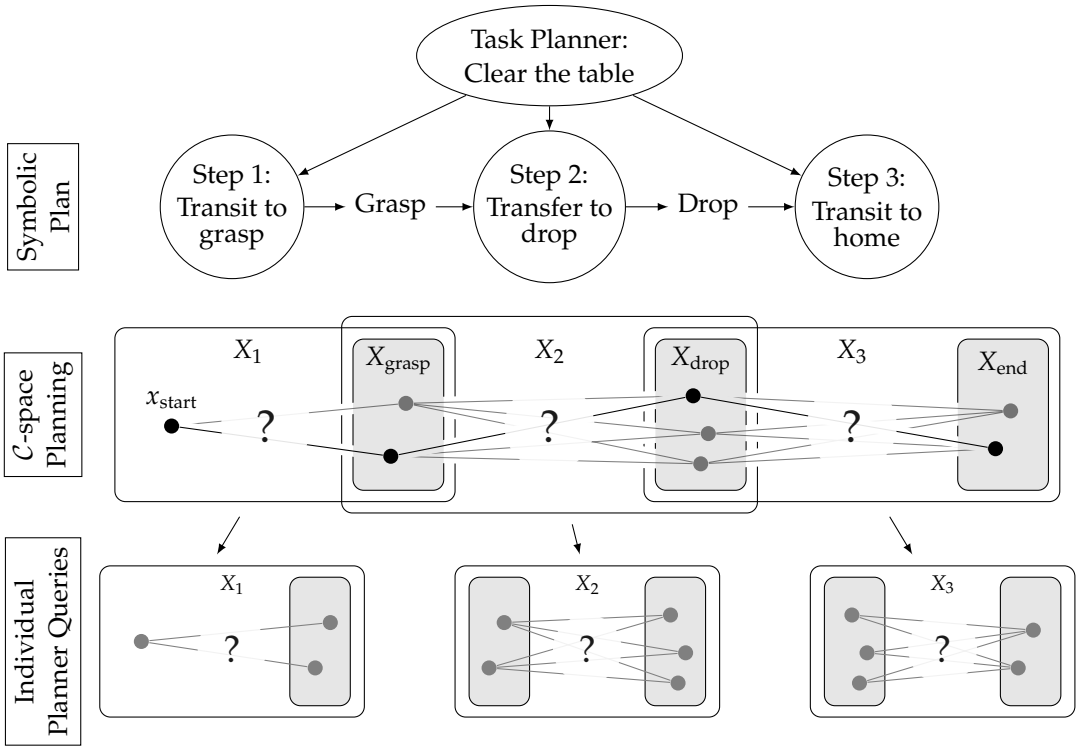


Figure 2.1: Diagram of multi-step planning framework. This thesis focuses on efficient geometric planning for manipulation tasks (the lower levels here). Task planning can be performed by an autonomous symbolic planner, or guided by a human operator.

2.1 *Quickly Searching Explicit Expensive Graphs*

In order for a robot to perform manipulation tasks in changing, unstructured environments, it must be able to quickly solve planning queries. In this section, we discuss formulating the motion planning problem as best-first graph search over paths and propose an algorithm on explicit graphs. Our approach is motivated by two insights.

Explicit Optimization of Ensemble Effort. This section references two different types of efficiency with regard to robotic tasks. First, once a planner has computed a solution path or trajectory, there is the cost incurred while executing that trajectory. This is the traditional cost optimized for by planners. Second, there is the cost incurred from actually computing the solution itself. We propose to optimize for both types of effort explicitly.

Explicit Graph Representation. We contend that representing graphs explicitly is better suited to search over roadmap graphs for two reasons. First, it is reasonable to store the entire graph *explicitly* in memory (e.g. 10k vertices for a 7-DOF arm); techniques to incrementally build the graph via an implicit graph representation are not necessary. Second, it is evaluating edge costs (as opposed to expanding vertices or maintaining a sorted open list) that dominates planning costs. In other words, precious planning effort is principally manifested in *evaluated edge costs*, not *determined vertex g-values*.

2.1.1 *Generic Best-First Search Algorithm over Paths*

Best-first search[2] is a general class of search algorithms. We choose to express the general algorith over *paths* instead of *vertices* for clarity and generality because we are focused primarily on explicit graphs.

[2] P. H. Winston. *Artificial Intelligene*. Addison-Wesley Pub. Co., 1977

Algorithm 1 Generic Best-First Search Algorithm Outline

```
1: procedure GenericBestFirst(G)
2:     loop
3:         π* = arg min_{π∈Π(G)} f(π)        ▷ For some path cost function f(π)
4:         if π* fully evaluated then
5:             return π*
6:         EvalPath(π*)                      ▷ For some evaluation function
```

This formulation admits two choices:

Cost Function $f(\pi)$. What is the cost function $f(\pi)$ over paths used to select the path for evaluation at each iteration? Traditional graph

search uses the following path objective:

$$f(\pi) = \hat{f}_x(\pi) : \textit{optimistic estimate of execution effort}. \tag{2.1}$$

In other worts, $\hat{f}_x(\pi)$ gives a lower bound on the cost of executing path π given the algorithm's current knowledge of the graph. In our case, if the path consists of a mix of evaluated and unevaluated edges, we could write this as:

$$\hat{f}_x(\pi) = \sum_{e \in \pi} \begin{cases} x[e] & \text{if edge } e \text{ evaluated} \\ \hat{x}(e) & \text{otherwise} \end{cases}, \tag{2.2}$$

with $\hat{x}(e)$ an admissible estimate of the edge's execution effort.

Evaluation Procedure EvalPath(π). How is a potential path evaluated? We discuss this later in this section.

The choice of these two components of the algorithm intimately depend on the graph representation. For appropriate selection of $f(\pi)$ and EvalPath, traditional algorithms such as A* [9] and the Bidirectional Heuristic Front-to-Front Algorithm [31] are instances of this general formulation.

2.1.2 *Penalizing Planning Effort*

So far, we've been searching for a path which optimizes our execution effort objective (2.2). However, as we motivated earlier, there are two distint notions of efficency; here, we focus instead on *planning efficiency*. Consider the following path objective:

$$f(\pi) = \hat{f}_p(\pi) : \textit{optimistic estimate of planning effort}. \tag{2.3}$$

For problems over large graphs, planning effort may be dominated by discovering vertex successors or maintaining a sorted vertex open list. However, in many manipulation problems, planning effort is instead dominated by *edge evaluations*. Therefore, our objective $\hat{f}_p$ penalizes remaining effort required to evaluate edges along a path:

Traditional metrics for planning effort include *vertiex expansions* and *heap percolates*.

$$\hat{f}_p(\pi) = \sum_{e \in \pi} \begin{cases} 0 & \text{if edge } e \text{ evaluated} \\ \hat{p}(e) & \text{otherwise} \end{cases}. \tag{2.4}$$

The new edge heuristic $\hat{p}(e)$ estimates this evaluation cost.

Suggested metrics for $\hat{p}$ include planning time or computational energy.

The first graph planner to explicitly include such a heuristic to estimate the remaining computational planning effort in a best-first search was A^*_ϵ [24]. While the approach we take is different, a motivating quote from this paper is relevant:

> "The heuristic [$\hat{x}$] ... is of an entirely different nature than the ... heuristic [$\hat{p}$] ... . The former anticipates the reduction in *solution quality* due to the remaining part of the solution once it is found; the latter estimates the *computational effort* required for completing the search."

2.1.3 Ensemble Effort Objective

In general, we might consider weighting each objective:

$$f(\pi) = \lambda \hat{f}_p(\pi) + (1-\lambda)\hat{f}_x(\pi). \tag{2.5}$$

We call this effort model *ensemble effort* in that it combines both planning and execution effort. Note that with $\lambda = 0$, we recover our old solution cost objective $\hat{f}_x(\pi)$. Represented over edges, we can write:

$$f(\pi) = \sum_{e \in \pi} \begin{cases} (1-\lambda)x[e] & \text{if edge } e \text{ evaluated} \\ \lambda\hat{p}(e) + (1-\lambda)\hat{x}(e) & \text{otherwise} \end{cases}. \tag{2.6}$$

We represent a particular choice of $\hat{p}(e)$ and $\hat{x}(e)$, along with the evaluation function $x(e)$, as an *ensemble effort model* denoted with $\mathcal{M}$:

For examples of ensemble effort models, see Section 2.2.2.

$$\mathcal{M} : (x, \hat{x}, \hat{p}) \tag{2.7}$$

Simplification with Propotional Heuristics. Suppose that our heuristic for planning effort were proportional to that for execution effort,

$$\hat{p}(e) = \alpha\, \hat{x}(e). \tag{2.8}$$

In this case, we can write:

This might happen if, for example, each were proportional to the edge's *distance* (with longer paths taking longer to both collision check and execute at constant velocity).

$$f(\pi) = (1-\lambda) \sum_{e \in \pi} \begin{cases} x[e] & \text{if edge } e \text{ evaluated} \\ \left[1 + \frac{\alpha\lambda}{1-\lambda}\right] \hat{x}(e) & \text{otherwise} \end{cases}. \tag{2.9}$$

Further, if we use an EvalPath() function which forward-evaluates vertices or edges (as is required with implicit graph representations), and $\lambda < 1$, we can rewrite (2.9) as:

$$f(\pi) \propto \underbrace{\sum_{e \text{ evaled}} x[e]}_{g[v_f]} + \underbrace{\left[1 + \frac{\alpha\lambda}{1-\lambda}\right]}_{\text{inflation factor } \epsilon} \underbrace{\hat{x}(e_{last})}_{h(v_f)}. \tag{2.10}$$

In other words, *weighted A* is equivalent to best-first search whose objective includes a planning effort term proportional to execution effort.* In particular, if planning effort is proportional to execution effort by a factor of α, a weighted A* search with inflation factor ϵ is the result of best-first search with $\lambda = \frac{\epsilon-1}{\alpha+\epsilon-1}$.

2.1.4 The E^8 Explicit Graph Search Algorithm

The E^8 algorithm (Exploiting Ensemble Effort Estimates on Explicit graphs with Expensive Edge Evaluations) is the result of applying best-first search with the λ-mediated objective from (2.5). The algorithm is *lazy*, in that edge evaluations are deferred until they are

Algorithm 2 E^8 Explicit Graph Search

```
1:  procedure E^8(G, V_start, V_goal, M, λ)
2:      x_eval[·] ← empty map (x_eval : E → R_0^+)
3:      for all e ∈ G do
4:          e.cost ← λ p̂(e) + (1 − λ) x̂(e)   ▷ Ensemble effort model M
5:      loop
6:          π* = BiDijkstras(G, V_start, V_goal)
7:          if e ∈ x_eval ∀ e ∈ π* then
8:              return π*
9:          E_to_eval ← PathEvalOrder(π*)              ▷ See Section 2.1.4
10:         for all e ∈ E_to_eval do
11:             x_eval[e] ← x(e)                  ▷ Evaluate edge (expensive!)
12:             e.cost ← (1 − λ) x_eval[e]        ▷ Update ensemble estimate
13:             if x_eval[e] > x̂(e) then
14:                 break
```

needed. In fact, it can be seen as a generalization of the LazyPRM [2], but which also considers planning effort in its objective. Further, the algorithm is *heuristic-focused*, guided by its cost model $\mathcal{M}$. Its behavior mimics that of an inflated heuristic planner depending on the selection of the planning/execution cost tradeoff parameter λ.

The E^8 algorithm (Algorithm 2) directly follows the outline of best-first search over paths (Algorithm 1). Since edge evaluations are expensive, we maintain a map $x_{\text{eval}}[\cdot]$ storing the known execution costs of all edges evaluated so far (line 2). Each edge's current cost (line 4) is derived from the problem's ensemble effort model $\mathcal{M}$.

At each iteration, we optimistically select the best path π^* which minimizes the this ensemble objective. We select over all paths which connect a vertex in V_{start} to a vertex in V_{goal} (any-to-any).

For types of planner specifications besides any-to-any, see Chapter 2.5.

If this path is already fully evaluated, we finish on line 8. Note that if edge costs $x(e)$ may be infinity (e.g. to denote an infeasible edge), the algorithm will terminate with a fully evaluated path with infinite cost if no feasible path exists.

Otherwise, we evaluate the path's unevaluated edges (lines 9 to 14). We do this in a particular order, as discussed later in this section. For each edge, we evaluate its execution cost $x(e)$ (line 11) and update our effort estimate (line 12) to account for (a) the actual execution effort and (b) the fact that no additional planning effort is needed. If the execution cost of any edge of the path proves more expensive than we had anticipated (line 13), we break and select a new path.

The E^8 algorithm admits a number of choices.

Choosing λ. The choice of the λ parameter affects the algorithm's tradeoff between planning and execution effort.

See research question Q1.

Finding the Optimistic-Optimal Path. The current implementation of E^8 uses bidirectional Dijkstra's algorithm (line 6) to select the lowest-cost path through the graph at each iteration of the algorithm. However, since the cost of only a few edges are adjusted at each iteration (line 12), it appears to be well-suited to incremental graph search algorithms (e.g. [16]) to improve search efficiency.

See research question Q2.

Selecting the Edge Evaluation Order. Once a candidate path is selected, its constituent unevaluated edges are evaluted in a particular order (line 9). Our current algorithm orders the edges alternating from the ends in. With different choices (e.g. forwards or backwards), E^8 looks a lot like A*.

See research question Q5.

2.1.5 *Repeated Queries*

The E^8 algorithm is *multi-query*, in that it maintains a data structure of evaluated edge execution costs $x_{\text{eval}}[\cdot]$ which allows reuse between different planning problems over the same graph G, either sequentially or in interleaved fashion. For different queries on the same graph, edge evaluations can simply be used in subsequent searches.

When used in this fashion, E^8 behaves similarily to Experience graphs[3]. E-graphs are a type of best-first search which are designed to find paths quickly by incentivizing the planner to rely on on edges from previous successful plans. While the E-graph planner is originally expressed over implicit graphs, we can instead express it explicitly as in Algorithm 1 with the following objective:

[3] M. Phillips, B. J. Cohen, S. Chitta, and M. Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, 2012

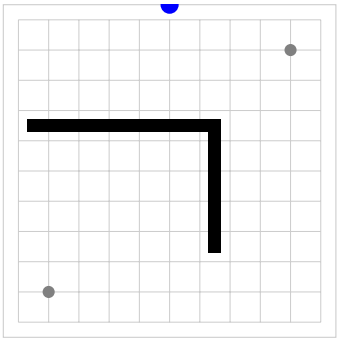
$$f_{\text{E-graphs}}(\pi) \propto \sum_{e \in \pi} \begin{cases} x[e] & \text{if edge } e \text{ evaluated, this search} \\ \epsilon\, x[e] & \text{if edge } e \text{ evaluated, previous search} \\ \epsilon\, \epsilon^E\, \hat{x}(e) & \text{otherwise} \end{cases} \tag{2.11}$$

The E^8 algorithm is therefore equivalent to a simplified version of the E-Graph algorithm with $\epsilon = 1$ and $\epsilon^E = 1 + \frac{\alpha\lambda}{1-\lambda}$, with the exception that all evaluated edges are placed in the graph, not just the edges on previous solution paths. Note that E-graph shortcuts are not necessary for small explicit graphs.

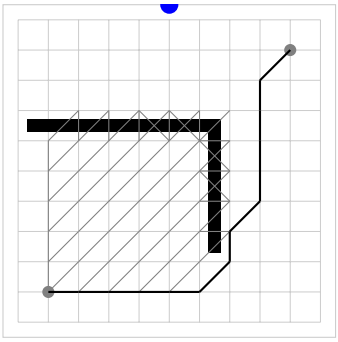
2.1.6 *Experimental Evaluation of the E^8 Algorithm*

We propose to perform experiments on graphs with expensive edge evaluations in order to evaluate the E^8 algorithm. See Figure 2.2 for an example problem.

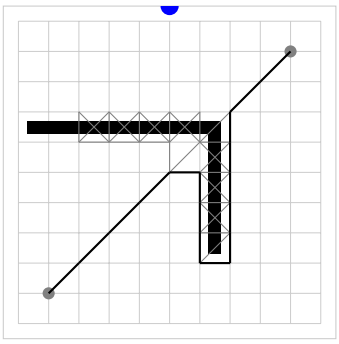
See research question Q3.



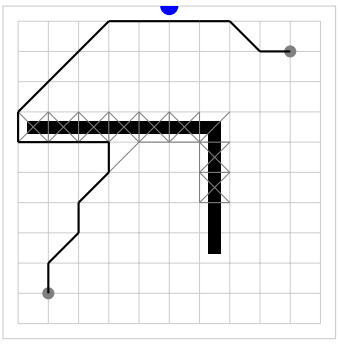
(a) Planning world.



(b) A* search.
Planning Effort: 692.3
Execution Effort: 14.2
Total Effort: 706.5



(c) Weighted A* search, $\epsilon = 3$
Planning Effort: 390.8
Execution Effort: 18.5
Total Effort: 409.3



(d) E^8 search, $\lambda = 0.5$
Planning Effort: 358.8
Execution Effort: 20.5
Total Effort: 379.3

Figure 2.2: A simple 2D explicit graph search problem, comparing three approaches. A robot (bottom left) must reach a goal position (top right) on an 8-connected graph while avoiding a wall (black). Execution effort is equal to path length. The planning effort required to check an edge is proportional to its distance to the blue radar at top. The E^8 algorithm's solution has a lower combined planning and execution effort.

2.2 *Roadmaps in Continuous Spaces*

This section explores how we can construct roadmaps covering configuration space that can then be searched by the E[8] algorithm. Here, we focus on the single-query problem; refer to Section 2.3 for extension to cached data structures. Due to this focus, we strive to compare against alternative single-query approaches as we discuss in Section 2.2.6.

Although our approach makes no commitment to the roadmap class (e.g. probabalistic or lattice-based, r-disk or k-nearest, etc.), we propose to use Random Geometric Graphs (RGGs) as roadmaps covering configuration space (see Figure 2.3), borrowing heavily from prior approaches [15] and analyses [14]. However, we are interested in trying other non-probabalistic sources of roadmap milestones (e.g. Halton sequences, Figure 2.4) as described in [19]. Note that by precomputing a sequence of samples, nearest-neighbor queries can be moved entirely offline. Also, since it can be computed offline, you can sparsify it [30].

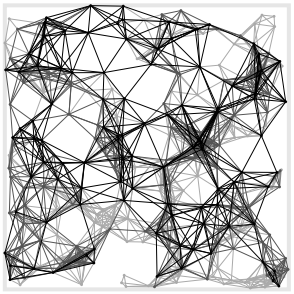


Figure 2.3: A 3-subgraph densified roadmap from pseudorandom milestones.

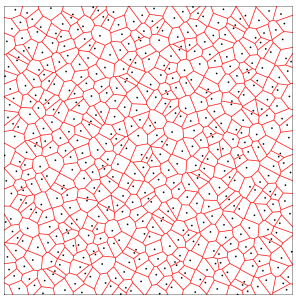


Figure 2.4: A Halton sequence (taken from Branicky, LaValle, Olson, and Yang, "Deterministic vs. Probabalistic Roadmaps."

2.2.1 *Continuous-Space Problem Formulation*

Consider a planning problem in a configuration space $\mathcal{C}$ between start and goal configurations within a collision-free subset $\mathcal{C}_{\text{free}} \subseteq \mathcal{C}$. A continuous feasible solution path $q(t)$ must then satisfy

$$\begin{aligned} q(t) &\in \mathcal{C}_{\text{free}} \; \forall \; t \in [0,1] \\ q(0) &= q_{start}, \; q(1) = q_{goal}. \end{aligned} \tag{2.12}$$

Eqns. (2.12) specify single configurations q_{start} and q_{goal}, but the formulation is trivially extended to start/goal sets.

While simple problems may admit explicit representations of $\mathcal{C}_{\text{free}}$, recent approaches to complex problems (e.g. sampling-based planners) instead reason implicitly via test function(s).[4] To capture this, we endow $\mathcal{C}_{\text{free}}$ with an *indicator function* $\mathbf{1}_{\text{free}}(q)$:

$$\mathbf{1}_{\text{free}}(q) = \begin{cases} \text{True} & \text{if } q \in \mathcal{C}_{\text{free}} \\ \text{False} & \text{otherwise.} \end{cases} \tag{2.13}$$

The set itself can equivalently be defined w.r.t. its indicator:

$$\mathcal{C}_{\text{free}} = \{q \in \mathcal{C} \mid \mathbf{1}_{\text{free}}(q) = \text{True}\}. \tag{2.14}$$

Common examples of such indicators (2.13) include validity checkers for geometric (workspace) collision, stability, and visibility constraints. Note that for complex problems, evaluation of these indicators tends to dominate planning effort.

[4] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/

Excusing the abuse of notation, we define an analogous indicator functional $\mathbf{1}_{\text{free}}[\cdot]$ which operates on path segments $q(t)$. We use parentheses for functions and brackets for functionals.

$$\mathbf{1}_{\text{free}}[q(t)] = \begin{cases} \text{True} & \text{if } q(t) \in \mathcal{C}_{\text{free}} \ \forall \ t \in [0,1] \\ \text{False} & \text{otherwise.} \end{cases} \tag{2.15}$$

A common way to approximate this functional is to call the subset's corresponding indicator function $\mathbf{1}_{\text{free}}(q)$ (e.g. a collision check) at some fixed resolution.

2.2.2 *Ensemble Effort Models*

The E^8 algorithm requires an ensemble effort model $\mathcal{M}$ over graph edges. Note that these models can be simply added element-wise.

Alg. 3 Set Validity Model $\mathcal{M}_{\text{valid}}$

function $x_{\text{valid}}(e, \mathcal{C}_{\text{free}})$
  if $\mathbf{1}_{\text{free}}[q_e(t)]$ **then**
    return 0
  else
    return ∞
function $\hat{x}_{\text{valid}}(e, \mathcal{C}_{\text{free}})$
  return 0
function $\hat{p}_{\text{valid}}(e, \mathcal{C}_{\text{free}})$
  return $\hat{p}_{\text{free}}[q_e(t)]$

Alg. 4 Distance Model $\mathcal{M}_{\text{dist}}$

function $x_{\text{dist}}(e)$
  return $||q_e(1) - q_e(0)||$
function $\hat{x}_{\text{dist}}(e)$
  return $||q_e(1) - q_e(0)||$
function $\hat{p}_{\text{dist}}(e)$
  return 0

Models of Planning Effort. Suppose we also have an estimate $\hat{p}_{\text{free}}[q(t)]$ of the effort required to evaluate the indicator functional (2.15), for example proportional to the number of collision checks required. The simplest way to capture subset membership of an edge e during the search is to map the indicator and its planning effort estimate to real-valued costs as described in $\mathcal{M}_{\text{valid}}$ (Algorithm 3). Here, invalid edges imply infinite execution effort.

Models of Execution Effort. The simplest model might simply compute Euclidean distances between vertices ($\mathcal{M}_{\text{dist}}$, Algorithm 4), although other models are possible (see Table 2.1). When posed as a traditional graph search problem, E^8 requires an additive effort models – the cost of a path is the sum of the costs of its consituent edges. However, for many problems it may be advantageous to consider non-additive models (e.g. total smoothed time), especially for execution costs for fast motions. Note that non-additive models render the path search problem more difficult.

Model	Additive?
Path Length	Yes
Bounded-Vel-Acc [10]	Yes
Smoothed	No

Table 2.1: Execution cost models. Additive models admit efficient graph search methods when choosing optimistic paths for evaluation.

2.2.3 *The E^8-PRM with Hard Batching*

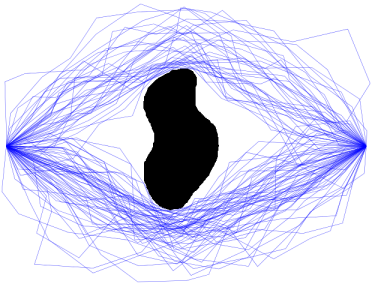
Algorithm 5 E^8-PRM Planner with Hard Batching

```
1: G ← empty graph
2: M ← M_valid(C_free) + M_exec
3: procedure E^8-PRM(G, V_start, V_goal, N, M, λ)
4:     loop
5:         PrmAddSamples(G, V_start, V_goal, N)
6:         π* ← E^8(G, V_start, V_goal, M, λ)        ▷ See Algorithm 2
7:         if x̂(π*) < ∞ then                          ▷ x̂(·) from cost model M
8:             return π*
```

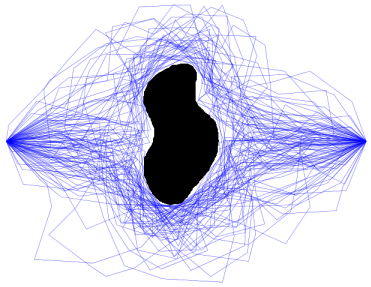
The E^8-PRM (Algorithm 5) operates on an initially empty persistent roadmap graph G. Each vertex represents a configuration $q \in \mathcal{C}$, and each edge represents a path $q(t)$ planned by a local planner.

The E^8-PRM proceeds in *batches*; at the start of each batch, the PrmAddSamples procedure adds N additional vertices to the graph sampled from $\mathcal{C}$ (including sampled start and goal configurations from $V_{\text{start}}, V_{\text{goal}}$ if not yet present), and edges are generated according to the PRM construction method.

With the exception of the path evaluation strategy (see Section 2.2.5), the E^8-PRM with $\lambda = 0$ is equivalent to the Lazy PRM [2]. Figure 2.5 shows solution paths for a simple 2D path planning problem for various runs of the algorithm.



(a) Paths with $\lambda = 0$
Average length: 733.0
Average check cost: 7219.5



(b) Paths with $\lambda = 1$
Average length: 836.5
Average check cost: 4692.6

Figure 2.5: Examples of paths for a 2D problem for different values of λ. As λ is increased, paths are longer, but are faster to find.

2.2.4 *Optimization in Expectation*

The purpose of the incremental graph densification strategy achieved by hard batching (Algorithm 5) is to account for spatial correlation in $\mathcal{C}_{\text{free}}$. We would like to motivate this more formally.

We propose to extend the E^8-PRM planner to minimize ensemble effort *in expection* using a probabalistic model of $\mathcal{C}_{\text{free}}$. Even if this is too expensive, we can motivate the incremental densification idea, with a graduated cost model to approximate a probabalistic model of the $\mathcal{C}$-space.

See research question Q4.

2.2.5 *Path Evaluation*

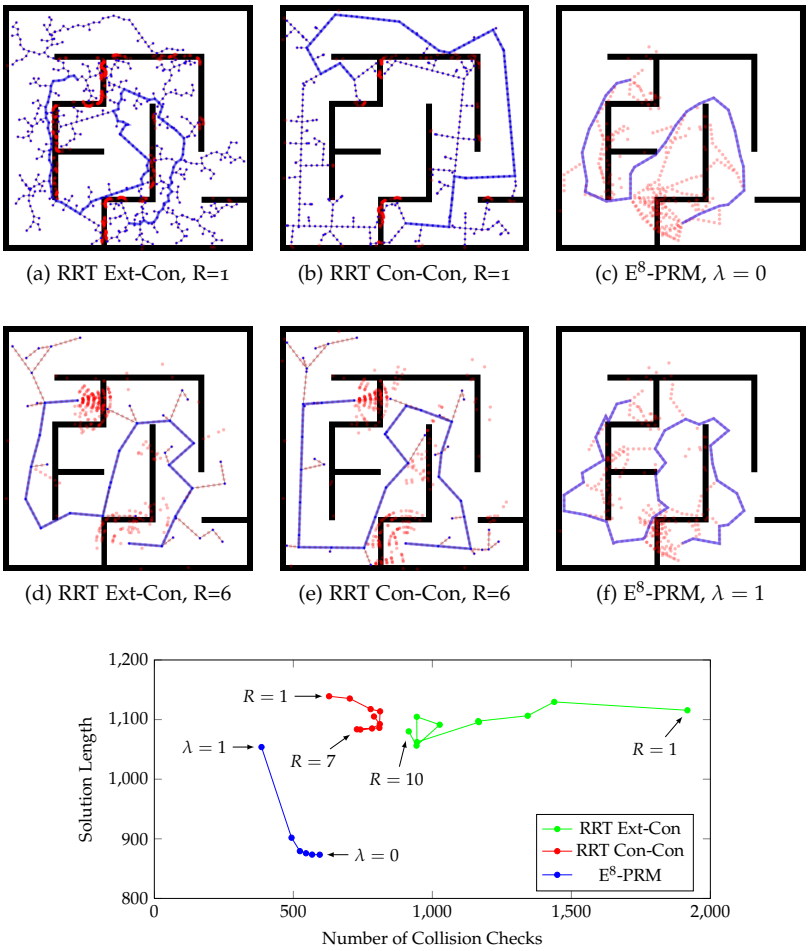
In order to use the E^8 algorithm, we must provide an edge evaluation function $x(e)$. For feasibility checks (e.g. tested via collision checkers), we approximate each edge check by evaluating the corresponding validity checker at a fixed resolution along the path.

More generally, however, since the E^8-PRM plans in a continuous space, we can actually implement more complex path tests, such as

the whole-path bisection test used in the Lazy PRM.

See research question Q5.

2.2.6 *Experimental Evaluation of the E^8-PRM*



(a) RRT Ext-Con, R=1 (b) RRT Con-Con, R=1 (c) E^8-PRM, $\lambda = 0$

(d) RRT Ext-Con, R=6 (e) RRT Con-Con, R=6 (f) E^8-PRM, $\lambda = 1$

(g) Plot of average collision checks vs solution path cost for the different algorithms.

Figure 2.6: Example runs with different planners, with the same sequence of samples. Note that E^8-PRM with $\lambda = 0$ is equivalent to the Lazy PRM. Red dots show collision checks.

I hope to show that the E^8-PRM is competitive with RRT-Connect in terms of planning effort required to find a feasible path.

We will compare this approach to single-query baseline approaches such as the RRT [17], EST [11], SBL [28], and Lazy PRM [2], approaches over lattices such as E-graphs, as well as asymptotically-optimal planners such as RRT* [14] and BIT* [7]. Also compare briefly against trajectory optimization approaches, though these are largely complementary. For example, there's CHOMP [36] and TrajOpt [29]. Defer discussion of caching to subsequent sections.

See Figure 2.6 for results on a simple 2D problem.

2.3 *The Multi-Set Planning Problem*

Motion planning approaches that build graphs in the collision-free subset of configuration space, e.g. the PRM [15] and RRT [20], have proven promising for high-dimensional articulated robotics problems in unstructured environments. These approaches devote a large amount of computational effort testing configurations and paths for collision, and the resulting graph can then be reused for other queries in the same collision-free subset. The E^8-PRM, introduced above in Section 2.2, is another such approach.

However, for manipuation problems, this subset of the robot's configuration space is sensitive to the locations and shapes of both people and objects in the environment, as well as the robot itself. In addition, it depends on the shape and pose of any object grasped by the robot. This makes it difficult not only to apply the results of prior planning computation to the current problem, but also to efficiently consider planned or hypothesized motions, since we must reconstruct our graph from scratch whenever the environment changes. This is especially the case for multi-step manipulation tasks that must be planned into the future.

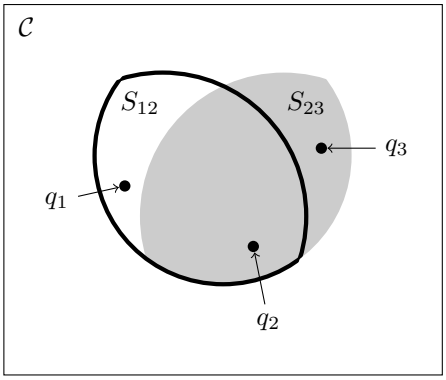
A large body of prior work has focused on methods to improve planning efficiency on manipulation problems. We show that many of these approaches are in fact special instances of a more general structure, which we formulate as the *multi-set planning problem*.

2.3.1 *Multi-Set Problem Formulation*

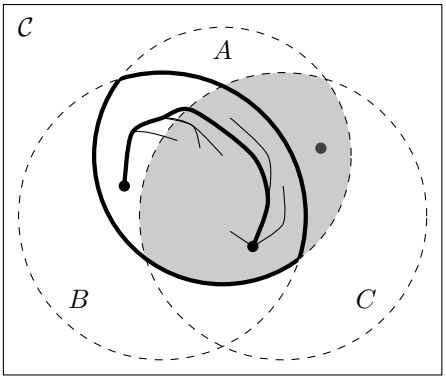
The multi-set planning problem is a generalization of both the movers' problem and the multi-query planning problem[5]. The reader is referred to Fig. 2.7 for a general example, as well as a simple instantiation on a 2D manipulation task. The multi-set problem formulation explicitly captures both planning and execution effort and can therefore be used as an ensemble effort model for use in the E^8-PRM planner across queries.

[5] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, Aug. 1996

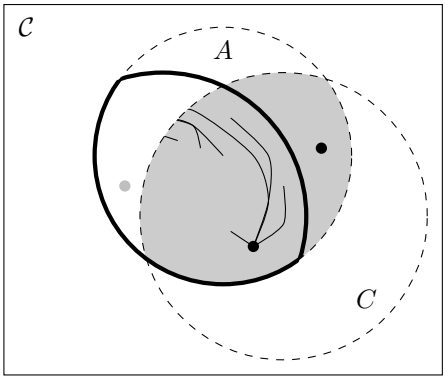
The multi-set planning problem is multi-query in a fixed configuration space $\mathcal{C}$. However, unlike related problems in which all queries demand solution paths contained within a single common subset of $\mathcal{C}$ (usually the set of collision-free configurations, denoted $\mathcal{C}_{\text{free}}$), the multi-set problem allows for the specification of a *family* of multiple such subsets $\mathcal{F} = \{A, B, \dots\}$. Like $\mathcal{C}_{\text{free}}$, each member of $\mathcal{F}$ is a subset of the common configuration space (that is, $S \subseteq \mathcal{C} \ \forall \ S \in \mathcal{F}$), and each subset S has its own indicator and planning estimator $\mathbf{1}_S[\cdot]$ and $\hat{p}_S(\cdot)$ as in Section 2.2.1. For example, in Fig. 2.7(e), $\mathcal{C}$-subset B consists of configurations free of collision between the robot and the



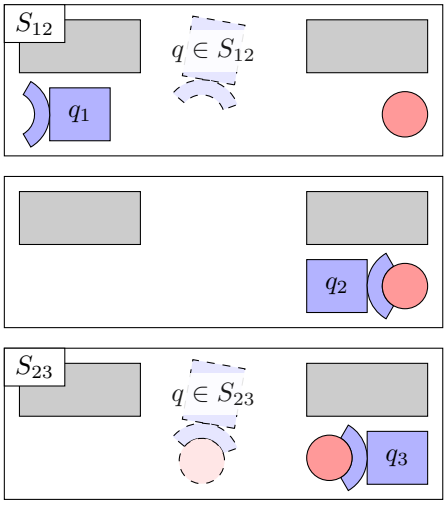
(a) A two-part multi-set problem in $\mathcal{C}$, first between q_1 and q_2 through S_{12}, then between q_2 and q_3 through S_{23}. The two free subsets S_{12} and S_{23} are distinct but related.



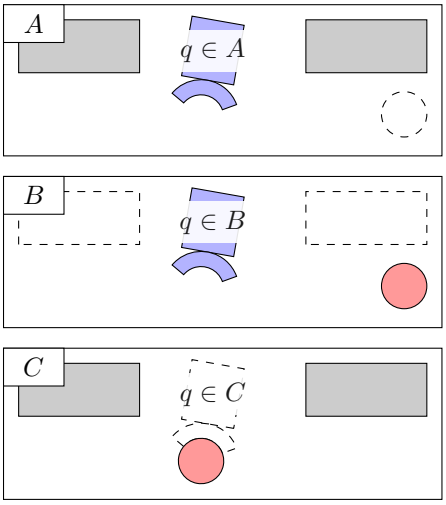
(b) The free subsets are related via other underlying subsets of $\mathcal{C}$, with $S_{12} = A \cap B$ and $S_{23} = A \cap C$. A planner solving the first part (from q_1 to q_2) has found paths in S_{12}.



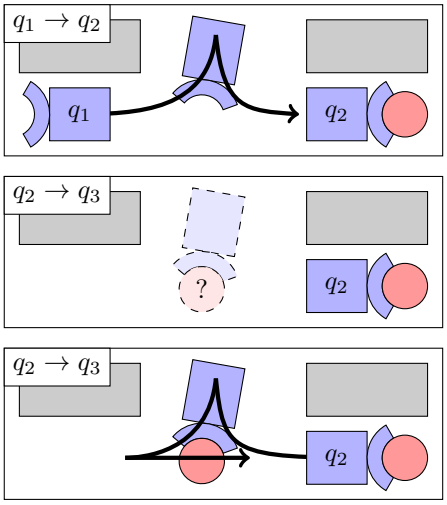
(c) Due to the set relations, a planner solving the second part (from q_2 to q_3 in S_{23}) can reuse any segment known to be in S_{12} by checking only for its membership in C.



(d) A forklift in a parking lot (q_1) must retrieve an object (q_2) and reverse park (q_3). This two-part problem requires plans in distinct collision-free $\mathcal{C}$-subsets S_{12} and S_{23}.



(e) Sets S_{12} and S_{23} are subsets of the configuration space of the robot $\mathcal{C} = \mathrm{SE}(2)$, and can be represented as intersections of underlying subsets A, B, and C as in (b).



(f) After planning a path from q_1 to q_2 (top), a planner can reuse a configuration in S_{12} (middle) by checking only for its membership in subset C, resulting in plan reuse (bottom).

Figure 2.7: An illustration of a multi-set planning problem in a common configuration space $\mathcal{C}$. The problem definition generalizes to an arbitrary number of configuration space subsets and set relations between them. When two queries in different subests are solved sequentially, a multi-set planner can reuse path segments less expensively. See Section 2.3.2 for examples in manipulation.

initial object pose.

The problem supports an arbitrary number of queries $\mathcal{U}$. Each query u demands a solution path through a *single* $\mathcal{C}$-subset $U \in \mathcal{F}$ (see Fig 2.8):

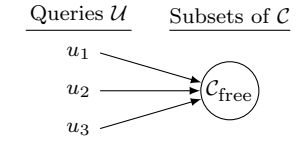
$$u : (q_{start},\ q_{goal},\ U). \tag{2.16}$$

Finally, the multi-set problem incudes a list of set relations $\mathcal{R}$ between the $\mathcal{C}$-subsets in $\mathcal{F}$. These can be expressed directly using set theoretic relations, or equivalently as logical statements on the corresponding indicator functions. Common types of such relations (containment and intersection) are illustated in Fig. 2.9. Fig. 2.7 gives an example of intersection relations; an example of containment is a padded (conservative) robot model (see Section 2.3.2).

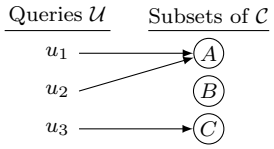
Together, these four elements (a configuration space $\mathcal{C}$, subsets $\mathcal{F}$ each with endowed indicators, a set of queries $\mathcal{U}$, and a list of subset relations $\mathcal{R}$) comprise a multi-set planning problem.

Example Problem. Consider the diagram from Fig. 2.7. $\mathcal{F}$ consists of five $\mathcal{C}$-subsets labeled A, B, C, S_{12}, and S_{23}, and we have two queries, $u_{12} : (q_1, q_2, S_{12})$ and $u_{23} : (q_2, q_3, S_{23})$. $\mathcal{R}$ consists of the two relations $S_{12} = A \cap B$ and $S_{23} = A \cap C$. Suppose a cost model $\mathcal{M}$ wherein evaluating the indicator $\mathbf{1}_A$ incurs cost 4, evaluating $\mathbf{1}_B$ and $\mathbf{1}_C$ incurs cost 2, and evaluating $\mathbf{1}_{S_{12}}$ and $\mathbf{1}_{S_{23}}$ incurs cost 6. In the manipulation example in Fig. 2.7(d), this would be the case if each pairwise outlined shape collision check incurs unit cost.

Suppose a graph structure within S_{12} has been grown to solve the first query u_{12}. During the subsequent solve of query u_{23}, an existing path segment known to be in S_{12} can be shown to also be contained within S_{23} by only evaluating $\mathbf{1}_C$. In the manipulation example, reusing an a configuration from the previous search would require only a check of cost 2, instead of cost 6 for a new configuration. Thus, we might hope that a planner may be biased towards reusing said path segments in this case.

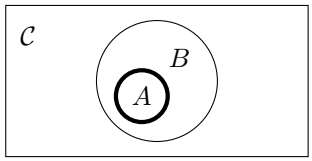


(a) Multi-query planning



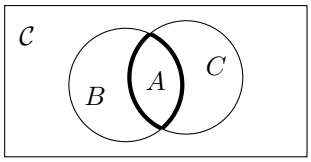
(b) Multi-set planning

Figure 2.8: While queries in multi-query planning reference the same subset of $\mathcal{C}$, each multi-set query references one of a number of such sets.



$A \subseteq B$
$\mathbf{1}_A \Rightarrow \mathbf{1}_B$

(a) Containment relation



$A = B \cap C$
$\mathbf{1}_B \wedge \mathbf{1}_C \Rightarrow \mathbf{1}_A$

(b) Intersection relation

Figure 2.9: Types of subset relations. Each relation can be expressed directly as set relations w.r.t a set S, or equivalently as logical statements on the corresponding indicator functions $\mathbf{1}_S(\cdot)$.

2.3.2 *Multi-Set Problems in Manipulation Tasks*

Instances of multi-set problems are especially prevalent when planning for manipulation tasks with articulated robots. This section details several such instances. While these instances are discussed separately here, they are often present simultaneously. See Section 2.4.3 for experimental results for a problem which includes several of the multi-set problem instances described below.

We also provide an implementation for the OpenRAVE [6] virtual kinematic planning environment which automatically discovers $\mathcal{C}$-subsets in manipulation tasks.

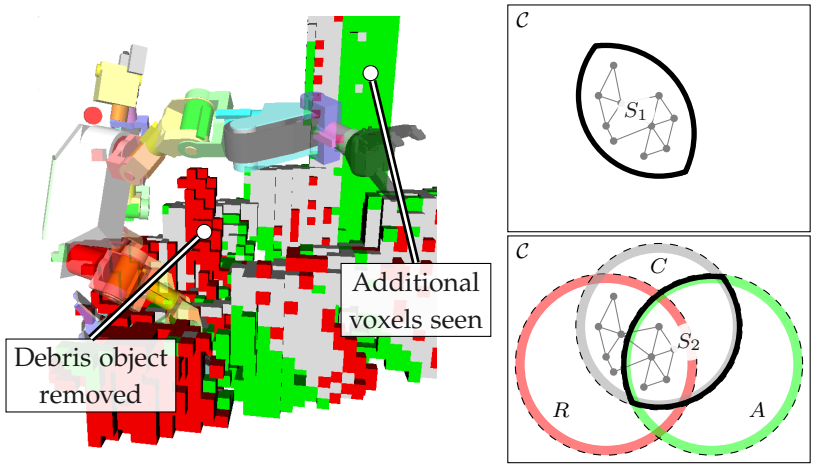


Figure 2.10: Structured or unstructured dynamic environments can be represented as a multi-set problem (see Section 2.3.2).

(Left) A disaster response robot maintains a dynamic unstructured environment model using coarse voxels (scene data from a debris-clearing task at a recent disaster response competition). Since the last planning query, voxels have been added (green) and removed (red).

(Right) $\mathcal{C}$-subsets and relations can be added retroactively. Here, the graph for an initial query is checked w.r.t S_1. After environment changes, S_1 is redefined in terms of the $\mathcal{C}$-subsets derived from the set of common, added, and removed elements, allowing for reuse on a query in S_2. Here, the planner need only check existing path segments against added voxels in order to reuse them for the current query.

Dynamic Environments. The sensors on most articulated robots allow them to maintain dynamic environment models to track changing collision geometry. These models might be structured (e.g. recognizing objects with known models) or unstructured (e.g. occupancy models). In both cases, even in a changing world, there are often areas that are fixed between planning queries.

Prior work (e.g. [12]) leverages this by imposing a dichotomy between *fixed* and *moving* components of $\mathcal{C}$. Our formulation extends this to an arbitrary number of such labels, including ones defined retroactively (i.e. during planning; see Fig. 2.10). By explicitly labeling such areas in workspace (and leveraging the *set containment* property [23]), we can represent this structure as a multi-set planning problem.

Grasped Objects One instance specific to manipulation problems is the handling of grasped objects. For example, consider a manipulator which grasps a geometric object. This affects the set of collision-free configurations across a large section of $\mathcal{C}$ relative to the old set of valid configurations S_{old}. However, the resulting $\mathcal{C}$-subset S_{new} can be represented simply as $S_{\text{new}} = S_{\text{old}} \cap G$, with G the set of robot configurations in which the *grasped object* (only) is deemed free of collision with the robot and environment. This structure is discussed in the context of the *conditional reachability graph*, part of the FFRob heuristic framework [8].

For example, consider the manipulation problem in Figure 2.14. The robot must find a path which moves its arm to grasp the cup. After the cup is grasped, the robot can reuse any edge in the existing roadmap by simply checking the grasped cup against the remainder of the environment. This structure, together with the approach to dynamic environments, are included together in the experimental results (Section 2.4.3).

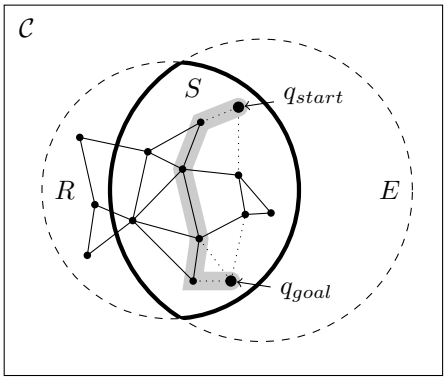


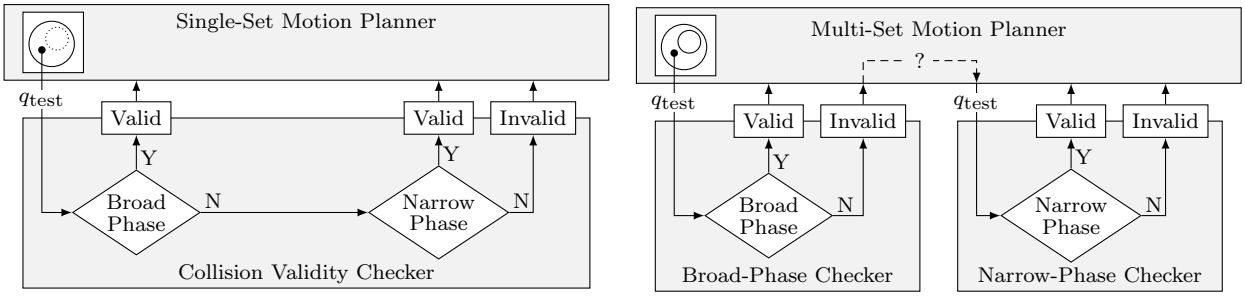
Figure 2.11: A roadmap is pre-computed in R, the subset of $\mathcal{C}$ consisting of configurations free of robot self-collision. Online, the planner must find a path that's also within E, the subset free of environment collision. When solving this query in $S = R \cap E$, the Multi-Set PRM automatically prefers potential paths with pre-computed

Cached Self-Collision-Checked Roadmaps Self-collision checking is a potentially expensive component to articulated motion planning; in contrast to environment checking, it is fundamentally quadratic in the number of moving links. Further, pairs of links to be checked tend to be relatively close to each other, reducing the effeciveness of broad-phase approaches.

Leven and Hutchinson [21] introduced the concept of a pre-cached roadmap consisting of configurations and paths already known to be valid w.r.t. self-collision. As a type of invariant in $\mathcal{C}$, this can be seen as a particular instance of multi-set planning. See Fig. 2.11.

Integration with Broad-Phase Collision Checking The multi-set formulation also enables motion planners to reason directly about different robot or environment models. For example, consider two geometric robot models, one with high quality (e.g. from a CAD program), and one hand-tuned "padded" model consisting of a small number of simple conservative bounding volumes. The $\mathcal{C}$-subsets derived from these models are related by $R_{\text{padded}} \subseteq R_{\text{CAD}}$. Collision checkers currently use a similar approach internally to speed up collision checks (see Fig. 2.12. and Fig 2.13).

Figure 2.12: Collision validity checking is a commonly used indicator function. The multi-set formulation allows an intelligent planner to reach inside the checker's "black box" and reduce the number of costly narrow-phase checks. Resulting paths tend to be cheaper to compute and stay further from obstacles.



(a) A single-set planner testing simply for membership in $\mathcal{C}_{\text{free}}$ treats a collision validity checker as a "black box." Internally, modern checkers first employ an inexpensive broad-phase check using a low-dimensional conservative representation to quickly identify non-colliding bodies before resorting to an expensive narrow-phase check.

(b) A multi-set planner can explicitly reason about the conservative nature of the broad-phase check. This allows it to defer some narrow phase checks (often indefinitely) and instead prefer paths that require fewer expensive checks.

2.4 *Multi-Set as an Effort Model for Roadmap Planners*

This section lays out how to leverage the multi-set formulation as a planning effort model for use in the E^8-PRM. The resulting algorithm, the Multi-Set PRM, exploits the multi-set structure inherent in manipulation problems by efficiently reuses planning computation between related queries.

2.4.1 *The Multi-Set PRM*

The Multi-Set PRM (Algorithm 6) is a simple extension of the E^8-PRM (Section 2.2). They key differences are that (a) it reasons about $\mathcal{C}$-subsets using logical propositions, and (b) it uses a multi-set ensemble effort model $\mathcal{M}_{\text{multi}}$ to capture relations between these subsets.

Algorithm 6 Multi-Set PRM Planner

1: $G \leftarrow$ empty graph
2: $P_{\text{global}} \leftarrow$ global propositions from $\mathcal{R}$
3: **procedure** MULTISETPRM($G, V_{\text{start}}, V_{\text{goal}}, U, N, \lambda$)
4: $\quad \mathcal{M} \leftarrow \mathcal{M}_{\text{valid}}(U) + \mathcal{M}_{\text{exec}}$
5: $\quad$ **return** E^8-PRM($G, V_{\text{start}}, V_{\text{goal}}, \mathcal{M}, N, \lambda$) ▷ See Algorithm 5.

Set Relations as Logical Propositions. The planner represents the list of set relations $\mathcal{R}$ (Section 2.3) specified in the multi-set problem formulation as a set of *logical propositions* P_{global} which are considered globally applicable. For example, the proposition $\mathbf{1}_A \Rightarrow \mathbf{1}_B$ follows from the relation $A \subseteq B$ (see Fig. 2.9). In addition, each edge e in the roadmap G is augmented with an initially empty set $e.P$ containing all additional edge-specific propositions known to be true as a result of any indicator functions evaluated over that edge. For example, if planner evaluated the indicator $\mathbf{1}_B[e]$ and it returned False, the proposition $\neg\mathbf{1}_B$ would be added to $e.P$. Together, such sets of propositions can be used as *premises* as part of an *argument* to demonstrate a *conclusion*; a logical solver can then be used validate or invalidate the argument. For example, an argument with these premises and conclusion $\{(\mathbf{1}_A \Rightarrow \mathbf{1}_B), (\neg\mathbf{1}_B)\} \Rightarrow (\neg\mathbf{1}_A)$ can be shown to be valid.

A Multi-Set Ensemble Effort Model. The key to the Multi-Set PRM is the ensemble effort model $\mathcal{M}_{\text{multi}}$ (Algorithm 7) which allows for evaluation of an edge as well as estimates of its planning and execution effort in a way that takes advantage of multi-set structure. The core of this model is the MULTIOPTCERT function which takes as input an edge q_e, a query $\mathcal{C}$-subset U, and a set of known logical propositions P_{known} for that edge, and as output produces an

optimistic validation certificate consisting of subset indicators which if evaluated would validate the membership of the edge in U with lowest effort.

Algorithm 7 Multi-Set Validitiy Effort Model $\mathcal{M}_{\text{multi}}$

function $x_{\text{multi}}(e, U)$
  $(\mathcal{F}_{\text{cert}}, b_{\text{cert}}, \hat{p}_{\text{cert}}) \leftarrow \textsc{MultiOptCert}(q_e, U, P_{\text{global}} \cup e.P_{\text{eval}})$
  for all $S \in \mathcal{F}_{\text{cert}}$ **do**
    $b_{\text{eval}} \leftarrow \mathbf{1}_S[q_e]$
    $e.P_{\text{eval}} \leftarrow e.P_{\text{eval}} \cup \left\{ \begin{array}{c} \mathbf{1}_S \text{ if } b_{\text{eval}} \\ \neg\mathbf{1}_S \text{ otherwise} \end{array} \right\}$
    if $b_{\text{eval}} \neq b_{\text{cert}}(S)$ **then**
      return ∞
  return 0
function $\hat{x}_{\text{multi}}(e, U)$
  return 0
function $\hat{p}_{\text{multi}}(e, U)$
  $(\mathcal{F}_{\text{cert}}, b_{\text{cert}}, \hat{p}_{\text{cert}}) \leftarrow \textsc{MultiOptCert}(q_e, U, P_{\text{global}} \cup e.P_{\text{eval}})$
  return $\hat{p}_{\text{cert}}$

Calculating the Multi-Set Optimistic Certificate The MultiOptCert function (Alg. 8) is tasked with computing the optimistically optimal set of indicator evaluations to perform for the edge in order to validate its membership in the query $\mathcal{C}$-subset U. The function returns three elements: (a) the family of $\mathcal{C}$-subsets $\mathcal{F}_{\text{cert}} \subseteq \mathcal{F}$ whose indicators are to be evaluated, (b) a binary function b_{res} which provides the desired indicator result for each evaluation, and (c) the total evaluation cost $\hat{p}_{\text{cert}}$ given by the $\hat{p}_S[\cdot]$ functionals (Section 2.3.1).

Algorithm 8 Multi-Set Optimistic Certification

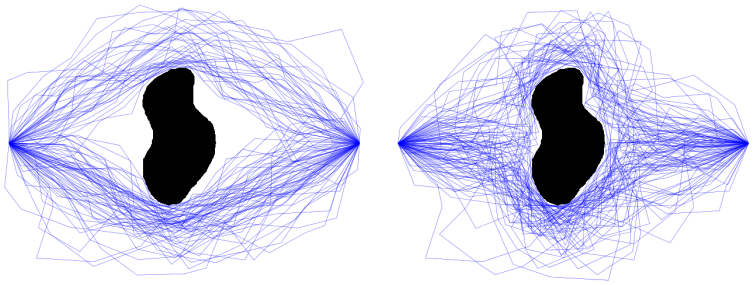
function $\textsc{MultiOptCert}(q_e, U, P_{\text{known}})$
  $\mathcal{T}_{\text{imply}} \leftarrow \emptyset$
  for all $\mathcal{F}_{\text{cert}} \in \mathcal{P}(\mathcal{F})$ **do**
    $\hat{p}_{\text{cert}} \leftarrow \sum_{S \in \mathcal{F}_{\text{cert}}} \hat{p}_S[q_e]$
    for all b_{res} **s.t.** $b_{\text{res}} : \mathcal{F}_{\text{cert}} \to \{\text{True}, \text{False}\}$ **do**
      $P_{\text{res}} \leftarrow \left\{ \begin{array}{c} \mathbf{1}_S \text{ if } b_{\text{res}}(S) \\ \neg\mathbf{1}_S \text{ otherwise} \end{array} \middle| \, S \in \mathcal{F}_{\text{cert}} \right\}$
      if $P_{\text{known}} \cup P_{\text{res}} \Rightarrow \mathbf{1}_U$ is valid **then**
        $\mathcal{T}_{\text{imply}} \leftarrow \mathcal{T}_{\text{imply}} \cup \{(\mathcal{F}_{\text{cert}}, b_{\text{res}}, \hat{p}_{\text{cert}})\}$
  return $(\mathcal{F}_{\text{cert}}, b_{\text{res}}, \hat{p}_{\text{cert}}) \in \mathcal{T}_{\text{imply}}$ with lowest $\hat{p}_{\text{cert}}$

The function accumulates a set $\mathcal{T}_{\text{imply}}$ of valid certificates which would imply membership in U. We proceed by considering all com-

binations of available $\mathcal{C}$-subset indicators $\mathcal{F}_{\text{cert}}$ (line 3). For each set of evaluations, we compute the planning effort $\hat{p}_{\text{cert}}$ which would be required. We then consider all possible outcomes for each indicator by iterating over all functions b_{res} mapping from $\mathcal{C}$-subset S to binary values (line 5). For each potential outcome b_{res}, we form the set of additional propositions P_{res}, and then use a propositional logic solver to determine whether the aggregate premises imply membership in the query $\mathcal{C}$-subset U. If so, this certificate is added to $\mathcal{T}_{\text{imply}}$, and the lowest-effort certificate is returned.

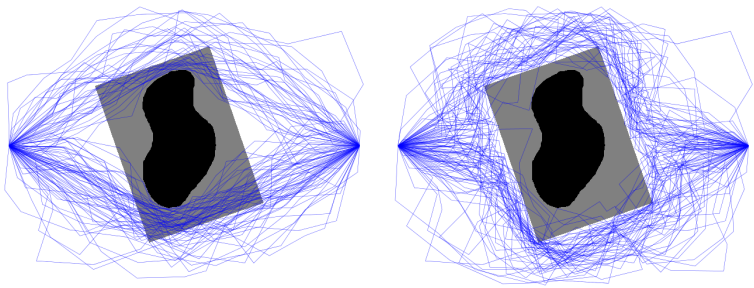
2.4.2 *Behavior of the Multi-Set PRM*

See Figure 2.13 for a simple example with integrated broad-phase collision checking as described in Section 2.3.2.



(a) Paths with $\lambda = 0$
Average length: 733.0
Average check cost: 7219.5

(b) Paths with $\lambda = 1$
Average length: 836.5
Average check cost: 4692.6



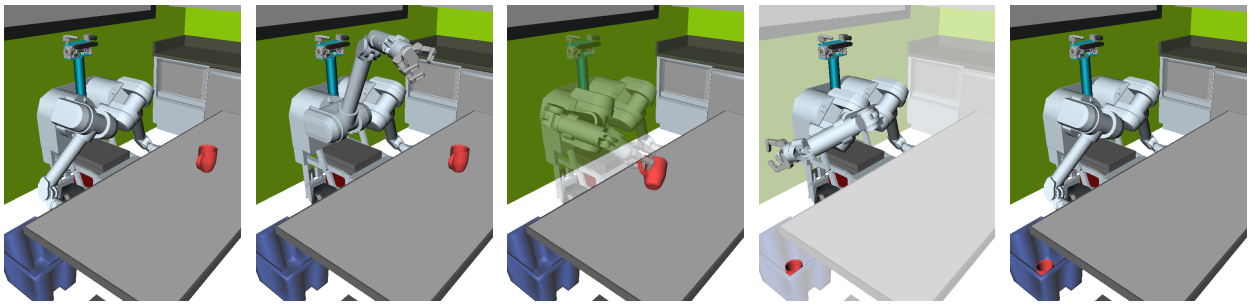
(c) Paths with $\lambda = 0$
Average length: 733.0
Average check cost: 2685.7

(d) Paths with $\lambda = 1$
Average length: 907.1
Average check cost: 1064.5

Figure 2.13: A simple 2D example of the Multi-Set PRM using a broad-phase check. Checking for collision with the grey box is 10x less expensive than with the actual black obstacle.

2.4.3 *Home Robot Manipulation Task Results*

We tested the Multi-Set PRM on the manipulation task described in Fig. 2.14. We used the r-disk PRM construction rule with $r = 2.0$ rad, and a batch size of $N = 1000$. Planning times are measured on a Lenovo T430s laptop. The planner was asked to solve each of the steps of the plan sequentially.



(a) Starting Configuration (b) Step 1, in S_1 (c) Step 2, in S_2 (d) Step 3, in S_3 (e) Ending Configuration.

Figure 2.14: A home robot performing a three-step manipulation task. It must move from its home configuration to grasp the cup, transfer it to a drop location above the bin, and return home. Experimental results for the Multi-Set PRM are shown in Table 2.2

We varied (a) the planning vs. execution parameter λ (see Section 2.1.4), and (b) the subset relations provided to the planner as described in Section 2.3.2. We measured the time required for planning (s) and the length of the resulting solution resulting path (rad) for each step of the task.

Note that the Multi-Set PRM, with no relations specified and $\lambda = 0$ reduces to the Lazy PRM. As expected, increasing λ resulted in decreased planning times but yielded longer paths. Including more $\mathcal{C}$-subset relations also significantly reduced planning times, and had little effect on path lengths on this problem. Note that the planning time results when using the cached self-collision-checked roadmap, denoted by (*), do not include the pre-computation time.

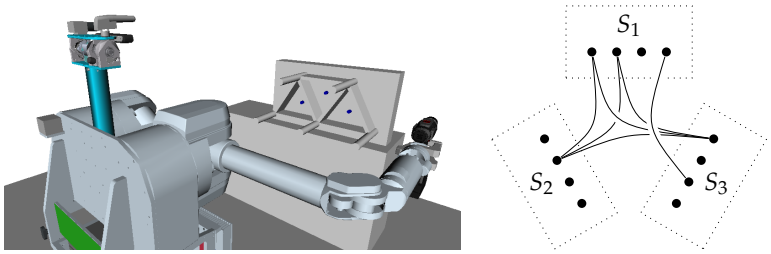
Note that including inter-step relations drastically reduces planning times for subsequent steps. We expect this trend would continue as more steps are included. Also, note that when $\lambda = 0$, path length is unchanged as the number of set relations is changed – this is because the paths that are selected for evaluation by the algorithm are a function only of their (constant) lengths.

Table 2.2: Home robot manipulation task results. The entry with no relations and $\lambda = 0$ is equivalent to the LazyPRM.

Relations	Cost	$\lambda = 0$				$\lambda = 0.5$				$\lambda = 1$			
		Step 1	Step 2	Step 3	Total	Step 1	Step 2	Step 3	Total	Step 1	Step 2	Step 3	Total
None	Plan	6.16 s	3.72 s	2.38 s	12.25 s	5.52 s	2.89 s	2.12 s	10.53 s	3.39 s	2.25 s	2.12 s	7.76 s
	Exec	14.22 rad	8.51 rad	4.23 rad	26.97 rad	15.07 rad	10.60 rad	4.23 rad	29.89 rad	15.07 rad	10.60 rad	4.23 rad	29.89 rad
Inter-Step	Plan	6.40 s	2.33 s	0.86 s	9.59 s	5.40 s	1.55 s	0.91 s	7.86 s	3.38 s	0.91 s	0.30 s	4.59 s
(Sec. 2.3.2, 2.3.2)	Exec	14.22 rad	8.51 rad	4.23 rad	26.97 rad	15.07 rad	12.21 rad	4.23 rad	31.51 rad	15.07 rad	12.21 rad	7.11 rad	34.40 rad
Self-Checked	Plan*	3.54 s	2.23 s	1.17 s	6.94 s	2.99 s	1.77 s	1.16 s	5.92 s	1.47 s	1.22 s	1.16 s	3.85 s
(Sec. 2.3.2)	Exec	14.22 rad	8.51 rad	4.23 rad	26.96 rad	14.22 rad	10.06 rad	4.23 rad	28.51 rad	14.22 rad	10.60 rad	4.23 rad	29.05 rad
Both	Plan*	3.25 s	1.79 s	0.90 s	5.94 s	2.88 s	1.55 s	0.92 s	5.35 s	1.47 s	1.88 s	0.31 s	3.66 s
	Exec	14.22 rad	8.51 rad	4.23 rad	26.96 rad	14.22 rad	8.51 rad	4.23 rad	26.96 rad	14.22 rad	9.64 rad	6.36 rad	30.22 rad

2.5 *Comprehensive Multi-Root Planning*

While the Multi-Set PRM allows planning computation to be reused between queries across steps of a manipulation task, a higher-level task planner must still distribute these queries in order to discover a low-cost path through all of the task's steps (see Figure 2.1). The types of queries we have discussed so far (using sets V_{start} and V_{goal}) take an *any-to-any* form – the planner will return a successful path which connects from any start to any goal vertex. However, when planning in parallel for multiple steps, a task planner instead requires a *diverse* set of paths.



(a) A real-world problem. (b) Illustration.

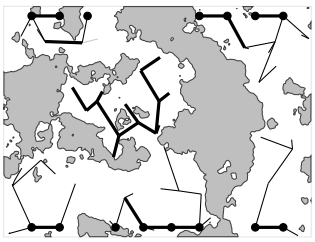
Figure 2.15: The comprehensive multi-root (CMR) problem.

To handle this, we formulate and study the *comprehensive multi-root* (CMR) planning problem [6] (Figure 2.15), in which feasible paths are desired between multiple regions. We propose two primary contributions which allow us to extend state-of-the-art sampling-based planners. First, we propose the notion of *vertex coloring* as a compact representation of the CMR objective on graphs. Second, we propose a method for *deferring edge evaluations* which do not advance our objective, by way of a simple criterion over these vertex colorings. The resulting approach can be applied to any CMR-agnostic graph-based planner which evaluates a sequence of edges. We prove that the theoretical performance of the colored algorithm is always strictly better than (or equal to) that of the corresponding uncolored version. We then apply the approach to the Probabalistic RoadMap (PRM) algorithm; the resulting *Colored Probabalistic RoadMap* (cPRM) is illustrated on 2D and 7D CMR problems.

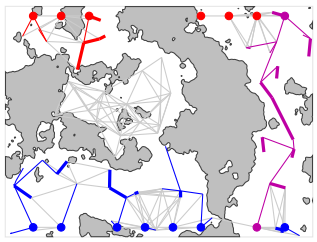
[6] C. M. Dellin and S. S. Srinivasa. A general technique for fast comprehensive multi-root planning on graphs by coloring vertices and deferring edges. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015

2.5.1 *The Comprehensive Multi-Root Problem*

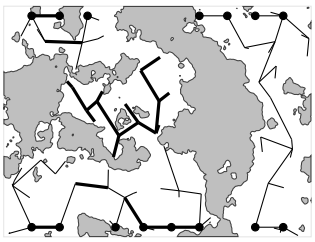
We work with the robot's configuration space $\mathcal{C}$ and its collision-free subset $\mathcal{C}_{\text{free}}$. We consider the general problem with N root sets in this space $\{S_1, \ldots, S_N\}$. We seek a diverse set of feasible paths between root sets – that is, we want to maximize the number of connected roots between sets. We call this the *comprehensive multi-root* (CMR) planning problem.



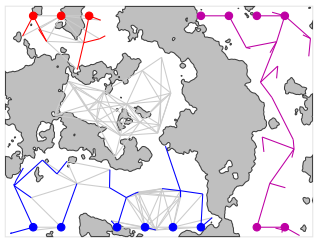
(a) PRM (86 Edges Checked) (195 Edges Considered, No Pairs)



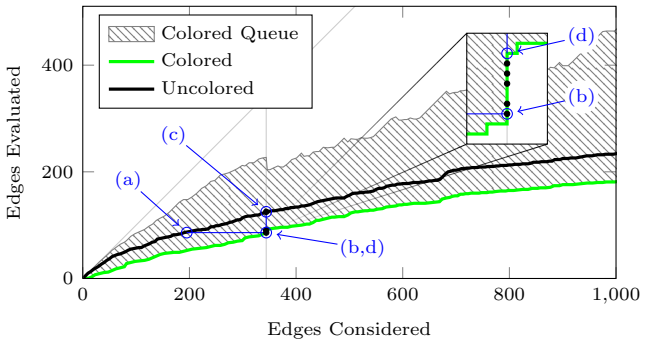
(b) cPRM (86 Edges Checked) (344 Edges Considered, 1 Pair)



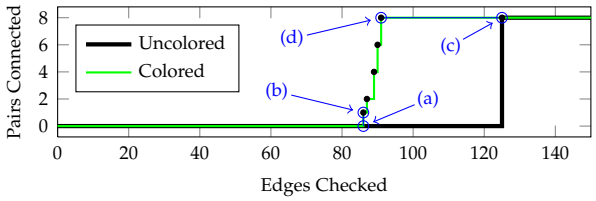
(c) PRM (344 Edges Considered) (125 Edges Checked, 8 Pairs)



(d) cPRM (344 Edges Considered) (91 Edges Checked, 8 Pairs)



(e) Edges considered, evaluated, deferred, and skipped.



(f) CMR Objective vs edges evaluated.

Figure 2.16: A comparison between an uncolored (a), (c) and colored (b), (d) forest-of-trees PRM on the same sequence of edges. Plot (e) shows the evolution of edges in both algorithms as they progress, and plot (f) shows the number of between-rootset pairs found by each.

We track progress via the r-score:

$$r = \left|\{\textsc{Path}(x_a, x_b) \mid x_a, x_b \text{ in different root sets}\}\right|. \tag{2.17}$$

For example, the solution illustrated in Figure 2.15(b) has an r-score of 6 out of a maximum of 27. Our objective is to *maximize* the r-score as *quickly* as possible.

2.5.2 *A CMR Example*

See Figure 2.16 for a sample CMR problem between a set of start vertices (at top) and a set of goal vertices (at bottom). This example uses a simple forest-of-trees PRM construction rule.

2.6 *The* PROTEUS *Task Planner*

I propose to write a simple task planner, PROTEUS (PROTEUS Reasons Over Task Effort Using Sampling). For a specified coupled manipulation planning problems, it will compute its multi-set structure and call into parallel instances of the Multi-Set PRM for each step using the CMR objective in order to find a solution path. This will allow for full-scope experimental evaluations to be conducted on the approach. We assume the step decomposition is provided by either a human operator or symbolic planner.

Interleaved Planning and Execution. While primary comparisons will consider planning in a separate phase from execution, PROTEUS will also support a simple version of interleaved planning execution for comparison to baseline approaches which commit to early choices. In fact, the approach in this thesis is amenable to interleaved planning and execution, since the Multi-Set PRM can continually updated the q_{init} configuration. See research question Q8 for details.

3
Summary of Proposed Work

3.1 Research Questions

I propose to address the following eight research questions in this thesis. This section discusses each question in turn. See Table 3.1 for a timeline.

	Research Question	Sec.
Q1	How should the λ parameter mediating between planning and execution cost in the E^8 search algorithm be chosen?	2.1
Q2	How can incremental graph search ideas (e.g. LPA*) be used to efficiently implement the E^8 algorithm?	2.1
Q3	How does the explicit graph approach of E^8 compare to weighted and anytime implicit algorithms?	2.1
Q4	How should discrete graphs be constructed in continuous $\mathcal{C}$-spaces with spatially correlated execution costs?	2.2
Q5	What path evaluation strategy performs best across manipulation tasks?	2.2
Q6	How does the extent of disclosed multi-set structure affect planner performance on multi-step problems?	2.4
Q7	How does the amount of inter-step coupling affect the merit of interleaving planning and execution?	2.6
Q8	How does the PROTEUS task planner compare experimentally to baseline approaches on HERB and CHIMP?	2.6

Q1 How should the λ parameter mediating between planning and execution cost in the E^8 search algorithm be chosen?

See Section 2.1.

The E^8 algorithm's objective, from (2.6), trades off between plan-

ning and execution effort, mediated by the parameter $\lambda \in [0, 1]$. Loosely, adjusting λ varies the behavior between *feasibility* (finding paths quickly) and *optimality* (of execution).

With $\lambda = 0$, E^8 focuses solely on minimizing execution effort. This is similar to the LazyPRM algorithm. On the other hand, with $\lambda = 0$, it focuses solely on minimizing planning effort, similarly to the RRT algorithm. We discuss these comparisons to existing continuous-space planners in Section 2.2.

If our ensemble effort model $\mathcal{M}$ specifies effort estimates in the same units (e.g. time or energy), $\lambda = \frac{1}{2}$ is a natural choice. However, due to the optimistic nature of the E^8 algorithm, a different value of λ may perform better in practice.

Minimizing total time in a greedy fashion implies $\lambda = 0.5$. For later steps in a multi-step plan, we might have an estimate of the probability P_e that the given query will actually be executed. We can then pose our optimistic objective as total planning and execution time in expection; this induces the following parameter choice:

$$\lambda = \frac{1}{1 + P_e}. \tag{3.1}$$

For example, $P_e = 1$ induces $\lambda = 0.5$; as $P_e \to 0$, $\lambda \to 1$. In other words, as the estimated probability of executing the path goes down, the planner becomes greedier w.r.t. planning effort at the expense of costlier solution paths.

This is all one-step greedy; it returns the optimal path optimistically, assuming it will be collision-free. If we have some estimate of the proportion P_u of evaluated edges which will be part of the final path, we can then choose a cost function which downweights the planning time.

Q2 How can incremental graph search ideas (e.g. LPA) be used to efficiently implement the E^8 algorithm?*

The E^8 algorithm (Algorithm 2) from Section 2.1 is currently executing a fully bidirectional Dijkstra's search during each iteration. The algorithm is clearly making multiple graph search queries over a fixed graph with only a few edge costs changing between queries. While, for small graphs, graph search time is small, for larger graphs, it will become significant. How can we efficiently implement LPA* [1] to improve planning efficiency on large graphs?

[1] S. Koenig, M. Likhachev, and D. Furcy. Lifelong Planning A*. *Artificial Intelligence Journal*, 2004

Q3 How does the explicit graph approach of E^8 compare to weighted and anytime implicit algorithms?

This question addresses the empirical performance of the E^8 algorithm when compared to baseline algorithms on implicit and explicit graphs with expensive edge evaluations. We will use metrics of measured planning and execution effort expended both on 2D graphs and on fixed lattice and RGG graphs in HERB's configuration space.

Q4 How should discrete graphs be constructed in continuous $\mathcal{C}$-spaces with spatially correlated execution costs?

Chapter 2.2 discusses how to embed roadmaps in $\mathcal{C}$ so that they can be searched by E^8.

The problem with naïvly running E^8 on a dense roadmap in $\mathcal{C}$ is that it tends to bunch up in local minima. This is because reducing the continuous planning problem to a graph search ignores the spatial correlation inherent in $\mathcal{C}_{\text{free}}$.

One way to capture this is to maintain a probabalistic model of $\mathcal{C}_{\text{free}}$, and then optimize in expectation. In particular, instead of greedily choosing the best path based on optimistic estimates of one-time planning and execution cost:

$$f(\pi) = \lambda \hat{f}_p(\pi) + (1-\lambda)\hat{f}_x(\pi), \tag{3.2}$$

we instead reason over the total *expected* remaining cost:

$$f(\pi) = E\left[\lambda f_p(\pi) + (1-\lambda) f_x(\pi)\right] \tag{3.3}$$

$$= P_{\text{free}}(\pi)\left[\lambda \hat{f}_p(\pi) + (1-\lambda)\hat{f}_x(\pi)\right] + (1 - P_{\text{free}}(\pi))\left[\lambda F_p + (1-\lambda)F_x\right] \tag{3.4}$$

Consider the the problem from Figure 3.1. There are an infinite number of paths to the goal, each consisting of walking along the sidewalk, followed by crossing the street perpendicuarly at a particular position x. The sidewalk is known to be collision-free, whereas each position on the street must be tested for collision with obstacles with planning validation cost $\hat{f}_p(\pi)$ independent of x. Execution cost $f_x(\pi)$ is given by $|x| + c$.

Suppose we first test walking straight across the street π_0 (knowing nothing, this is clearly the optimistically cheapest path) and this is deemed in collision. Which path should we consider next (e.g π_a or π_b)?

What is our model for $P_{\text{free}}(\pi)$? Relate to GPs for classification[2].

We are operating under assumptions:

- Single-shot greedy (won't choose *sets* of paths which minimize remaining effort)

[2] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, Massachusetts and London, England, 2006. Available at http://www.gaussianprocess.org/gpml/

- Operates over *paths* instead of configurations or edges (won't probe points, no explicit exploration)

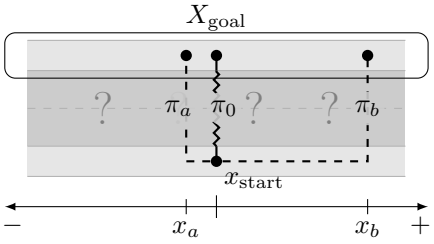


Figure 3.1: Simple example problem to illustrate optimizing remaining ensemble cost in expectation.

Q5 What path evaluation strategy performs best across manipulation tasks?

Because the E^8 and E^8-PRM algorithms operate over explicit graphs and roadmaps, they have the freedom to evaluate edges in any order. In contrast, search algorithms over implicit graphs must evaluate vertices via e.g. the graph's SUCCESSORS(v) function. This freedom may allow intelligent alternating or bisection approaches to reduce average-case planning cost.

Since E^8 operates on graphs, it is constrained to evaluating entire edges, one at a time. On the other hand, the E^8-PRM roadmap algorithm has the luxury of evaluating individual configurations at any point on the path; the Lazy PRM algorithm, for example, reported promising results by performing a bisection collision check on the entire path.

This question serves to identify the best-performing path evaluation strategies during testing on HERB and CHIMP manipuation tasks.

Q6 How does the extent of disclosed multi-set structure affect planner performance on multi-step problems?

The preliminary HERB experimental results reported in Table 2.2 evaluate the Multi-Set PRM's performance with different combinations of multi-set structure is provided to the planner. The purpose of this question is to extend this analysis to a broader set of manipulation tasks which exploit other types of multi-set structure, such as those listed in Section 2.3 in order to identify particular types of structure which yield the most promising performance.

Q7 How does the amount of inter-step coupling affect the merit of interleaving planning and execution?

As discussed in the introduction, a large amount of coupling between tasks – what we loosely define as the extent to which early choices affect the performance of later portions of the task – motivates longer planning horizons. The importance of considering this coupling is motivated by the performance of the planning system used on the CHIMP robot at the DRC Trials competition [3], which often failed to find solutions to multi-step tasks because it committed too early.

We hypothesize that highly coupled tasks will favor approaches such as PROTEUS (Chapter 2.6) that wait to find a full task plan before beginning execution, while less coupled tasks will favor interleaved approaches such as Hierarchical Task and Motion Planning in the Now [4]. The purpose of this question is to validate this hypothesis through experimental evaluation on HERB and CHIMP.

[3] C. Dellin, K. Strabala, G. C. Haynes, D. Stager, and S. Srinivasa. Guided manipulation planning at the DARPA Robotics Challenge trials. In *2014 International Symposium on Experimental Robotics (ISER 2014)*, June 2014

[4] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1470–1477, May 2011

Q8 How does the PROTEUS *task planner compare experimentally to baseline approaches on* HERB *and* CHIMP*?*

This question serves to evaluate the full task performance of the proposed approach in comparison to state-of-the-art baseline approaches. In order to evaluate the efficacy of the approach, this chapter details a set of experiments on multiple robot systems both in simulation and on real hardware.

Here, we describe the test platforms, software, metrics, control variables, and baseline approaches that are used in the experimental evaluation.

HERB: The Home Exploring Robot Butler. The HERB robot [32] is human-scale mobile manipulator designed to assist in home environments. It has two seven-degree-of-freedom Barrett WAM arms mounted on a Segway RMP base. It includes a BLAH KWh battery and has three onboard rack-mount computers.

CHIMP: the CMU Highly Intelligent Mobile Platform. CHIMP [33] is a disaster response robot built by NREC at CMU for the Darpa Robotics Challenge. It is a tracked mobile manipulation robot consisting of four limbs comprising 26 limb degrees of freedom. It has Robotiq adaptive 3-finger grippers.

OpenRAVE Simulation Environment. We use the OpenRAVE [6] simulation environment for planning and collision checking. It implements kinematics, robot models, and interfaces to several collision checkers. When reporting planning effort, we present both total time and energy used, as well as number of collision checks performed for comparison to other checkers. See the section on metrics below.

Open Motion Planning Library. We use the Open Motion Plan-

ning Library (OMPL) planning framework [34] to implement our planners and for baseline approaches which we compare against.

Metrics. Our primary metric is *total task effort* exerted to accomplish the task. For each run of our algorithms, we record this effort as measured in (a) time, in seconds, and (b) energy, in Joules. In order to present these data as fairly as possible, we record the split between planning and execution effort for each plan. We also report the difference between expected and actual execution effort for the solution path that may result from an approximate execution effort model (see Chapter 2.2).

To enable comparisons to other robots and collision checkers, we also report number of collision checks as a proxy for collision checking effort. We also report the number of triangles in each collision model.

While our approach terminates automatically given an the input λ parameter between planning and execution effort, we also compare against anytime approaches that return multiple improving solutions over time. We present plots of all of these results for comparison, and also report the data points that result from several a-priori time budgets.

Baseline Approaches. We propose to test our task planner against a suite of state-of-the-art approaches, including:

- Hierarchical task and motion planning in the now [13]. Note that we compare against HPN both interleaved and non-interleaved.
- The seqeutial task planner used by by the CMU team at the Darpa Robotics Challenge [3], based on the Constrained Bi-Directional RRT [5].
- Other hybrid symbolic/geometric planners.

For a fair comparison, we also select parameters for other approaches using a set of testing problems.

[5] D. Berenson, S. Srinivasa, D. Ferguson, and J. Kuffner. Manipulation planning on constraint manifolds. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2009

3.2 *Timeline*

See Table 3.1 for the timeline.

I'm not sure if integration of my OMPL planner is in the best interests of the DRC project. Will know more about this after the March 2015 DRC testbed meeting. In any case, I'd like to test against Trials data (and perhaps Finals data if it's available).

3.3 *Open-Source Software*

I propose to release the following open-source software packages as part of my thesis deliverables:

Topic	**Section**	**Questions**	**Deadline**
Guided Manipulation Planning at the DRC Trials [3]			February 2014 (ISER) (completed)
Comprehensive Multi-Root Planning [4]	2.5		October 2014 (ICRA) (completed)
Reduce, Reuse, Recycle: Multi-Set Planning [5]	2.3, 2.4	Q1, Q2	January 2015 (RSS) (in submission)
Proposal			March 2015
Multi-Set Planning for the DRC	2.4, 2.6	Q5, Q7, Q8	June 2015 (Humanoids)
HERB Experiments	2.4, 2.6	Q5, Q6, Q8	July-August 2015
The E^8-PRM: Optimizing Total Task Cost	2.1, 2.2	Q1, Q3, Q4	September 2015 (AAAI)
Thesis Writing			September-November 2015
Bored Robots: Hypothesized Conservative Volumes			October 2015 (ICRA)
Defense			December 2015

Table 3.1: Proposed Timeline.

- The E^8-PRM (OMPL).
- The multi-set decomposer (OpenRAVE+OMPL).
- The PROTEUS task planner.

4
Bibliography

[1] D. Berenson, S. Srinivasa, D. Ferguson, and J. Kuffner. Manipulation planning on constraint manifolds. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2009.

[2] R. Bohlin and E. Kavraki. Path planning using Lazy PRM. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 521–528 vol.1, 2000.

[3] C. Dellin, K. Strabala, G. C. Haynes, D. Stager, and S. Srinivasa. Guided manipulation planning at the DARPA Robotics Challenge trials. In *2014 International Symposium on Experimental Robotics (ISER 2014)*, June 2014.

[4] C. M. Dellin and S. S. Srinivasa. A general technique for fast comprehensive multi-root planning on graphs by coloring vertices and deferring edges. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015.

[5] C. M. Dellin and S. S. Srinivasa. Reduce, reuse, recycle: Exploiting C-space structure in similar problems with multi-set planning. In *Robotics: Science and Systems*, 2015, in submission.

[6] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, Aug. 2010.

[7] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, May 2015.

[8] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Ffrob: An efficient heuristic for task and motion planning. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2014.

[9] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.

[10] K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2493–2498, May 2010.

[11] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Robotics and Automation, 1997. Proceedings of the 1997 IEEE International Conference on*, volume 3, pages 2719–2726 vol.3, Apr. 1997.

[12] L. Jaillet and T. Simeon. A PRM-based motion planner for dynamically changing environments. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, pages 1606–1611 vol.2, Sept. 2004.

[13] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1470–1477, May 2011.

[14] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[15] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, Aug. 1996.

[16] S. Koenig, M. Likhachev, and D. Furcy. Lifelong Planning A*. *Artificial Intelligence Journal*, 2004.

[17] J. J. Kuffner and S. M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995–1001 vol.2, 2000.

[18] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[19] S. M. LaValle, M. S. Branicky, and S. R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):673–692, 2004.

[20] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 473–479, 1999.

[21] P. Leven and S. Hutchinson. Toward real-time path planning in changing environments. In *Proceedings, Workshop on the Algorithmic Foundations of Robotics (WAFR 'oo)*, 2000.

[22] T. Lozano-Pérez. Spatial planning: A configuration space approach. *Computers, IEEE Transactions on*, C-32(2):108–120, Feb. 1983.

[23] W. S. Newman and M. S. Branicky. Real-time configuration space transforms for obstacle avoidance. *The International Journal of Robotics Research*, 10(6):650–667, 1991.

[24] J. Pearl and J. H. Kim. Studies in semi-admissible heuristics. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-4(4):392–399, July 1982.

[25] M. Phillips, B. J. Cohen, S. Chitta, and M. Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, 2012.

[26] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, Massachusetts and London, England, 2006. Available at http://www.gaussianprocess.org/gpml/.

[27] W. Ruml and M. B. Do. Best-first utility-guided search. In *International Joint Conference on Artificial Intelligence*, pages 2378–2384, 2007.

[28] G. Sánchez-Ante and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *ISRR*, pages 403–417, 2001.

[29] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*, volume 9, pages 1–10, 2013.

[30] D. Shaharabani, O. Salzman, P. Agarwal, and D. Halperin. Sparsification of motion-planning roadmaps by edge contraction. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4098–4105, May 2013.

[31] L. Sint and D. de Champeaux. An improved bidirectional heuristic search algorithm. *J. ACM*, 24(2):177–191, Apr. 1977.

[32] S. Srinivasa, D. Berenson, M. Cakmak, A. Collet, M. Dogar, A. Dragan, R. Knepper, T. Niemueller, K. Strabala, M. Vande Weghe, and J. Ziegler. HERB 2.0: Lessons learned from developing a mobile manipulator for the home. *Proceedings of the IEEE*, 100(8):2410–2428, Aug. 2012.

[33] T. Stentz, H. Herman, A. Kelly, E. Meyhofer, G. C. Haynes, D. Stager, B. Zajac, J. A. Bagnell, J. Brindza, C. Dellin, M. George, J. Gonzalez-Mora, S. Hyde, M. Jones, M. Laverne, M. Likhachev, L. Lister, M. Powers, O. Ramos, J. Ray, D. Rice, J. Scheifflee, R. Sidki, S. Srinivasa, K. Strabala, J.-P. Tardif, J.-S. Valois, J. M. Vande Weghe, M. Wagner, and C. Wellington. CHIMP, the CMU Highly Intelligent Mobile Platform. *Submitted*, 2014.

[34] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, Dec. 2012. http://ompl.kavrakilab.org.

[35] P. H. Winston. *Artificial Intelligene*. Addison-Wesley Pub. Co., 1977.

[36] M. Zucker, R. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. Bagnell, and S. Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9–10):1164–1193, 2013.