Anytime Geometric Motion Planning on Large Dense Roadmaps

Shushman Choudhury

July 11, 2017

CMU-RI-TR-17-34

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Siddhartha S. Srinivasa, CMU RI (chair)
Michael Erdmann, CMU RI/CS
Stefanos Nikolaidis, CMU RI

Submitted in partial fulfillment of the requirements
for the degree of Master of Science.

Abstract

For real-world robotic systems, the ability to efficiently obtain motion plans of good quality over a diverse range of problems, is crucial. Large, dense motion-planning roadmaps can typically ensure the existence of such good quality solutions for most problems.

This thesis proposes an algorithmic framework for anytime planning on large, dense motion-planning roadmaps. The size of the roadmap graph creates difficulties for most existing approaches to graph-based planning algorithms. This is compounded by the specific challenges for certain motivating domains like manipulation, where collision checks and therefore edge evaluations are extremely expensive, and the configuration space is typically high-dimensional. We have two key insights to deal with these challenges.

Firstly, we frame the problem of anytime motion planning on roadmaps as one of searching for the shortest path over increasingly dense subgraphs of the entire roadmap graph. We study the space of all subgraphs of the roadmap graph, and consider densification strategies which traverse this space, selecting subgraphs along the way to be searched. We also analyse the behaviour of these strategies across the subgraph space, as well as the tradeoff between worst-case search effort and bounded suboptimality.

Secondly, we develop an anytime roadmap planning algorithm that is efficient with respect to collision checks for obtaining the first feasible path and successively shorter feasible paths. We use a model that computes the probability of unevaluated configurations being collision-free, and is updated with collision checks. Our algorithm, which we call POMP or Pareto-optimal Motion Planner, searches for paths which are Pareto-optimal in path length and collision likelihood. It gradually prioritizes path length, eventually finding the shortest feasible path.

We empirically evaluate both our ideas on a number of simulated cases, against contemporary motion planning algorithms, and show favourable performance over a range of problems.

Acknowledgments

There are a large number of people who have made my experience as a MS student enjoyable and enlightening, and I will try to do justice to some of them here. Of course I must begin by expressing my utmost gratitude to my advisor Siddhartha Srinivasa, who I have had the pleasure of working with and learning from for the better part of the last 3 years. As I have told Sidd often, he gave me my opportunity back when I had no idea what I was doing or what constituted research in robotics. From thinking about ideas rigorously, to implementing them intelligently, to expressing them enthusiastically - Sidd has played a pivotal role in shaping me into the graduate student I am today, and I do hope to continue to benefit from his guidance, even as I move on to other things.

I must also mention my other MS committee member, Mike Erdmann. I think it's safe to say that Mike and I had one of the highest levels of interaction between any of the MS students and their non-advisor committee members. Mike's Robomath course was very important in setting me up for grad school with a strong foundational base, but more than that it introduced me to someone who went on to be not just my committee member, but someone I consider a friend. I have had many fruitful discussions with him in his interesting office on the 9th floor of Gates, and I will miss them when I leave. For all his experience and expertise in many fundamental aspects of robotics, Mike has a humility and curiosity for ideas, no matter how seemingly trivial or simple, that is very inspiring.

It is with great pride that I call myself a member of the Personal Robotics Lab - because of its high quality research that spans several areas of robotics, because of how diverse and open and welcoming it is, and above all because of the remarkable individuals I have had the pleasure of interacting with there. I share an interesting personal dynamic with each of you, and you have made my work a much more pleasant and knowledgeable experience. A few special Thank Yous are in order - to Stefanos for being the coolest student committee member ever; to Gilwoo for being my academic partner for most of the second half of my MS; to Shervin for being my life guru; to Laura for

reminding me there are things other than work that matter; to Henny for lending me her ear whenever I asked; to Rachel for never failing to inspire me; to Jimmy for being a great minion and then peer; to Vinny for being my buddy; to Chris for teaching me most of what I know about the right way to implement things, to Mike2 for the same as Chris and also the Nick Cage jokes; to Oren for guiding me in many areas of my research; to Aditya for being my favourite punching bag and protege; and finally to Rosario for too many reasons to count.

Other than fellow lab members I am fortunate to have many wonderful friends whose presence has greatly increased the quality of my life. A big thank you to Dhruv, Xuning, Tim, Achal, Christine, Shohin, Rick, Devin, Reuben, Merritt, Jon, Anirudh, Vishal, Puneet, Paloma, David, Azaraksh, Logan, Cara, Rogerio, the Brians, Jen, Elena, and many more for all the guidance and the good times. A special thank you to Abhijeet for all the conversations and laughs, both here and in Korea. And to Doyel for making me the luckiest man.

Finally, whatever I am today is because I have had the privilege of being the 'baby' of the most loving family I could ever have asked for. To my parents and to my first, or rather, my zeroth friend, my brother Sanjiban, this, as everything else I have ever done, is for you.

Contents

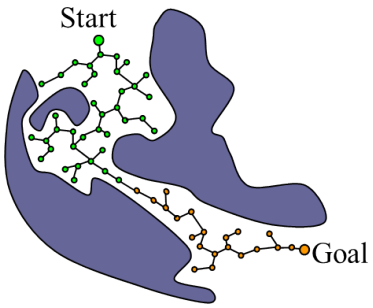
List of Figures

1
Introduction

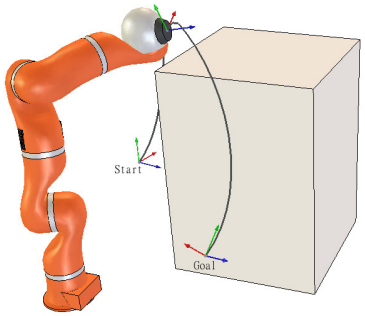
Motion planning is one of the fundamental aspects of robotics. Given some desired start and goal configuration of the robot, the motion planning problem seeks to obtain a sequence of discrete movements that satisfy the physical constraints of the robot and the environment and (maybe) additionally optimizes some objective function (see Fig. 1.1 for some examples). There are multiple reference texts devoted to motion planning [Latombe, 2012, LaValle, 2006] and we will not re-invent the wheel for the purpose of this thesis.

We focus on a specific sub-part of the motion planning problem, geometric motion planning on roadmaps. In *geometric* motion planning, we only consider the state-space constraints of the robot. All configurations in the plan must be within the state-space limits and must not put the robot in collision with the environment or with itself. The collision-free solution path to a geometric motion planning problem is typically not enough for execution in most robotic systems - it must subsequently be converted to a trajectory which also satisfies the dynamical constraints of the system.

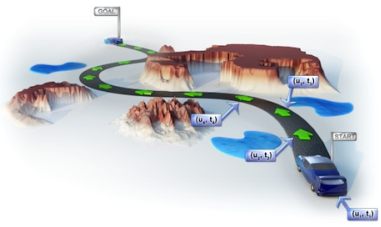
The term *roadmap* refers to one of the popular constructs to solve the geometric motion planning problem. Roadmap-based methods are an example of the *sampling-based* approach to motion planning, which samples points in the robot's configuration space rather than explicitly constructing a representation of the obstacles[Lozano-Perez, 1983], which typically cannot be done analytically for arbitrary configuration spaces. The feasibility of configurations is verified using a collision detection module, which is treated as a black box by the motion planning algorithm. A roadmap is a topological graph where the vertices are sampled configurations of the robot, and the edges are connections between vertices that can be created by some local planner (or just a straight line between them). Given start and goal configurations, the geometric motion planning problem is solved by connecting the start and goal to the roadmap and finding a feasible path on the roadmap between start and goal using some graph search algorithm [Dijkstra,



(a) `http://users.wpi.edu/~dberenson`



(b) `http://coppeliarobotics.com`



(c) `http://ompl.kavrakilab.org`

Figure 1.1: Three typical examples of motion planning problems - (a) a point robot in a 2D world (b) the end-effector of a manipulator and (c) an autonomous vehicle in an outdoor environment.

1959, Hart et al., 1968].

The basic idea behind roadmaps was introduced with the term *probabilistic roadmaps* (PRM) [Kavraki et al., 1996]. The PRM approach is to construct the roadmap graph in configuration space, remove all vertices and edges that are in collision in an offline *pre-processing* stage and then search for the shortest path between start and goal configurations in an online *query* stage. It is particularly well suited to multi-query settings where the environment does not change but multiple start-goal pairs are generated, for which the shortest feasible path needs to be found.

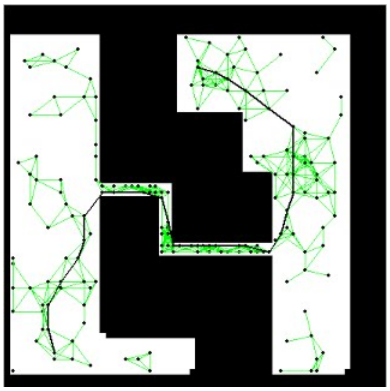


Figure 1.2: Courtesy `https://sites.google.com/site/moslemk`. An example of a probabilistic roadmap used to solve a 2D motion planning problem.

N.B - To avoid any confusion, please note that in this thesis we consider the LazyPRM [Bohlin and Kavraki, 2000] setting rather than the classical PRM setting in that we do not have a phase where we remove all edges and nodes in collision with the environment before beginning the plan. That would not be practical in the real-time case where we want a solution as soon as possible after observing the environment. Once we obtain an environment on which we want to solve some planning problem, we evaluate nodes and edges just-in-time (also called lazy) for that specific problem.

We ideally want a geometric motion planning algorithm to obtain *good quality paths over a wide range of motion planning problems*. What does that mean in terms of roadmaps? To be able to find any feasible path over a diverse set of problems, we would want the roadmaps to be *large*, with a very high number of vertices (samples) so as to cover the configuration space sufficiently. And to get the best quality paths possible with those vertices, we would want the roadmaps to be *dense* or even *complete*, i.e. with a potential edge between every pair of vertices.

The problem with large, dense roadmaps is that any shortest path algorithm that directly searches over the entire roadmap graph may be too computationally expensive to be practical in real-time. Therefore we consider *anytime motion planning*, which finds an initial solution quickly and uses its length (or some other relevant objective function) as a bound for future solutions.

This thesis proposes an algorithmic framework for anytime geometric motion planning on large, dense roadmaps.

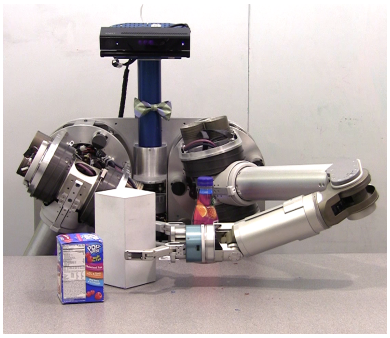


Figure 1.3: A (rather outdated) snapshot of the Personal Robotics Laboratory's robot HERB (Home Exploring Robot Butler).

Though our approach, algorithms and analyses are all completely general, in terms of applications we are particularly motivated by motion planning for manipulators, such as for HERB [Srinivasa et al., 2010], shown in Fig. 1.3.

1.1 Observations for Manipulation Planning

We make three important observations about manipulation planning that inform our general approach to the anytime motion planning problem, and the particular aspects and challenges to consider.

Observation 1: Manipulators have several degrees of freedom

Most typical manipulators have many degrees of freedom, one for each independently controlled joint. For instance, the Barrett WAM arm [Smith and Rooks, 2006] which is used by HERB as well as several other research platforms, has 7 active DOFs. The same is true for each arm of the PR2, excluding the additional DOF at the wrist. This makes the planning problem fairly high-dimensional. The number of samples needed to cover a space grows exponentially with the dimensionality of the problem. Therefore roadmaps for manipulation planning need to be especially large.

Observation 2: Collision Checks can be very expensive

For manipulators, collision checking typically involves processing a triangular mesh model of the manipulator as well as models of the obstacles in the simulator that is used, as shown in Fig. 1.4. The more articulated the robot, the more complex the mesh model, and the more computationally expensive the collision check. In the classical PRM approach, a significant fraction of time is spent on collision checking [Sánchez and Latombe, 2002], irrespective of the problem setting. This issue is further exacerbated when the collision checks are particularly expensive. On roadmaps, this is manifested as *expensive edge evaluations* because evaluating an edge involves checking several embedded configurations.

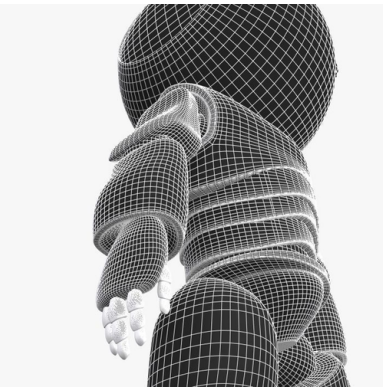


Figure 1.4: A triangular mesh model for an articulated robot can often be extremely complex. This makes collision checks, which need to process the entire mesh model in the worst case, very computationally expensive.

Observation 3: The tradeoff between execution time and planning time is crucial

Optimality and bounded sub-optimality are typically considered to be important properties for motion planning algorithms. However, manipulation planning on large dense roadmaps is perhaps the most appropriate example where the execution speedup obtained by the shortest (or approximately shortest) path may be negated by the extra planning time required to find it. For a manipulation planning problem, the configuration space of interest is a subset of the swath of space that the manipulator can carve out. The difference between the shortest and any non-shortest path is much smaller for a manipulation planning problem than for say, a mobile robot or aerial vehicle

planning problem traveling large distances. Therefore the tradeoff between execution time (represented by the path length) and planning time (represented by the effort to find a path) becomes particularly important [Dellin, 2016].

1.2 Key Ideas

Our approach to anytime geometric motion planning on roadmaps is based on two key ideas. They are motivated to some extent by the specific challenges for manipulation planning but as such are completely general and can be applied to any geometric motion planning setting. We outline them here.

Idea 1: Anytime planning as a search over a sequence of subgraphs

Our key insight for solving the anytime planning problem in large, dense roadmaps is to provide existing path-planning algorithms with a sequence of increasingly dense subgraphs of the roadmap graph, using some *densification strategy* [Choudhury et al., 2016c]. At each iteration, we run a shortest-path algorithm on the current subgraph to obtain an increasingly tighter approximation of the true shortest path (Fig. 1).

Existing approaches to anytime planning on graphs [Likhachev et al., 2004, 2005, van den Berg et al., 2011] typically modify the objective function in some way, thereby sacrificing optimality for a quicker solution. Some of those approaches guarantee bounded sub-optimality instead. This is quite reasonable to do when the graph that is being searched over is of a moderate size. But a single search with a modified objective function has a worst case complexity of $O(|V| \log |V| + |E|) \equiv O(N^2)$ which may be unacceptable for real-time applications, when dealing with large, dense roadmaps. Also, there is no formal guarantee that these approaches will decrease search time and they may still search all edges of a given graph [Wilt and Ruml, 2012].

Instead, we consider explicitly solving sub-problems of the entire problem, and gradually increasing the size of each sub-problem until we solve the original problem. On a roadmap graph, this naturally maps to searching over increasingly dense subgraphs of the entire roadmap graph. The nature of the subgraph may be able to bound the sub-optimality of the solutions obtained, and more importantly, the tradeoff between the worst-case search effort and bounded sub-optimality. This is analysed in more detail in chapter 4.

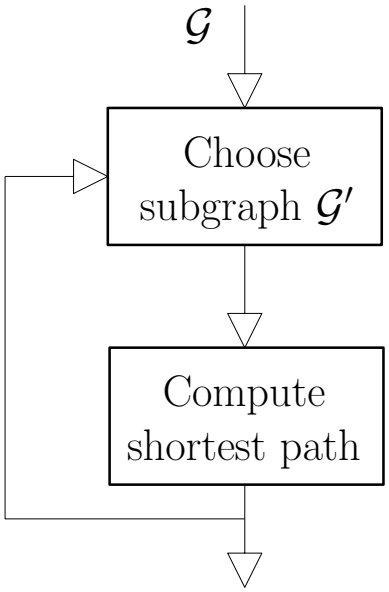


Figure 1.5: The idea behind densification is to search for the shortest path over a sequence of increasingly dense subgraphs of the entire roadmap graph.

Idea 2: Searching over configuration space beliefs

Performing a collision check provides exact information but is computationally expensive. We assume that the roadmaps we search over are embedded in a continuous ambient space, where nearby points tend to share the same collision state. Therefore, we can use a model of the world to estimate the probability of unevaluated configurations to be free or in collision. We ensure that updating and querying the model is inexpensive. Searching for paths based on collision probability does not guarantee optimality, but may speed up the computation of some feasible path. Furthermore, we search for paths that are pareto-optimal in collision probability and path length, adjusting the tradeoff between them to eventually obtain the shortest path on the graph we are currently searching [Choudhury et al., 2016b]. See Fig. 2. We propose using this search technique, which we will elaborate upon in chapter 5, for *searching each individual roadmap subgraph* generated by the densification strategy.

We want to try and minimize the number of collision checks while searching for feasible paths in a roadmap. A number of previous works have tried to address this - using the probability of collision as a heuristic to guide the search over paths to obtain a feasible path[Nielsen and Kavraki, 2000]; lazily and optimistically searching for the shortest path in a roadmap [Bohlin and Kavraki, 2000, Dellin and Srinivasa, 2016]; probabilistically modelling obstacle locations to combine exploration and exploitation in a hybrid approach [Knepper and Mason, 2012]; using collision probabilities learned from previous instances to modify the roadmap cost function, and filter out unlikely configurations [Pan et al., 2013].

Past work lacks, however, a way to connect the two problems of finding a feasible path quickly and finding the shortest feasible path in the roadmap. Our key insight is that this can be achieved by balancing the probability of collision and the length of a path in the objective. We show that this behaviour is approximately equivalent to searching for paths of minimum expected cost, with a gradually decreasing penalty of being in collision.

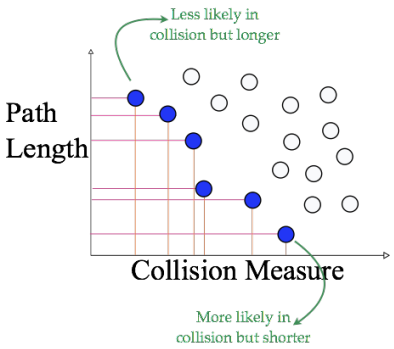


Figure 1.6: We reason about the tradeoff between path length and collision measure (related to probability) for candidate paths. Specifically, we efficiently search for paths that are pareto-optimal in the two quantities and select them for lazy evaluation.

1.3 Contributions

This thesis proposes an algorithmic framework for anytime geometric motion planning in large, dense roadmaps that is particularly well suited for the difficult high-dimensional setting of manipulation. Based on our key ideas, the following are our contributions:

- We motivate the use of a large, dense roadmap for motion planning. In particular we propose using the same roadmap constructed

for a particular robot's configuration space over a wide range of problems that the robot may be required to solve. This lets us take advantage of the structure embedded in the roadmap, as well as various preprocessing benefits of using the same roadmap repeatedly (chapter 3).

- We frame the problem of anytime planning on a roadmap as searching over a sequence of subgraphs of the entire roadmap graph. We discuss several densification strategies to generate this sequence of subgraphs. For the specific case where the samples are generated from a low-dispersion (Sec. 2.2) deterministic sequence, we analyse the tradeoff between effort and bounded sub-optimality (chapter 4, Choudhury et al. [2016c])
- We present an anytime planning algorithm (for a reasonably sized roadmap) that maintains a belief over configuration space collision probabilities and efficiently searches for paths that are pareto-optimal in collision probability and path length (chapter 5, Choudhury et al. [2016b]).

We conclude in chapter 6 by discussing an overall perspective of our approach and its limitations, as well as interesting questions for future research.

2
Background

We briefly outline the various concepts and relevant prior works that this thesis is based on.

2.1 Motion Planning

As stated in chapter 1, motion planning is a fundamental problem in robotics. There are comprehensive texts on the subject [LaValle, 2006, Latombe, 2012] that discuss the various developments in this field, from the origins in the classical piano mover's problem [Schwartz and Sharir, 1983]. We are specifically interested in *geometric motion planning* which focuses on finding a valid path between start and goal configurations that is not in collision anywhere along the path [Lozano-Pérez and Wesley, 1979].

2.1.1 Sampling-based motion planning

Sampling-based planning approaches build a graph, or a roadmap, in the configuration space, where vertices are configurations and edges are local paths connecting configurations. A path is then found by traversing this roadmap while checking if the vertices and edges are collision free. Initial algorithms such as PRM [Kavraki et al., 1996] and RRT [LaValle and Kuffner, 1999] were concerned with finding *a feasible* solution. However, in recent years, there has been growing interest in finding high-quality solutions. Karaman and Frazzoli [Karaman and Frazzoli, 2011] introduced variants of the PRM and RRT algorithms, called PRM* and RRT*, respectively and proved that, asymptotically, the solution obtained by these algorithms converges to the optimal solution. However, the running times of these algorithms are often significantly higher than their non-optimal counterparts. Thus, subsequent algorithms have been suggested to increase the rate of convergence to high-quality solutions. They use different approaches such as lazy computation [Bohlin and Kavraki,

2000, Janson et al., 2015b, Salzman and Halperin, 2015, Dellin and Srinivasa, 2016], informed sampling [Gammell et al., 2014a], pruning vertices [Gammell et al., 2014b], relaxing optimality [Salzman and Halperin, 2016], exploiting local information [Choudhury et al., 2016a] and lifelong planning together with heuristics [Koenig et al., 2004].

2.1.2 Efficient path-planning algorithms

We are interested in path-planning algorithms that attempt to reduce the amount of computationally expensive edge expansions performed in a search. This is typically done using heuristics such as for A* [Hart et al., 1968], for Iterative Deepening A* [Korf, 1985] and for Lazy Weighted A* [Cohen et al., 2014]. Some of these algorithms, such as Lifelong Planning A* [Koenig et al., 2004] allow recomputing the shortest path in an efficient manner when the graph undergoes changes. *Anytime* variants of A* such as Anytime Repairing A* [Likhachev et al., 2004] and Anytime Nonparametric A* [van den Berg et al., 2011] efficiently run a succession of A* searches, each with an inflated heuristic. This potentially obtains a fast approximation and refines its quality as time permits. However, there is no formal guarantee that these approaches will decrease search time and they may still search all edges of a given graph [Wilt and Ruml, 2012]. For a unifying formalism of such algorithms relevant to explicit roadmaps, in settings where edge evaluations are expensive, and for additional references, see [Dellin and Srinivasa, 2016].

2.1.3 Finite-time properties of sampling-based algorithms

Extensive analysis has been done on *asymptotic* properties of sampling-based algorithms, i.e. properties such as connectivity and optimality when the number of samples tends to infinity [Kavraki et al., 1998, Karaman and Frazzoli, 2011].

We are interested in bounding the quality of a solution obtained using a *fixed* roadmap for a finite number of samples. When the samples are generated from a *deterministic* sequence, [Janson et al., 2015a, Thm2] give a closed-form solution bounding the quality of the solution of a PRM whose roadmap is an r-disk graph. The bound is a function of r, the number of vertices n and the *dispersion* of the set of points used. (See for an exact definition of dispersion and for the exact bound given by [Janson et al., 2015a]).

Similar bounds have been provided by [Dobson et al., 2015] when randomly sampled i.i.d points are used. Specifically, they consider a PRM whose roadmap is an r-disk graph for a *specific* radius $r = c \cdot (\log n/n)^{1/d}$ where n is the number of points, d is the dimension and c is some constant. They then give a bound on the probability that

the quality of the solution will be larger than a given threshold.

2.2 Dispersion

The *dispersion* $D_n(\mathcal{S})$ of a sequence $\mathcal{S}$ is defined as

$$D_n(\mathcal{S}) = \sup_{x \in \mathcal{X}} \min_{s \in \mathcal{S}} \rho(x, s)$$

Intuitively, it can be thought of as the radius of the largest empty ball (by some metric) that can be drawn around any point in the space $\mathcal{X}$ without intersecting any point of $\mathcal{S}$. A lower dispersion implies a better *coverage* of the space by the points in $\mathcal{S}$. When $\mathcal{X}$ is the d-dimensional Euclidean space and ρ is the Euclidean distance, deterministic sequences with dispersion of order $O(n^{-1/d})$ exist. A simple example is a set of points lying on grid or a lattice.

Other low-dispersion deterministic sequences exist which also have low *discrepancy*, i.e. they appear to be random for many purposes. Specifically, the discrepancy of a set of points is the deviation of the set from the uniform random distribution. The corresponding mathematical definition of discrepancy is

$$\mathbb{D}_n(\mathcal{S}, \mathcal{R}) = \sup_{R \in \mathcal{R}} \{ \left| \frac{|\mathcal{S} \cap R|}{n} - \frac{\mu(R)}{\mu(\mathcal{X})} \right| \}$$

where $\mathcal{R}$ is a collection of subsets of $\mathcal{X}$ called the *range space*. It is typically taken to be the set of all axis-aligned rectangular subsets. The μ operator refers to the *Lebesgue Measure* or the generalized volume of the operand [Bartle, 2014].

One such example is the *Halton sequence* [Halton, 1960]. We will use them extensively for our analysis because they have been studied in the context of deterministic motion planning [Janson et al., 2015a, Branicky et al., 2001]. Halton sequences are constructed by taking d prime numbers, called generators, one for each dimension. Each generator g induces a sequence, called a Van der Corput sequence. The k'th element of the Halton sequence is then constructed by taking the k'th element of each of the d Van der Corput sequences. For Halton sequences, tight bounds on dispersion exist. Specifically, $D_n(\mathcal{S}) \leq p_d \cdot n^{-1/d}$ where $p_d \approx d \log d$ is the d^{th} prime number. Subsequently in this paper, we will use D_n (and not $D_n(\mathcal{S})$) to denote the dispersion of the first n points of $\mathcal{S}$.

Previous work bounds the length of the shortest path computed over an r-disk roadmap constructed using a low-dispersion deterministic sequence [Janson et al., 2015a, Thm2]. Specifically, given start and target vertices, consider all paths Γ connecting them which have δ-clearance for some δ. Note that a path has clearance δ if every point on the path is at a distance of at least δ away from every obstacle. Set

$\delta_{\max}$ to be the maximal clearance over all such δ. If $\delta_{\max} > 0$, then for all $0 < \delta \leq \delta_{\max}$ set $c^*(\delta)$ to be the cost of the shortest path in Γ with δ-clearance. Let $c(\ell, r)$ be the length of the path returned by a shortest-path algorithm on $\mathcal{G}(\ell, r)$ with $\mathcal{S}(\ell)$ having dispersion D_ℓ. For $2D_\ell < r < \delta$, we have that

$$c(\ell, r) \leq \left(1 + \frac{2D_\ell}{r - 2D_\ell}\right) \cdot c^*(\delta). \tag{2.1}$$

Notably, for n random i.i.d. points, the lower bound on the dispersion is $O\left((\log n/n)^{1/d}\right)$ [Niederreiter, 1992] which is strictly larger than for deterministic samples.

For domains other than the unit hypercube, the insights from the analysis will generally hold. However, the dispersion bounds may become far more complicated depending on the domain, and the distance metric would need to be scaled accordingly. This may result in the quantitative bounds being difficult to deduce analytically.

2.3 Configuration Space Belief

The probability of collision of a path is derived from an approximate model of the configuration space of the robot. Since we explicitly seek to minimize collision checks, we build up an incremental model using data from previous collision tests, instead of sampling several, potentially irrelevant configurations apriori. This idea has been studied [Burns and Brock, 2003, 2005a] in similar contexts. Furthermore, the evolving probabilistic model can be used to guide future searches towards likely free regions. Previous work has analyzed and utilized this exploration-exploitation paradigm for faster motion planning [Rickert et al., 2008, Knepper and Mason, 2012, Pan et al., 2013, Arslan and Tsiotras, 2015].

2.4 Multi-objective Path Search

We reason about both the path length and the probability of collision for an individual candidate path. This analysis is built upon a considerable body of work dealing with bi-criteria path problems. Early work has conducted a systematic study of these problems [Hansen, 1980], and devised methods to obtain non-dominated or Pareto optimal paths [Climaco and Martins, 1982]. It has also provided insights directly relevant to shortest path problems for robots[Mitchell and Sastry, 2003].

3
Problem Definition

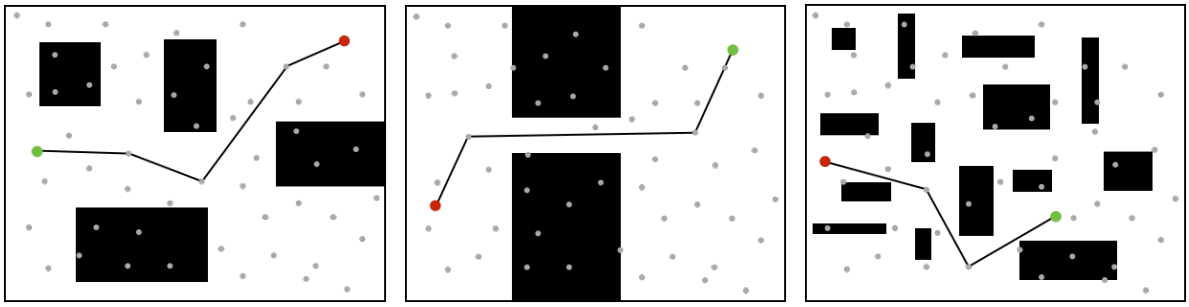
Let $\mathcal{X}$ denote a d-dimensional C-space, $\mathcal{X}_{\text{free}}$ the collision-free portion of $\mathcal{X}$, $\mathcal{X}_{\text{obs}} = \mathcal{X} \setminus \mathcal{X}_{\text{free}}$ its complement and let $\zeta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be some distance metric. For simplicity, we assume that $\mathcal{X} = [0,1]^d$ and that ζ is the Euclidean norm. Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be some sequence of points where $s_\ell \in \mathcal{X}$ for some integer $n \in \mathbb{N}$ and denote by $\mathcal{S}(\ell)$ the first ℓ elements of $\mathcal{S}$. We define the r-disk graph $\mathcal{G}(\ell, r) = (V_\ell, E_{\ell,r})$ where $V_\ell = \mathcal{S}(\ell)$, $E_{\ell,r} = \{(u,v) \mid u, v \in V_\ell \text{ and } \zeta(u,v) \leq r\}$ and each edge (u,v) is of length $\zeta(u,v)$. See [Karaman and Frazzoli, 2011, Solovey et al., 2016] for various properties of such graphs in the context of motion planning. Our definition assumes that $\mathcal{G}$ is embedded in $\mathcal{X}$. Set $\mathcal{G} = \mathcal{G}(n, \sqrt{d})$, namely, the complete[1] graph defined over $\mathcal{S}$. In a complete graph there is an edge between every pair of vertices.

[1] Using a radius of $\sqrt{d}$ ensures that every two points will be connected due to the assumption that $\mathcal{X} = [0,1]^d$ and that ζ is Euclidean.

We propose using the same roadmap (with online, lazy evaluations for each specific environment) over a range of problem instances. As mentioned in chapter 1, we cannot do the pre-processing in the sense of the classical PRM approach due to real-time constraints. However, using a fixed roadmap structure allows us to do some environment-agnostic preprocessing that other classes of approaches like tree-growing [Kuffner and LaValle, 2000] or trajectory optimization [Ratliff et al., 2009] cannot do.

We can pre-compute all the nearest neighbors for each of the vertices in the roadmap, and filter out the configurations in self-collision, thereby requiring us to only check for robot-environment collisions during planning. Previous work has shown that both the computation of nearest neighbours [Kleinbort et al., 2016] and detecting self-collision [Srinivasa et al., 2016] are expensive components of motion planning algorithms.

As we have motivated earlier in chapter 1, we would need to use *large dense* motion-planning roadmaps to get good quality solutions over a wide range of problems. An illustration of why this is necessary is shown with some $\mathbb{R}^2$ problems in Fig. 3.1. For ease of analysis we



(a) (b) (c)

Figure 3.1: Consider the range of toy problems shown here : (a) A few obstacles with large gaps between them (b) A single narrow passage (c) Several obstacles with narrow gaps between them. A large dense roadmap is necessary to find feasible paths of good quality over such a diverse set of problems.

assume that the roadmap is complete, but our densification strategies and analysis can be extended to *dense* roadmaps that are not complete.

To save computation, although we create the graph explicitly, we do not evaluate it apriori, so we do not know if any of its vertices or edges are in C_{free} or C_{obs}. This setup is similar to that of the LazyPRM [Bohlin and Kavraki, 2000], which then searches over the graph optimistically until it finds the shortest feasible path.

A query $\mathcal{Q}$ is a scenario with start and target configurations. Let the start and target configurations be s_1 and s_2, respectively. The obstacles induce a mapping $\mathcal{M} : \mathcal{X} \to \{\mathcal{X}_{\text{free}}, \mathcal{X}_{\text{obs}}\}$ called a *collision detector* which checks if a configuration or edge is collision-free or not. Typically, edges are checked by densely sampling along the edge, and performing expensive collision checks for each sampled configuration, hence the term *expensive edge evaluation.* A feasible path is denoted by $\gamma : [0,1] \to \mathcal{X}_{\text{free}}$ where $\gamma[0] = s_1$ and $\gamma[1] = s_2$. In terms of the graph, a path is *feasible* if every included edge is in $\mathcal{X}_{\text{free}}$. Slightly abusing this notation, set $\gamma(\mathcal{G}(\ell, r))$ to be the shortest collision-free path from s_1 to s_2 that can be computed in $\mathcal{G}(\ell, r)$, its clearance as $\delta(\mathcal{G}(\ell, r))$ and denote by $\gamma^* = \gamma(\mathcal{G})$ and $\delta^* = \delta(\mathcal{G})$ the shortest path and its clearance that can be computed in $\mathcal{G}$, respectively.

Our problem calls for finding a sequence of increasingly shorter feasible paths $\gamma_0, \gamma_1 \ldots$ in $\mathcal{G}$, converging to γ^*. We assume that $n = |\mathcal{S}|$ is sufficiently large, and the roadmap covers the space well enough so that for any reasonable set of obstacles, there are multiple feasible paths to be obtained between start and goal. Therefore, we do not consider a case where the entire roadmap is invalidated by obstacles. The large value of n makes any path-finding algorithm that directly searches $\mathcal{G}$, thereby performing $O(n^2)$ calls to the collision-detector, too time-consuming to be practical.

4
Densification

(*This chapter is based on work presented in [Choudhury et al., 2016c].*) As mentioned in chapter 3, a *densification strategy* is a means to traverse the space of r-disk subgraphs of the entire roadmap graph, selecting subgraphs along the way to be searched, all the way to the fully connected roadmap. In this chapter, we discuss our general approach of searching over the space of all (r-disk) subgraphs of $\mathcal{G}$.

We start by characterizing the boundaries and different regions of this space. Subsequently, we introduce two densification strategies—edge batching and vertex batching. As we will see, these two are complementary in nature, which motivates our third strategy, which we call hybrid batching.

4.1 The space of subgraphs

To perform an anytime search over $\mathcal{G}$, given the collision detector $\mathcal{M}$, we iteratively search a sequence of graphs $\mathcal{G}_0(n_0, r_0) \subseteq \mathcal{G}_1(n_1, r_1) \subseteq \ldots \subseteq \mathcal{G}_m(n_m, r_m) = \mathcal{G}$. If no feasible path exists in the subgraph, we move on to the next subgraph in the sequence, which is more likely to have a feasible path.

We use an incremental path-planning algorithm that allows us to efficiently recompute shortest paths. Our problem setting of increasingly dense subgraphs is particularly amenable to such algorithms. However, any alternative shortest-path algorithm may be used. We emphasize again that we focus on the meta-algorithm of choosing which subgraphs to search. Further details on the implementation of these approaches are provided in Sec. 4.4.

Fig. 4.1 depicts the set of possible graphs $\mathcal{G}(\ell, r)$ for all choices of $0 < \ell \leq n$ and $0 < r \leq \sqrt{d}$. Specifically, the graph depicts $|E_{\ell,r}|$ as a function of $|V_\ell|$. We discuss it in detail to motivate our approach for solving the problem of anytime planning on large, dense roadmaps and the specific sequence of subgraphs we use. First, consider the curves that define the boundary of all possible graphs: The vertical

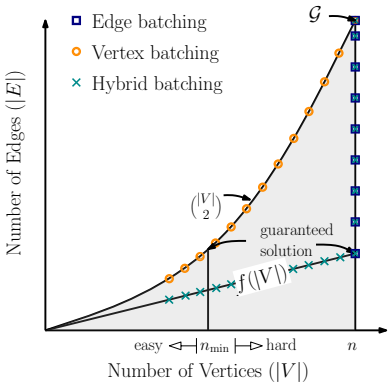


Figure 4.1: Our meta algorithm leverages existing path-planning algorithms and provides them with a sequence of subgraphs. To do so we consider densification strategies for traversing the space of r-disk subgraphs of the roadmap $\mathcal{G}$. , The x-axis and the y-axis represent the number of vertices and the number of edges (induced by r) of the subgraph, respectively. A particular subgraph is defined by a point in this space. *Edge batching* searches over all samples and adds edges according to an increasing radius of connectivity. *Vertex batching* searches over complete subgraphs induced by progressively larger subsets of vertices. *Hybrid batching* uses the minimal connection radius $f(|V|)$ to ensure connectivity until it reaches $|V| = n$ and then proceeds like edge batching.

line $|V| = n$ corresponds to subgraphs defined over the entire set of vertices, where batches of edges are added as r increases. The parabolic arc $|E| = |V| \cdot (|V| - 1)/2$, corresponds to complete subgraphs defined over increasingly larger sets of vertices.

Recall that we wish to approximate the shortest path γ^* which has some minimal clearance δ^*. Given a specific graph, to ensure that a path that approximates γ^* is found, two conditions should be met: (i) The graph includes some minimal number $n_{\min}$ of vertices. The exact value of $n_{\min}$ will be a function of the dispersion $D_{n_{\min}}$ of the sequence $\mathcal{S}$ and the clearance δ^*. (ii) A minimal connection radius r_0 is used to ensure that the graph is connected. Its value will depend on the sequence $\mathcal{S}$ (and not on δ^*).

In Fig. 4.2, requirement (i) induces a vertical line at $|V| = n_{\min}$. Any point to the left of this line corresponds to a graph with too few vertices to prove any guarantee that a solution will be found. We call this the *vertex-starvation* region. Requirement (ii) induces a curve $f(|V|)$ such that any point below this curve corresponds to a graph which may be disconnected. We call this the *edge-starvation* region. The exact form of the curve depends on the sequence $\mathcal{S}$ that is used. Any point outside the starvation regions represents a graph $\mathcal{G}(\ell, r)$ such that the length of $\gamma(\mathcal{G}(\ell, r))$ may be bounded. For a discussion on specific bounds, see Sec. 4.3.1

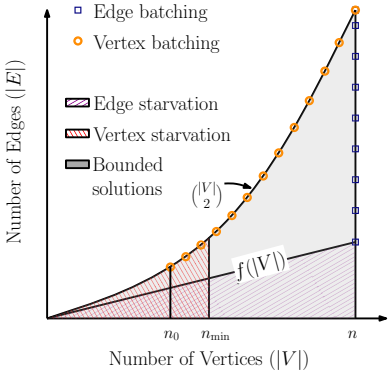


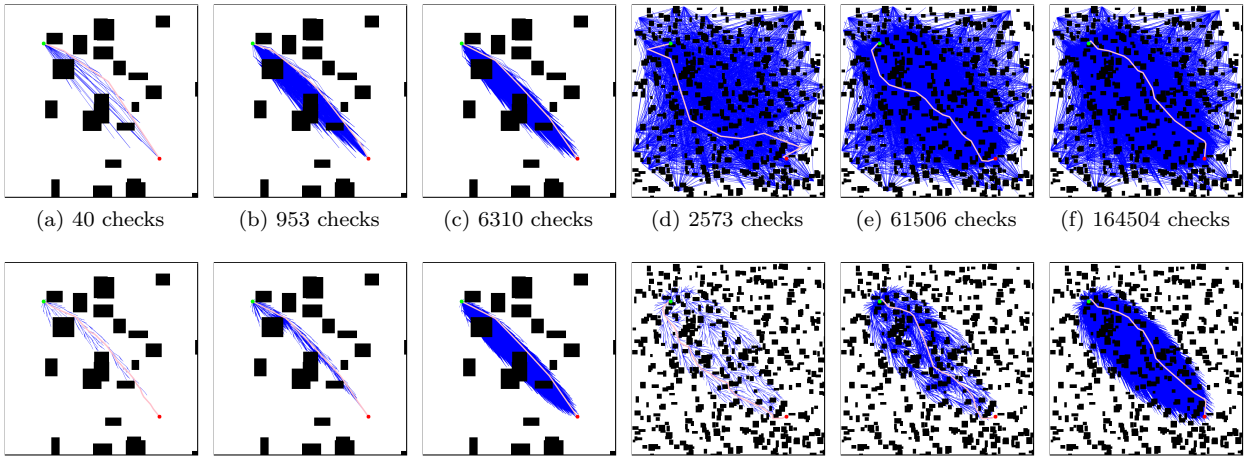
Figure 4.2: *Vertex Starvation* happens in the region with too few vertices to ensure a solution, even for a fully connected subgraph. *Edge Starvation* happens in the region where the radius r is too low to guarantee connectivity.

4.2 Densification Strategies

Our goal is to search increasingly dense subgraphs of $\mathcal{G}$. This corresponds to a sequence of points on the space of subgraphs (Fig. 4.2) that ends at the upper right corner of the space. We discuss three general densification strategies, and defer the discussion on the choice of parameters used for each strategy to Sec. 4.4

4.2.1 Edge Batching

All subgraphs include the complete set of vertices $\mathcal{S}$ and the edges are incrementally added via an increasing connection radius. Specifically, $\forall i\ n_i = n$ and $r_i = \eta_e r_{i-1}$ where $\eta_e > 1$ and r_0 is some small initial radius. Here, we choose $r_0 = O(f(n))$, where f is the *edge-starvation* boundary curve defined previously. It defines the minimal radius to ensure connectivity (in the asymptotic case) using r-disk graphs. Specifically, $f(n) = O\left(n^{-1/d}\right)$ for low-dispersion deterministic sequences and $f(n) = O\left((\log n/n)^{-1/d}\right)$ for random i.i.d sequences [Janson et al., 2015a, Karaman and Frazzoli, 2011, Salzman et al., 2016]. Using Fig. 4.2, this induces a sequence of points along the vertical line at $|V| = n$ starting from $|E| = O(n^2 r_0^d)$ and ending at



(a) 40 checks (b) 953 checks (c) 6310 checks (d) 2573 checks (e) 61506 checks (f) 164504 checks

(g) 206 checks (h) 676 checks (i) 15099 checks (j) 1390 checks (k) 4687 checks (l) 78546 checks

Figure 4.3: Visualizations of vertex batching (upper row) and edge batching (lower row) on easy (left pane) and hard (right pane) $\mathbb{R}^2$ problems respectively. The same set of samples $\mathcal{S}$ is used in each case. For easy problems, vertex batching finds the first solution quickly with a sparse set of initial samples. Additional heuristics hereafter help it converge to the optimum with fewer edge evaluations than edge batching. The harder problem has 10× more obstacles and lower average obstacle gaps. Therefore, both vertex and edge batching require more edge evaluations for finding feasible solutions and the shortest path. In particular, vertex batching requires multiple iterations to find its first solution, while edge batching still does so on its first search, albeit with more collision checks than for the easy problem. Note that the coverage of collision checks only appears similar at the end due to resolution limits for visualization.

$|E| = O(n^2)$.

4.2.2 Vertex Batching

In this variant, all subgraphs are complete graphs defined over increasing subsets of the complete set of vertices $\mathcal{S}$. Specifically $\forall i\ r_i = r_{\max} = \sqrt{d}$, $n_i = \eta_v n_{i-1}$ where $\eta_v > 1$ and the base term n_0 is some small number of vertices. Because we have no priors about the obstacle density or distribution, the chosen n_0 is a constant and does not vary due to n or due to the volume of $\mathcal{X}_{\text{obs}}$. Using Fig. 4.2, this induces a sequence of points along the parabolic arc $|E| = |V| \cdot (|V| - 1)/2$ starting from $|V| = n_0$ and ending at $|V| = n$. The vertices are chosen in the same order with which they are generated by $\mathcal{S}$. So, $\mathcal{G}_0$ has the first n_0 samples of $\mathcal{S}$, and so on.

Intuitively, the relative performance of these densification strategies depends on problem hardness. We use the clearance of the shortest path, δ^*, to represent the hardness of the problem. This, in turn, defines $n_{\min}$ which bounds the vertex-starvation region. Specifically we say that a problem is easy (resp. hard) when $\delta^* \approx \sqrt{d}$ (resp. $\delta^* \approx \Omega(D_n(\mathcal{S}))$). For easy problems, with larger gaps between obstacles, where $\delta^* \approx O(\sqrt{d})$, vertex batching can find a solution quickly with fewer samples and long edges, thereby restricting the work done for future searches. In contrast, assuming that $n > n_{\min}$, edge batching will find a solution on the first iteration but the time to do so may be far greater than for vertex batching because the number of samples is so large. For hard problems where $\delta^* \approx O(D_n)$, vertex batching may require multiple iterations until the number of samples it uses is large enough and it is out of the vertex-starvation region. Each of these searches would exhaust the fully connected subgraph before

terminating. This cumulative effort is expected to exceed that required by edge batching for the same problem, which is expected to find a feasible albeit sub-optimal path on the first search. A visual depiction of this intuition is given in Fig. 4.4. Since we are focused on problems with expensive edge evaluations, we treat work due to edge evaluations as a reasonable approximation of the total work done by the search. An empirical example of this is shown in Fig. 4.3.

4.2.3 Hybrid Batching

Vertex and edge batching exhibit generally complementary properties for problems with varying difficulty. Yet, when a query $\mathcal{Q}$ is given, the hardness of the problem is not known a-priori. In this section we propose a hybrid approach that exhibits favourable properties, regardless of the hardness of the problem.

This hybrid batching strategy commences by searching over a graph $\mathcal{G}(n_0, r_0)$ where n_0 is the same as for vertex batching and $r_0 = O(f(n_0))$. As long as $n_i < n$, the next batch has $n_{i+1} = \eta_v n_i$ and $r_i = O(f(n_i))$. When $n_i = n$ (and $r_i = O(f(n))$), all subsequent batches are similar to edge batching, i.e., $r_{i+1} = \eta_e r_i$ (and $n_{i+1} = n$).

This can be visualized on the space of subgraphs as sampling along the curve $f(|V|)$ from $|V| = n_0$ until $f(|V|)$ intersects $|V| = n$ and then sampling along the vertical line $|V| = n$. See Fig. 4.1 and Fig. 4.4 for a mental picture. As we will see in our experiments, hybrid batching typically performs comparably (in terms of path quality) to vertex batching on easy problems and to edge batching on hard problems.

The intuition behind this is straightforward. If the problem is easy, then hybrid batching finds a feasible solution early on, typically when the number of vertices is similar to that needed by vertex batching for a feasible solution. Thus, the work would be far less than that for edge batching. On the other hand, if the problem is hard, then hybrid batching would have to get much closer to the $|V| = n$ line before the dispersion becomes low enough to find a solution. However, it would not involve as much work as for vertex batching, because the radius decreases when the number of vertices increases, unlike vertex batching which uses $r_i = \sqrt{d}$ for every iteration i.

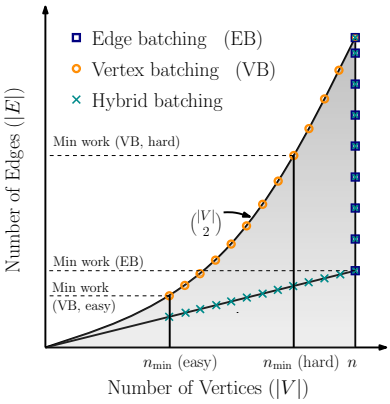


Figure 4.4: A visualization of the work required by our densification strategies as a function of the problem's hardness. Here work is measured as the number of edges evaluated. This is visualized using the gradient shading where light gray (resp. dark grey) depicts a small (resp. large) amount of work. Assuming $n > n_{\min}$, the amount of work required by edge batching remains the same regardless of problem difficulty. For vertex batching the amount of work required depends on the hardness of the problem. Here we visualize an easy and a hard problem using $n_{\min}$ (easy) and $n_{\min}$ (hard), respectively.

4.3 Analysis for Halton Sequences

In this section we consider the space of subgraphs and the densification strategies that we introduced in Sec. 4.2 for the specific case that $\mathcal{S}$ is a Halton sequence (Sec. 2.2). We start by describing the boundaries of the starvation regions. We then continue by simulating

the bound on the quality of the solution obtained as a function of the work done for each of our strategies.

Recall that the since we are considering the unit hypercube $[0,1]^d$, then $\delta_{\max} < \sqrt{d}$. We use (Eq. 2.1) to first obtain bounds on the vertex and edge starvation regions, and subsequently analyze the tradeoff between work and solution quality for vertex and edge batching.

4.3.1 Starvation region bounds

To bound the vertex starvation region we wish to find $n_{\min}$ after which bounded sub-optimality can be guaranteed to find the first solution. Note that δ^* is the clearance of the shortest path $\gamma*$ in $\mathcal{G}$ connecting s_1 and s_2, that p_d denotes the d^{th} prime and $D_n \leq p_d/n^{1/d}$ for Halton sequences. For (Eq. 2.1) to hold we require that $2D_{n_{\min}} < \delta^*$. Thus,

$$2D_{n_{\min}} < \delta^* \Rightarrow 2\frac{p_d}{n_{\min}^{1/d}} < \delta^* \Rightarrow n_{\min} > \left(\frac{2p_d}{\delta^*}\right)^d \tag{4.1}$$

Indeed, one can see that as the problem becomes harder (namely, δ^* decreases), $n_{\min}$ and the entire vertex-starvation region grows.

We now show that for Halton sequences, the edge-starvation region has a linear boundary, i.e. $f(|V|) = O(|V|)$. Using (Eq. 2.1) we have that the minimal radius $r_{\min}(|V|)$ required for a graph with $|V|$ vertices is

$$r_{\min}(|V|) > 2D_{|V|} \Rightarrow r_{\min}(|V|) > \frac{2p_d}{(|V|)^{1/d}}. \tag{4.2}$$

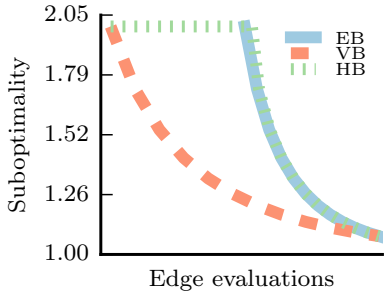
For any r-disk graph $\mathcal{G}(\ell, r)$, the number of edges is $|E_{\ell,r}| = O\left(\ell^2 \cdot r^d\right)$. In our case,

$$f(|V|) = O\left(|V|^2 \cdot r_{\min}^d(|V|)\right) = O(|V|). \tag{4.3}$$

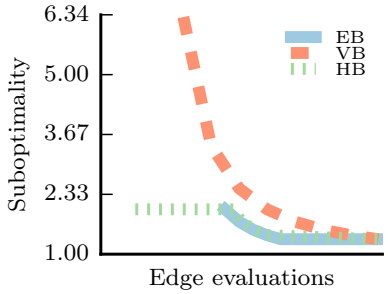
4.3.2 Effort-to-Quality ratio

We now compare our densification strategies in terms of their worst-case anytime performance. Specifically, we plot the cumulative amount of work as subgraphs are searched, measured by the maximum number of edges that may be evaluated, as a function of the bound on the quality of the solution that may be obtained using (Eq. 2.1). We fix a specific setting (namely d and n) and simulate the work done and the suboptimality using the necessary formulae. This is done for an easy and a hard problem. See Fig. 4.5.

Indeed, this simulation coincides with our discussion on properties of both batching strategies with respect to the problem difficulty. Vertex batching outperforms edge batching on easy problems and vice versa. Hybrid batching lies somewhere in between the two approaches with the specifics depending on problem difficulty.



(a) Easy problem



(b) Hard problem

Figure 4.5: A simulation of the work-suboptimality tradeoff for vertex, edge and hybrid batching. Here we chose $n = 10^6$ and $d = 4$. The easy and hard problems have $\delta^* = \sqrt{d}/2$ and $\delta^* = 5D_n$, respectively. The plot is produced by sampling points along the curves $|V| = n$ and $|E| = |V| \cdot (|V| - 1)/2$ and using the respective values in (Eq. 2.1). Note that x-axis is in log-scale.

4.4 Implementation

Our analysis and exposition so far has been independent of any parameters for the densification or any other implementation decisions. In this section we outline the specifics of how we implement the densification strategies for anytime planning on roadmaps, before going into the results for the same.

4.4.1 Densification Parameters

We choose the parameters for each densification strategy such that the number of batches is $O(\log_2 n)$.

4.4.1.1 Edge Batching

We set $\eta_e = 2^{1/d}$. Recall that for r-disk graphs, the average degree of vertices is $n \cdot r_i^d$, therefore this value (and hence the number of edges) is doubled after each iteration. We set $r_0 = 3 \cdot n^{-1/d}$.

4.4.1.2 Vertex Batching

We set the initial number of vertices n_0 to be 100, irrespective of the roadmap size and problem setting, and set $\eta_v = 2$. After each batch we double the number of vertices.

4.4.1.3 Hybrid Batching

The parameters are derived from those used for vertex and edge batching. We begin with $n_0 = 100$, and after each batch we increase the vertices by a factor of $\eta_v = 2$. For these searches, i.e. in the region where $n_i < n$, we use $r_i = 3 \cdot n^{-1/d}$. This ensures the same radius at n as for edge batching. Subsequently, we increase the radius as $r_i = \eta_e r_{i-1}$, where $\eta_e = 2^{1/d}$.

4.4.2 Optimizations

For the densification strategies to be useful in practice, we employ certain optimizations. To motivate this, consider the total work done (measured in edge evaluations) by the batching strategies in the worst case. Recall that for naively searching the entire roadmap, this work is $\binom{n}{2} \approx \frac{n^2}{2}$. Furthermore for our worst-case analysis, we assume that both batching strategies run for $\log_2 n$ iterations.

For vertex batching, the number of vertices at a given iteration is $n_i = 2^i$ and the number of vertices is $|E_i| \approx n_i^2/2 = 2^{2i-1}$. The

worst-case work complexity for vertex batching is:

$$\sum_{i=1}^{\log_2 n} |E_i| = \sum_{i=1}^{\log_2 n} 2^{2i-1} = \frac{2n^2}{3} + \text{lower order terms} \tag{4.4}$$

For edge batching, with low-dispersion sequences, we have that $|E_i| \approx n2^i$. The worst-case work complexity for edge batching is:

$$\sum_{i=1}^{\log_2 n} |E_i| = \sum_{i=1}^{\log_2 n} n2^i = 2n^2 + \text{lower order terms} \tag{4.5}$$

Note that hybrid batching's worst-case work would be greater than edge batching, so we omit the expression for brevity. Therefore, in all cases, the worst case work done by any batching strategy is strictly larger than searching $\mathcal{G}$ directly. Thus, we consider numerous optimizations and their effect on the overall performance.

4.4.2.1 Search Technique

Each subgraph is searched using Lazy A* [Cohen et al., 2014] with incremental rewiring as in LPA* [Koenig et al., 2004]. For details, see the search algorithm used for a single batch of BIT* [Gammell et al., 2014b]. This lazy variant of A* has been shown to outperform other path-planning techniques for motion-planning search problems with expensive edge evaluations [Dellin and Srinivasa, 2016].
N.B - We use the search technique described above in [Choudhury et al., 2016c] to compare directly with BIT*. However, as mentioned in chapter 3, in this thesis we propose using POMP (chapter 5, Choudhury et al. [2016b]) as the technique for searching each individual subgraph.

4.4.2.2 Caching Collision Checks

Each time the collision-detector $\mathcal{M}$ is called for an edge, we store the ID of the edge along with the result using a hashing data structure. Subsequent calls for that specific edge are simply lookups in the hashing data structure which incur negligible running time. Thus, $\mathcal{M}$ is called for each edge at most once.

4.4.2.3 Sample Pruning and Rejection

For anytime algorithms, once an initial solution is obtained, subsequent searches should be focused on the subset of states that could potentially improve the solution. When the space $\mathcal{X}$ is Euclidean, this, so-called "informed subset", can be described by a prolate hyperspheroid [Gammell et al., 2014a]. For our densification strategies, we prune away all existing vertices (for all batching), and reject the

newer vertices (for vertex and hybrid batching), that fall outside the informed subset.

Successive prunings due to intermediate solutions significantly reduces the average-case complexity of future searches [Gammell et al., 2014b], despite the extra time required to do so, which is accounted for in our benchmarking. Note that for Vertex and Hybrid Batching, which begin with only a few samples, samples in successive batches that are outside the current ellipse can just be rejected. This is cheaper than pruning, which is required for Edge Batching. Across all test cases, we noticed poorer performance when pruning was omitted.

In the presence of obstacles, the extent to which the complexity is reduced due to pruning is difficult to obtain analytically. As shown in Theorem 4.1, however, in the assumption of free space, we can derive results for Edge Batching. This motivates using this heuristic.

Theorem 4.1. *Running edge batching in an obstacle-free d-dimensional Euclidean space over a roadmap constructed using a deterministic low-dispersion sequence with $r_0 > 2D_n$ and $r_{i+1} = 2^{1/d} r_i$, while using sample pruning and rejection makes the worst-case complexity of the total search, measured in edge evaluations, $O(n^{1+1/d})$.*

Proof. Let c^i_{best} denote the cost of the solution obtained after i iterations by our edge batching algorithm, and $c_{\min} = \rho(s_1, s_2) \leq \sqrt{d}$ denote the cost of the optimal solution. Using (Eq. 2.1),

$$c^i_{\text{best}} \leq (1 + \varepsilon_i)\, c_{\min}, \tag{4.6}$$

where $\varepsilon_i = \frac{2D_n}{r_i - 2D_n}$. Using the parameters for edge batching,

$$\varepsilon_{i+1} = \frac{2D_n}{r_{i+1} - 2D_n} = \frac{2D_n}{2^{\frac{1}{d}} r_i - 2D_n} \leq \frac{\varepsilon_i}{2^{\frac{1}{d}}}. \tag{4.7}$$

Let $i_{\max}$ be the maximum number of iterations and recall that we have $i_{\max} = O(\log_2 n)$.

Note that the fact that vertices and edges are pruned away, does not change the bound provided in (Eq. 4.6). To compute the actual number of edges considered at the ith iteration, we bound the volume of the prolate hyperspheriod $\mathcal{X}_{c^i_{\text{best}}}$ in $\mathbb{R}^d$ (see [Gammell et al., 2014b]) by,

$$\mu\left(\mathcal{X}_{c^i_{\text{best}}}\right) = \frac{c^i_{\text{best}} \left(\left(c^i_{\text{best}}\right)^2 - c^2_{\min}\right)^{\frac{d-1}{2}} \xi_d}{2^d}, \tag{4.8}$$

where ξ_d is the volume of an $\mathbb{R}^d$ unit-ball. Using (Eq. 4.6) in (Eq. 4.8),

$$\mu\left(\mathcal{X}_{c^i_{\text{best}}}\right) \leq \varepsilon_i^{\frac{d-1}{2}} (1 + \varepsilon_i) (2 + \varepsilon_i)^{\frac{d-1}{2}} \Gamma_d, \tag{4.9}$$

where $\Gamma_d = \xi_d \cdot (c_{\min}/2)^d$ is a constant. Using (Eq. 4.7) we can bound the volume of the ellipse used at the i'th iteration, where $i \geq 1$,

$$\begin{aligned}\mu\left(\mathcal{X}_{c^i_{\text{best}}}\right) &\leq \varepsilon_i^{\frac{d-1}{2}} (1+\varepsilon_0)(2+\varepsilon_0)^{\frac{d-1}{2}} \Gamma_d \\ &\leq \eta^{-\frac{i(d-1)}{2}} \varepsilon_0^{\frac{d-1}{2}} (1+\varepsilon_0)(2+\varepsilon_0)^{\frac{d-1}{2}} \Gamma_d \\ &\leq 2^{-\frac{i(d-1)}{2d}} \mu\left(\mathcal{X}_{c^0_{\text{best}}}\right)\end{aligned} \tag{4.10}$$

Furthermore, we choose r_0 such that $\mu\left(\mathcal{X}_{c^0_{\text{best}}}\right) \leq \mu(\mathcal{X})$. Now, the number of vertices in $\mathcal{X}_{c^i_{\text{best}}}$ can be bounded by,

$$n_{i+1} = \frac{\mu\left(\mathcal{X}_{c^i_{\text{best}}}\right)}{\mu(\mathcal{X})} n \leq 2^{-\frac{i(d-1)}{2d}} n. \tag{4.11}$$

Recall that we measure the amount of work done by the search at iteration i using $|E_i|$, the number of edges considered. Thus,

$$|E_i| = O\left(n_i^2 r_i^d\right) = O\left(n^2 2^{-\frac{i(d-1)}{d}} \left(r_0 2^{\frac{i}{d}}\right)^d\right) = O\left(n 2^{\frac{i}{d}}\right) \tag{4.12}$$

Finally, the total work done by the search over all iterations is

$$O\left(\sum_{i=0}^{\log_2 n} n 2^{\frac{i}{d}}\right) = O\left(n \sum_{i=0}^{\log_2 n} 2^{i/d}\right) = O\left(n^{1+\frac{1}{d}}\right). \tag{4.13}$$

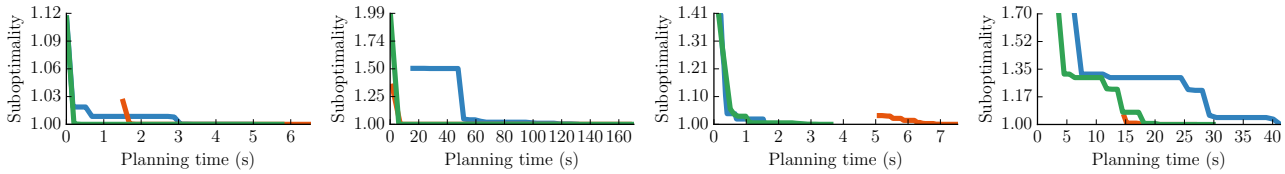
□

A similar result for vertex batching cannot be obtained simply because in the obstacle free case, vertex batching would find a solution immediately, rendering this analysis trivial. We omit the result for hybrid batching, but it can be shown by a similar process that it too has worst case work $O(n^{1+1/d})$.

4.4.2.4 Path shortcutting with local clique search

A useful local optimization technique that edge (and hybrid) batching allow for is searching the fully connected subgraph, i.e. the clique induced by the vertices on an intermediate feasible path [Geraerts and Overmars, 2007]. The extent of possible improvement depends on the actual vertices and the obstacle distribution, but intuitively, the optimization is useful for discovering longer edges early on that may be part of the globally optimal path. This can help address our earlier comment that edge (and hybrid) batching may suffer if the optimal path has very long edges.

If Halton sequences are used, for sufficiently large n, as shown earlier, we can bound the sub-optimality of intermediate path cost $c_{\mathcal{G}}$ in terms of the cost of the global optimum with δ-clearance, $c^*(\delta)$.



(a) $\mathbb{R}^2$ - Easy (b) $\mathbb{R}^2$ - Hard (c) $\mathbb{R}^4$ - Easy (d) $\mathbb{R}^4$ - Hard

Figure 4.6: Experimental results in random unit hybercube scenarios for ■ vertex batching ■ edge batching, and ■ hybrid batching. The y-axis is the ratio between the length of the path produced by the algorithm and length of γ^* (the shortest path on $\mathcal{G}$) for that problem. The naive strategy of searching the complete graph requires the following times to find a solution - (a) **44**s, (b) **200**s, (c) **12**s and (d) **56**s. In each case this is significantly more than the time for any other strategy to reach the optimum. Figure best viewed in color.

Furthermore we can bound the number of vertices lying on an intermediate path to be $\kappa < \frac{c_{\mathcal{G}}}{D(\mathcal{S})} < \frac{((1+\epsilon(n,r))c^*(\delta)}{D(\mathcal{S})}$. Searching for the shortest path on a clique of size κ is $O(\kappa^2)$.

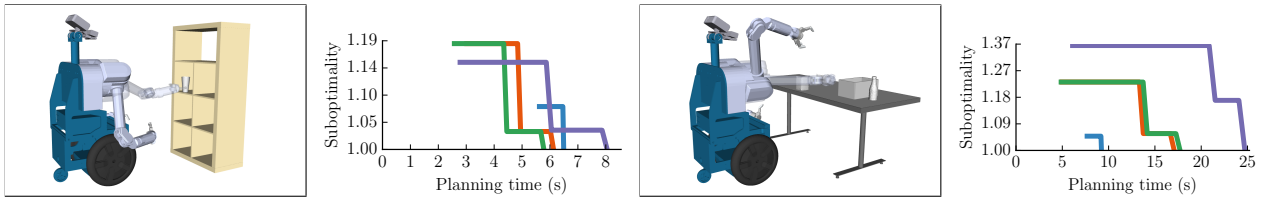
4.5 Experiments

Our implementations of the various strategies are based on the publicly available OMPL [Şucan et al., 2012] implementation of BIT* [Gammell et al., 2014b]. Other than the specific parameters and optimizations mentioned earlier, we use the default parameters of BIT*. Notably, we use the Euclidean distance heuristic, an approximately sorted queue, and limit graph pruning to changes in path length greater than 1%.

4.5.1 Random hypercube scenarios

The different batching strategies are compared to each other on problems in $\mathbb{R}^d$ for $d = 2, 4$. The domain is the unit hypercube $[0, 1]^d$ while the obstacles are randomly generated axis-aligned d-dimensional hyperrectangles. All problems have a start configuration of $[0.25, 0.25, \ldots]$ and a goal configuration of $[0.75, 0.75, \ldots]$. We used the first $n = 10^4$ and $n = 10^5$ points of the Halton sequence for the $\mathbb{R}^2$ and $\mathbb{R}^4$ problems, respectively.

Two parameters of the obstacles are varied to approximate the notion of problem hardness described earlier – the number of obstacles and the fraction of $\mathcal{X}$ which is in $\mathcal{X}_{obs}$, which we denote by ζ_{obs}. An easy problem is one which has fewer obstacles and a smaller value of ζ_{forb}. The converse is true for a hard problem. Specifically, in $\mathbb{R}^2$, we have easy problems with 100 obstacles and $\zeta_{\text{obs}} = 0.33$, and hard problems with 1000 obstacles and $\zeta_{\text{obs}} = 0.75$. In $\mathbb{R}^4$ we maintain the same values for ζ_{obs}, but use 500 and 3000 obstacles for easy and hard problems, respectively. For each problem setting ($\mathbb{R}^2$/$\mathbb{R}^4$; easy/hard) we generate 30 different random scenarios and evaluate each strategy with the same set of samples on each of them. Each random scenario has a different set of solutions, so we show a representative result for each problem setting in Fig. 4.6.



(a) (b) (c) (d)

Figure 4.7: We show results on 2 manipulation problems for ■ vertex batching ■ edge batching, ■ hybrid batching and ■ BIT*. For each problem the goal configuration of the right arm is rendered translucent. Both of the problems are fairly constrained and non-trivial. The problem depicted in ((c)) has a large clear area in front of the starting configuration, which may allow for a long edge. This could explain the better performance of vertex batching. The naive strategy takes 25s for ((b)) and 44s for ((d)) respectively.

The results align well with our intuition about the relative performance of the densification strategies on easy and hard problems. Note that the naive strategy of searching $\mathcal{G}$ with A* directly requires considerably more time to report the optimum solution than any other strategy. We mention the numbers in the accompanying caption of Fig. 4.6 but avoid plotting them so as not to stretch the figures. Note the reasonable performance of hybrid batching across problems and difficulty levels.

4.5.2 Manipulation planning problems

We also run simulated experiments on HERB [Srinivasa et al., 2010], a mobile manipulator designed and built by the Personal Robotics Lab at Carnegie Mellon University. The planning problems are for the 7-DOF right arm, on the problem scenarios shown in Fig. 4.7. We use a roadmap of 10^5 vertices defined using a Halton sequence $\mathcal{S}$ which was generated using the first 7 prime numbers. In addition to the batching strategies, we also evaluate the performance of BIT*, using the same set of samples $\mathcal{S}$. BIT* had been shown to achieve anytime performance superior to contemporary anytime algorithms. The hardness of the problems in terms of clearance is difficult to visualize in terms of the C-space of the arm, but the goal regions are considerably constrained. As our results show (Fig. 4.7), all densification strategies solve the difficult planning problem in reasonable time, and generally outperform the BIT* strategy on the same set of samples.

4.6 Discussion

In this chapter we presented, analyzed and implemented several densification strategies for anytime geometric motion planning on large dense roadmaps. We provided theoretical motivation for these densification techniques, and showed that they outperform the naive search significantly on difficult planning problems.

In this work we demonstrate our analysis for the case where the set of samples is generated from a low-dispersion deterministic sequence. A natural extension is to provide a similar analysis for a sequence

of random i.i.d. samples. Here, $f(|V|) = O(\log |V|)$ [Karaman and Frazzoli, 2010] instead of $O(|V|)$. When out of the starvation regions we would like to bound the quality obtained similar to the bounds provided by (Eq. 2.1). A starting point would be to leverage recent results [Dobson et al., 2015] for Random Geometric Graphs under expectation, albeit for a *specific* radius r.

Another question we wish to pursue is alternative possibilities to traverse the subgraph space of $\mathcal{G}$. As depicted in Fig. 4.1, our densification strategies are essentially ways to traverse this space . We discuss three techniques that traverse relevant boundaries of the space. But there are innumerable trajectories that a strategy can follow to reach the optimum. It would be interesting to compare our current batching methods, both theoretically and practically, to those that go through the interior of the space.

5
Search over Configuration Space Beliefs

(*This chapter is based on work presented in [Choudhury et al., 2016b].*)

In this chapter we develop an algorithm for anytime planning on roadmaps that uses a model of the configuration space to search for successively shorter paths that are likely to be feasible. It does so by searching for paths that are pareto-optimal in path length and collision probability, hence we call it Pareto-Optimal Motion Planner (POMP). As mentioned in chapter 3, we propose using POMP for searching each individual subgraph generated by the specific densification strategy used.

N.B - For this specific chapter we have a similar problem setting to that which we have been considering all this while, i.e. we are searching for a sequence of successively shorter feasible paths in a roadmap graph $G = (V, E)$ embedded in $\mathcal{X}$. We will sometimes use the symbol γ to refer to a path, interchangeably with γ. For this specific chapter we do not assume the roadmaps are large and dense, as we allow the densification strategy to handle that scenario.

5.1 Configuration Space Beliefs

For high dimensional problems, maintaining an explicit representation of $\mathcal{X}_{\text{obs}}$ (and hence $\mathcal{X}_{\text{free}}$) is not computationally feasible. We are interested in regimes where we have an *implicit* representation in the form of a *collision checker*, which takes as input a configuration $q \in \mathcal{X}$ and outputs which of the two sets $\mathcal{X}_{\text{obs}}$ or $\mathcal{X}_{\text{free}}$ the configuration belongs to. As mentioned in chapter 1, we focus on problem domains where performing each check is computationally expensive.

We observe immediately that the collision checker is an expensive but perfect *binary classifier*, classifying a queried configuration into $\mathcal{X}_{\text{obs}}$ or $\mathcal{X}_{\text{free}}$. We can then formulate an inexpensive but uncertain model Φ that takes in a configuration and outputs its belief that the query is collision-free, represented as $\rho : \mathcal{X} \mapsto [0, 1]$. We can build and

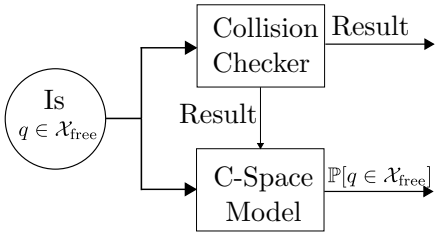
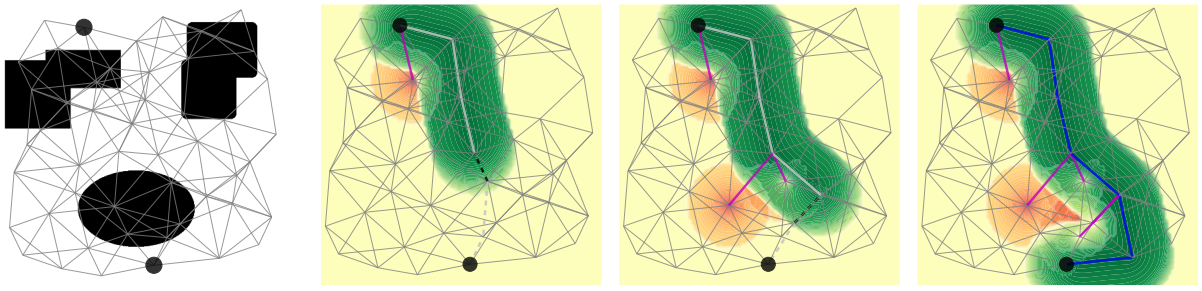
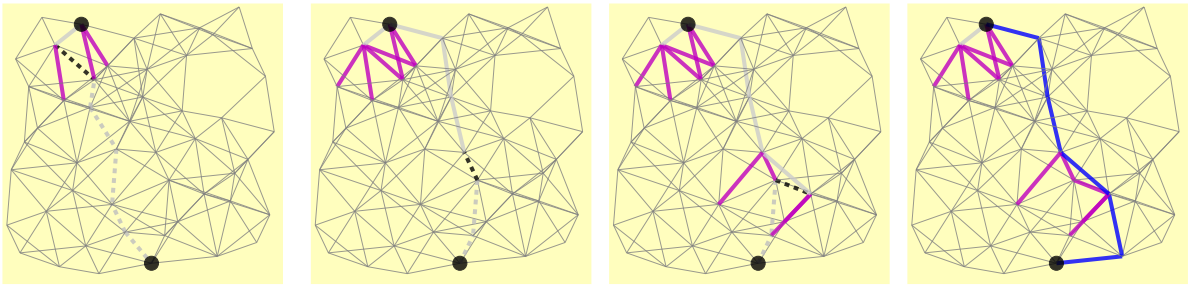


Figure 5.1: The configuration space belief model is updated with the results of collision checks and is queried to obtain the collision probability of an unknown configuration.



(a) Problem Environment (b) Model - Iteration 4 (c) Model - Iteration 7 (d) Model - First Path



(e) No Model - Iteration 4 (f) No Model - Iteration 10 (g) No Model - Iteration 15 (h) No Model - First Path

Figure 5.2: An illustration of the benefit of using configuration space beliefs. The upper and lower rows show runs of POMP on a 2D planning problem, with some finite model radius and zero radius respectively. The heatmap represents the belief model with green representing the belief of being free, and orange the belief of being in collision. The thin grey edges are unevaluated. The dashed edges are being evaluated. Thick grey edges are evaluated free, and thick magenta edges are evaluated in collision. The blue edges represent the first feasible path in each case. Using the belief model, POMP requires 10 evaluations for the first feasible path, while without the model, it requires 18 evaluations.

update this model using a black-box learner [Pan et al., 2013]. Given a query q, we now have the choice of either inexpensively evaluating $\rho(q)$ from the model $\mathbf{\Phi}$, or expensively querying the collision checker. A representation is shown in Figure 5.1. A number of different models exist in the literature [Burns and Brock, 2005a, Knepper and Mason, 2012, Pan et al., 2013] which could be used. We utilize a k-NN method similar to one used previously [Pan et al., 2013].

When $q_i \in \mathcal{X}$ is evaluated for feasibility, we obtain $F(q_i) = 0$ if $q_i \in \mathcal{X}_{\text{free}}$ or 1 otherwise. Then we add $(q_i, F(q_i))$ to the model. Given some new query point q, we obtain the k closest known instances to q, say $\{q_1, q_2 \dots q_k\}$, and then compute a weighted sum of $F(q_i)$ where a weight $w_i = \frac{1}{||q - q_i||}$. Therefore,

$$\mathbb{P}[q \text{ in collision }] = 1 - \rho(q) = \frac{\mathbf{w}.\mathbf{F}}{|\mathbf{w}|} \tag{5.1}$$

where $\mathbf{w} = [w_1, w_2 \dots w_k]^T$ and $\mathbf{F} = [F(q_1), F(q_2) \dots F(q_k)]^T$.

In principle, the k-NN lookup is $O(\log N)$ while a collision check is $O(1)$. However, for the roadmaps that we tested on, the time for a single collision check was significantly higher than for a model lookup. Asymptotically, the lookup time will exceed check time, which may happen in certain kinds of problems.

5.2 Edge Weights

We define two edge weight functions. The first is $w_l : E \to [0, \infty)$ and measures the length of an edge based on our distance metric on $\mathcal{X}$. For edges that are evaluated to be in collision, the weight is set to ∞. The path length is represented as $L(\gamma) = \sum_{e \in \gamma} w_l(e)$.

The second is $w_m : E \to [0, \infty)$, and it relates to the probability of the edge to be collision-free based on our model M. Specifically, $w_m(e) = -\log(\rho(e))$, where $\rho(e)$ is the probability of e to be collision-free. Note that a known-free edge has $w_m(e) = 0$ and a known-colliding edge has $w_m(e) = \infty$. If we assume conditional independence of configurations given the edge, we can write the log-probability of a path being in collision, $M(\gamma)$, in the same summation form as $L(\gamma)$:

$$-\log \mathbb{P}(\gamma \in \mathcal{X}_{\text{free}}) = -\log \prod_{e \in \gamma} \rho(e) = \sum_{e \in \gamma} w_m(e) \equiv M(\gamma) \tag{5.2}$$

We will refer to this M as the *collision measure* of the path. Ensuring that both $M(\gamma)$ and $L(\gamma)$ are additive over edges enables efficient searches.

5.3 Weight Constrained Shortest Path

Our first objective is to obtain some initial feasible path quickly, irrespective of path length. We search for paths that are most likely to be free according to our model. Once we have a feasible path, we search only for paths of shorter length, based on their likelihood of being free. Specifically, we want to search over paths most likely to be free, with a length lower than some upper bound, where the bound reduces over time, with each feasible solution. One way to represent this is by repeatedly solving

$$\begin{aligned} \hat{\gamma} = \ & \underset{\gamma}{\text{argmin}} \quad && M(\gamma) \\ & \text{subject to} \quad && L(\gamma) < L^* \end{aligned} \tag{5.3}$$

and subsequently evaluating the returned solution for feasibility. The initial bound $L_0^* = \infty$, after which $L_1^* = L(\hat{\gamma}_0)$, where $\hat{\gamma}_0$ is the first feasible solution, and $L_2^* = L(\hat{\gamma}_1)$ and so on. Therefore, the first iteration of the problem is an unconstrained shortest path problem. For a particular finite upper bound, however, this problem is an instance of the Weight Constrained Shortest Path (WCSP) problem.

Intuitively, it seems that this is how we want to formulate the problem. However, a closer look will reveal that this formulation is neither the most efficient nor the most appropriate. This will lead us

to the eventual formulation that we present. We visualize paths on a 2D plane in terms of their two weights - the path length L and the collision measure M. Each path is a point on this plane, as shown in Fig. 5.3.

For such bicriteria problems, a point (path) is *strictly dominated* by another point if it is worse off in both criteria. For instance, if there are two points τ, τ' such that $L(\tau') \geq L(\tau)$ and $M(\tau') \geq M(\tau)$, then τ strictly dominates τ', i.e. $\tau \succ \tau'$. A point is *Pareto optimal* if it is not strictly dominated by any other point. The set of Pareto optimal points is known as the *Pareto frontier*.

Consider a simple approach that uses WCSP and evaluates paths lazily, updating the model Φ after each search and solving the updated problem defined in 5.3. Let us call this the LazyWCSP method. It repeatedly performs a horizontal sweep over the points in the plane under some horizontal line, the upper length bound (initially there is no line as the bound is ∞). It selects the left-most one (with minimum collision measure) to evaluate, say γ_i. If the path is infeasible, it is moved infinitely to the right, and if feasible, it is moved left onto the y-axis, with the collision measure becoming 0. The upper bound L^* is now $L(\gamma_i)$, represented by a horizontal line. The collision data of previously unknown configurations updates the model Φ, which in turn updates the x-coordinate of certain points.

Recall that the search for the first feasible path $\hat{\gamma_0}$ is an unconstrained shortest path problem with edge weights defined by w_m and lazy evaluation of paths [Nielsen and Kavraki, 2000]. For the subsequent paths, we have to repeatedly solve the WCSP problem lazily.

There are two major issues with that approach. Firstly, from a practical viewpoint, the WCSP problem is known NP-Hard. There are algorithms for solving it in pseudopolynomial time, by dynamic programming [Desrochers and Soumis, 1988] and Lagrangean relaxation [Handler and Zang, 1980], but it is highly inefficient to do so repeatedly. Secondly, and more fundamentally, the progress of LazyWCSP does not appropriately address our goal of trading off between path length and collision measure. Consider the scenarios in Fig. 5.4. In Fig. 5.4b, LazyWCSP would evaluate a point γ_1 of marginally lower collision measure and higher path length than another one, γ_2. In the general case, where they may be many such points, as in Fig. 5.4c, this leads to prioritizing several paths that are less promising, i.e. that have a lower gain in length with respect to collision measure.

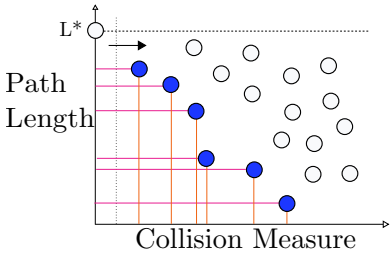
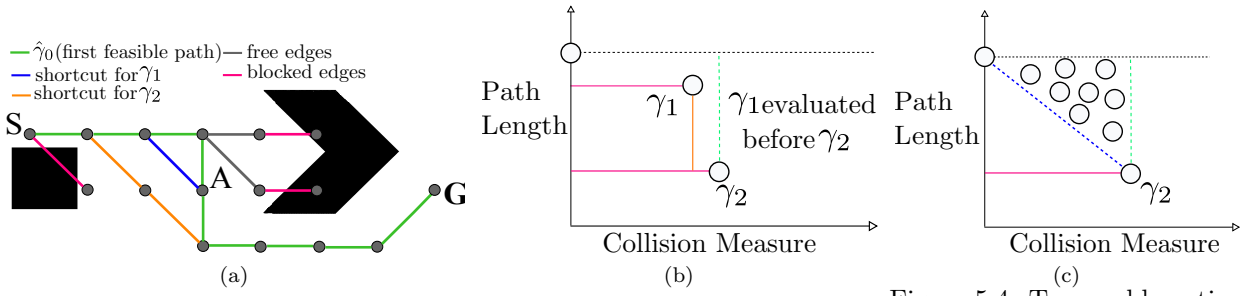


Figure 5.3: The LazyWCSP algorithm performs a horizontal sweep on the length(L) - measure(M) plane to select the left-most point. If each point chosen is feasible, and there are no updates to the model, this sweeps out the Pareto frontier of valid points, with respect to initial weights.

5.4 Convex Hull of the Pareto Frontier

Let us assume that we do not update the model Φ between successive searches; the x-coordinates of unevaluated paths do not change. If



(a) (b) (c)

Figure 5.4: Two problematic scenarios for using LazyWCSP. A toy case is shown in (a), along with two possible corresponding length(L) - measure(M) plots. In (b), a small decrement in collision measure for γ_1 is prioritized over a larger decrement in path length for γ_2. In (c), all points above the blue line and to the left of γ_2 are evaluated before the more promising γ_2. These points correspond to several paths through A.

an evaluated path is free, it becomes the best feasible solution, otherwise it is removed. Under this assumption, LazyWCSP traces out the Pareto frontier of the feasible paths with respect to their initial coordinates, as shown in Figure Fig. 5.3. In both of the examples in Figure Fig. 5.4, this defers the evaluation of the more promising γ_2.

We can control the tradeoff between the two weights by defining the objective function as a convex combination of the two weights,

$$J^\alpha(\gamma) = \alpha L(\gamma) + (1-\alpha)M(\gamma) \ , \ \alpha \in [0,1] \tag{5.4}$$

This is the key idea behind our algorithm POMP, or Pareto-Optimal Motion Planner (Algorithm 5.1). Minimizing J^α for various choices of α, traces out the convex hull of the Pareto frontier of the initial coordinates, as shown in Fig. 5.5a. The α parameter represents the tradeoff between the weights. Also, optimizing over J^α implicitly satisfies the constraint on L. If the current solution is γ_i, then for any path γ'

$$\begin{aligned} & J^\alpha(\gamma') < J^\alpha(\gamma_i) \\ & \implies \alpha L(\gamma') + (1-\alpha)M(\gamma') < \alpha L(\gamma_i) \ [\text{ as } M(\gamma_i) = 0] \\ & \implies L(\gamma') < L(\gamma_i) \end{aligned}$$

The path objective function is additive over edges, so each iteration of the algorithm is now a shortest path search problem. The edge weight for each search is

$$w_j^\alpha(e) = \alpha w_l(e) + (1-\alpha)w_m(e) \ , \ \alpha \in [0,1] \tag{5.5}$$

When the previous assumption is relaxed, i.e. when collision measures of paths are updated after each search, the corresponding points move and the Pareto frontier moves as well. POMP begins with $\alpha = 0$ and the upper bound $L^* = \infty$, as stated before. After the first feasible solution is obtained, α is increased, and for each value of α, the shortest path search is carried out with the weight function w_j^α. Either POMP finds a feasible path, and the next search uses this path

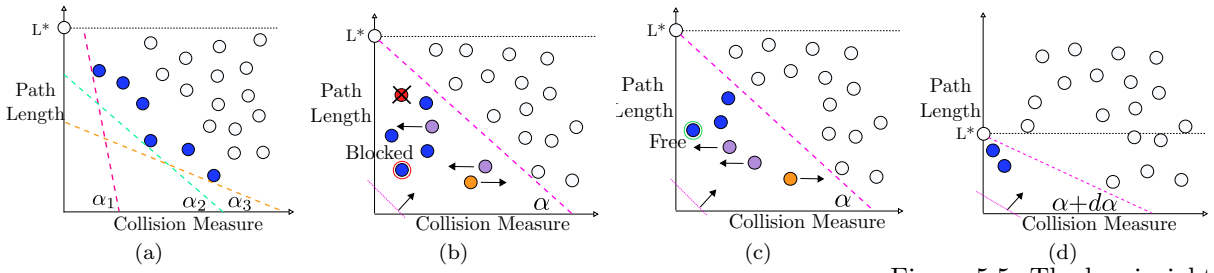


Figure 5.5: The key insights to our algorithm. As shown in (a), under the assumption of no model updates, the convex hull of the Pareto frontier of the paths is traced out for different values of α. The actual behaviour of POMP in the absence of that assumption is shown through (b) - (d). The diagonal sweep corresponds to searching with J^α, and the first point found in the sweep corresponds to the path that minimizes $J^\alpha(\gamma)$. When a path is evaluated, other paths may have their M-values ■ increased or ■ decreased, and may be ■ deleted if they share infeasible edges with it.

as the current solution, or it does not, and the previous solution is returned. In the latter case, there are no further updates the model can make (as the path returned is fully evaluated), and no other paths can be found with the current α. Therefore the search is restarted after increasing α. A visual description of an intermediate search of POMP is in Figure 5.5.

POMP is outlined in Algorithm 5.1. Algorithm 5.2 is a helper method for lazily evaluating paths. The term **yield** is used instead of **return** to emphasize that the algorithm has anytime behaviour.

Algorithm 5.1: POMP

Input : $G = (V, E)$, w_l, w_m, s, t, Φ

repeat
  $\gamma_0 \leftarrow$ `Dijkstra_Path`(G, w_m)
  `LazyEvalPath`(G, γ_0, Φ)
  if $\gamma_0 \in C_{\text{free}}$ **then**
    yield γ_0
  end if
until $\gamma_0 \in C_{\text{free}}$
$\gamma_{\text{curr}} = \gamma_0$
$\alpha \leftarrow \alpha'$
while $\alpha \leq 1$ **do**
  $w_{\text{j}}^\alpha(e) = \alpha w_l(e) + (1-\alpha) w_m(e),\ \forall\ e$
  $\gamma_{\text{new}} \leftarrow$ `AStar_Path`(G, w_{j}^α)
  `LazyEvalPath`(G, γ_{new}, Φ)
  if $\gamma_{\text{new}} \neq \gamma_{\text{curr}}$ and $\gamma_{\text{new}} \in C_{\text{free}}$ **then**
    $\gamma_{\text{curr}} \leftarrow \gamma_{\text{new}}$
    yield γ_{curr}
  end if
  Increment α
end while
$\gamma_{\text{shortest}} \leftarrow \gamma_{\text{curr}}$

Algorithm 5.2: LazyEvalPath

Input : $G = (V, E)$,γ, Φ

for $e \in \gamma$ **do**
  if e is unevaluated **then**
    `Evaluate`(e)
    Update Φ with collision data for e. This updates w_m for all unevaluated edges.
    if $e \in C_{\text{obs}}$ **then**
      $w_l(e), w_m(e) \leftarrow \infty$ ▷ Known blocked edge.
      return $\gamma \in C_{\text{obs}}$
    else
      $w_m(e) \leftarrow 0$ ▷ Known free edge.
    end if
  end if
end for
return $\gamma \in C_{\text{free}}$

5.5 Minimizing Expected Path Length

We will show that the behaviour of POMP is equivalent to minimizing expected path length, with some approximation. This formulation has been used in similar contexts [Missiuro and Roy, 2006].

Let $\hat{J}(\gamma) = \mathbb{E}[J(\gamma)]$ be the expected length of path γ. By the linearity of expectation,

$$\hat{J}(\gamma) = \sum_{e \in \gamma} \hat{w}_{\text{j}}(e)$$

where $\hat{w}_{\text{j}}(e)$ is the expected length of e. For any edge e, we use a model where the length of the edge, if free, is $w_l(e)$, and if in collision is β, where β is a penalty factor ($\beta > 0$). Though we require $\gamma \in \mathcal{X}_{\text{free}}$, we do not consider a $w_l(e)/\infty$ length model as that would make expected lengths infinite for any unevaluated edges. Because the algorithm eventually evaluates edges, no infeasible paths will be reported as solutions. Therefore,

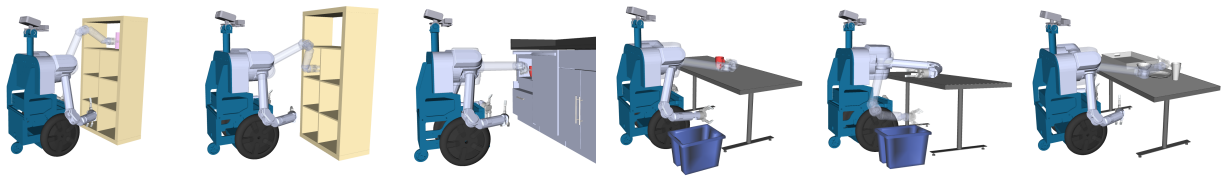
$$\hat{w}_{\text{j}}(e) = \rho(e) w_l(e) + (1 - \rho(e))\beta$$

using the standard notion of expectation over a single event. By the Taylor series expansion,

$$\log(\rho(e)) = \rho(e) - 1 - \frac{(\rho(e) - 1)^2}{2} + \dots$$
$$\implies -\log(\rho(e)) \approx (1 - \rho(e)) \text{ [neglecting other terms]}$$

Therefore, we obtain,

$$\hat{w}_{\text{j}}(e) = w_l(e)\rho(e) - \beta\log(\rho(e)) \equiv w_l(e) + \frac{\beta}{\rho(e)} w_m(e) \tag{5.6}$$



(a) P1 (b) P2 (c) P3 (d) P4 (e) P5 (f) P6

Figure 5.6: The test cases we use for our experiments. We name them P1 through P6 for reference. The planning is for the right arm of the robot, which is at the starting configuration in each case. The translucent rendered arm represents the desired goal configuration.

Compare this to $w_j^{\alpha}(e)$ in Eq. 5.5

$$\begin{aligned} w_j^{\alpha}(e) &= \alpha w_l(e) + (1-\alpha) w_m(e) \\ &\equiv w_l(e) + \frac{(1-\alpha)}{\alpha} w_m(e) \text{ [for minimizing]} \end{aligned}$$

Therefore, the effect of $\frac{(1-\alpha)}{\alpha}$ is equivalent to that of $\frac{\beta}{\rho(e)}$ as β goes from $\beta' \gg 0$ to 0 and α correspondingly goes from 0 to 1.

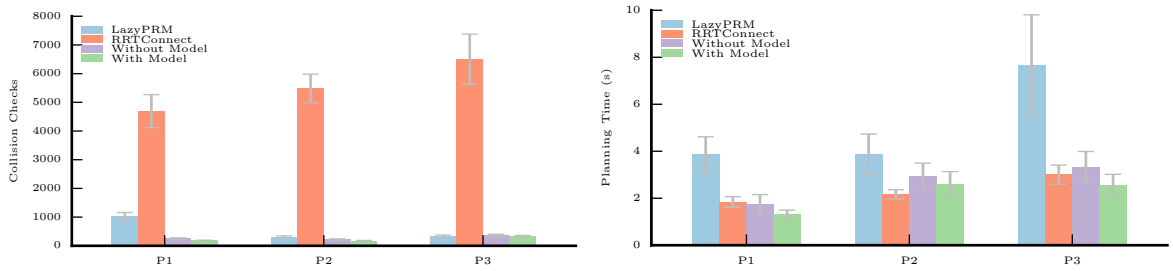
Intuitively, β represents the penalty factor that POMP assigns to additional collision checks. Reducing the penalty factor β from $\beta' \gg 0$ to 0 is analogous to increasing α in the earlier formulation from 0 to 1. Both operations represent the increased risk of collision the algorithm is willing to take while searching for edges that, if free, may potentially lead to shorter paths. It should also be noted that at the stage where $\alpha = 1 \implies \beta = 0$, POMP is equivalent to LazyPRM [Bohlin and Kavraki, 2000].

5.6 Experiments

We evaluate POMP through a number of simulated experiments on HERB [Srinivasa et al., 2010], a mobile manipulator designed and built by the Personal Robotics Lab at Carnegie Mellon University. We consider two hypotheses - the benefit of the model for computing the first solution, and the anytime performance.

Our experiments are run on 6 different planning problems for the 7-DOF right arm, shown in Fig. 5.6. The first three problems - P1, P2, P3 - are used for evaluating the first hypothesis. They have goal configurations with significant visibility constraints. The next three problems - P4, P5, P6 - are used for the second hypothesis. Their goal configurations are less constrained than the first set. Thus they have more feasible solutions and better demonstrate anytime behaviour.

For each problem, we test POMP over 50 different roadmaps. The distribution of the nodes is generated by Halton sequences [Halton, 1964], which have low dispersion, and the node positions are offset by random amounts. The roadmaps have approximately 14000 nodes, and the r-disk radius for connectivity is 0.3 radians.



(a) Number of Collision Checks

(b) Planning Time (s)

Figure 5.7: A comparison of our algorithm POMP with LazyPRM and RRTConnect, in terms of the average planning time and collision checks required for computing the first feasible path. POMP requires far fewer checks than RRTConnect, but spends additional time searching and updating the roadmap.

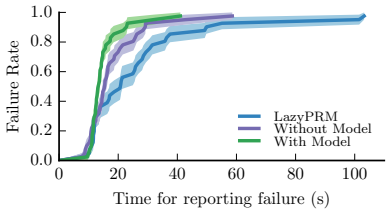
Using explicit roadmaps allows us to eliminate all nodes and edges which have configurations in self collision in a pre-processing step, thereby requiring us to only evaluate environmental collisions at runtime. We utilize the same set of default model parameters for each run of POMP - the joint angle resolution is 0.04 radians, the k for k-NN lookup is 15, the prior belief is 0.5, and α increases in steps of 0.1.

5.6.1 Benefit of model for first feasible path

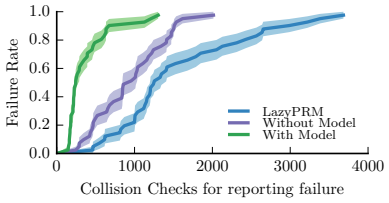
We evaluate the planning time and the number of collision checks required to obtain a feasible solution. We compare against the widely used LazyPRM [Bohlin and Kavraki, 2000] and RRT-Connect [Kuffner and LaValle, 2000]. For RRTConnect, we use the standard OMPL [Sucan et al., 2012] implementation. For LazyPRM, we use the search of POMP with $\alpha = 1$ on the same roadmaps as POMP. We also compare against a variant of POMP that does not use a belief model - it assigns the same probability of collision to all unknown configurations and only sets them to 0 or 1 when they are evaluated. This is to demonstrate the advantage of the model-based heuristic. We name these variants 'With Model' and 'Without Model'.

Fig. 5.7 shows the average collision checks and planning time to compute the first feasible solution for the various algorithms. This is for those roadmaps that have at least one feasible solution for the problem. A second perspective is shown in Fig. 5.9, which shows the success rate of the methods with time and checks. This plot considers all of the 50 roadmaps, whether they have a feasible solution or not, and so the success rate of the methods using them (With Model, Without Model, LazyPRM) all have the same upper bound.

The figures show that over all problems, POMP with a belief model shows superior average-case performance. Furthermore, the length of the first feasible path returned by POMP is better than RRTConnect. For the three problems, the average length of feasible paths computed by POMP is approximately 60% that of paths computed by RRT-

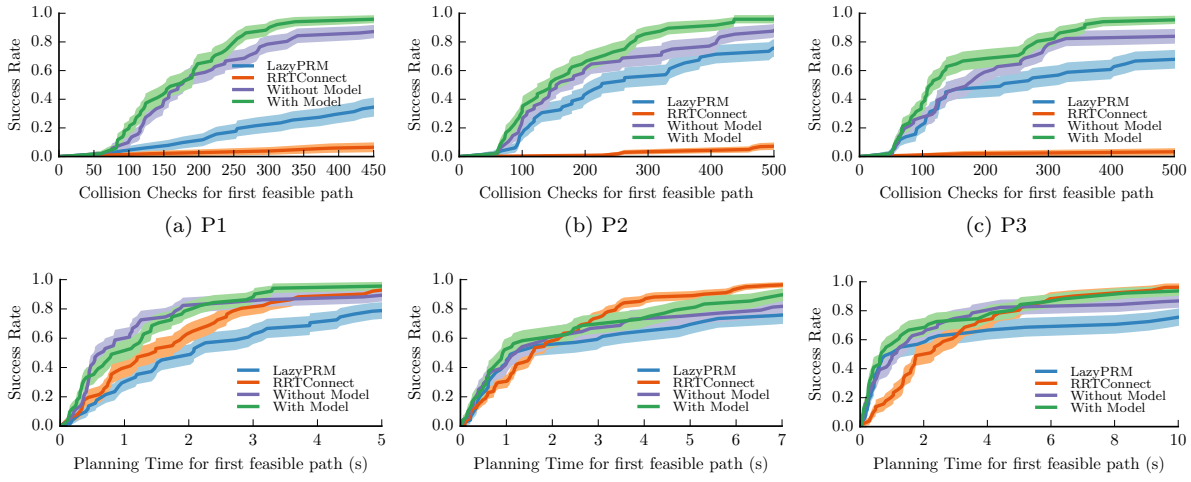


(a)



(b)

Figure 5.8: POMP with a model reports failure faster than without a model, and LazyPRM, for the same roadmaps and problem instances. This is aggregated over all failure cases from P1 through P3.



(a) P1 (b) P2 (c) P3

(d) P1 (e) P2 (f) P3

Figure 5.9: These plots shows how the collision checks and planning time required varies with the percentage of successful runs for each algorithm. Note that With Model, Without Model and LazyPRM all run on the same roadmap, and would all report a feasible solution if one existed on the roadmap. In each case, the x-axis is cut off after all runs of With Model, on roadmaps with feasible solutions, have concluded. RRTConnect would of course keep searching till it found a solution, and asymptotically its success rate would be 1.

Connect. Additionally, for cases where the roadmap has no feasible solution, POMP using a model reports failure more quickly than the variant without a model and LazyPRM (Fig. 5.8).

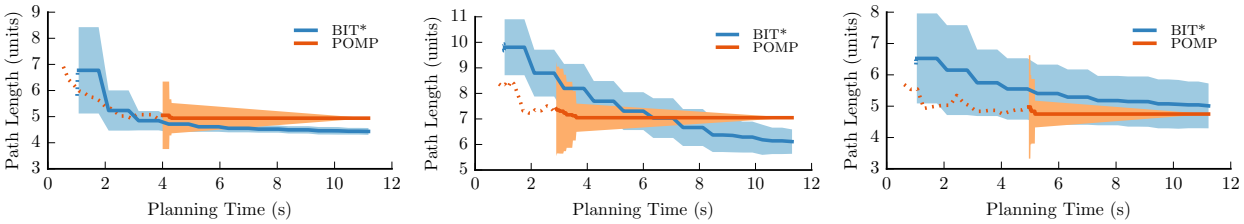
An interesting observation from Fig. 5.7 is that though RRTConnect has an order of magnitude more collision checks than POMP, the planning time is still comparable. A qualitative breakdown of the timing shows that POMP spends far less time than RRT-Connect actually doing collision checking. However, it also has far greater overhead for searching the roadmap for candidate paths and updating the collision measure of edges after collision tests.

5.6.2 Anytime behaviour of POMP

We also evaluate the anytime performance of finding shorter feasible paths over time, up to the optimal path in the roamdap. We compare against BIT* [Gammell et al., 2014b] (OMPL implementation), which has demonstrated an anytime performance superior to others. We run tests for 3 different problems P4, P5 and P6, and demonstrate the results in Fig. 5.10. Note that POMP works with only the roadmap provided, without any incremental sampling or rewiring, so the path length does not improve once the shortest feasible path has been obtained. BIT* adds more samples, however, and can continue to obtain improved paths with time.

5.7 Discussion

Given a roadmap constructed apriori, and a black-box configuration space model, POMP efficiently searches for shorter feasible paths in an anytime fashion. We thoroughly evaluated POMP for a set



(a) Anytime - P4 (b) Anytime - P5 (c) Anytime - P6

Figure 5.10: A comparison of the anytime performance of POMP with that of BIT*. This is done on 3 separate problems that better demonstrate anytime behaviour than the ones used earlier. The dotted line begins after 50% of the runs have found a solution. The solid line begins after all runs have found a solution. The flattening of the lines for POMP happens after the final roadmap finds the shortest path, as there is no further scope of improvement.

of roadmaps and model parameters and observed consistently good results in comparison to the state of the art. We have shown results for single-query problems, but POMP is also well-suited to multi-query problem instances that enable model re-use.

Like other PRM-based methods, POMP encounters the issue of there not being any feasible path on the roadmap for a particular environment and planning problem. The fast reporting of failure favours beginning with sparse roadmaps and incrementally densifying when no feasible path is found. Given that some approximate model of the world would already be available, techniques like utility-guided sampling [Burns and Brock, 2005b] could be used to efficiently sample new points in areas where they are likely to be beneficial.

For our implementation of POMP, the C-space belief model uses a simple but effective k-nearest neighbour lookup. We have previously referred to similar models in the literature. Other interesting approaches would be reasoning about the manifolds of the sample points [Salzman et al., 2013] and using persistent homology for occupancy maps [Bhattacharya et al., 2015].

6
Conclusion

In this MS thesis, we proposed an algorithmic framework for anytime geometric motion planning on large, dense roamdaps. Specifically,

> *This thesis proposes an algorithmic framework for two-level anytime geometric motion planning on large, dense roadmaps by generating subgraphs using densification and searching each subgraph using POMP*

We justified the benefits of using a roadmap that is very large and dense, along with some heuristic search techniques and lazy evaluations. In Sec. 4.6 We cast the problem of anytime planning on a large, dense roadmap to searching for the shortest feasible path over a sequence of subgraphs of the roadmap, using some densification strategy. When the set of samples is generated from a low-dispersion quasi-random Halton sequence, our analysis shows effort-vs-suboptimality bounds over the increasingly dense subgraphs of the complete roadmap.

In Sec. 5.7 we proposed an algorithm for anytime planning on a roadmap (of reasonable size) called Pareto-Optimal Motion Planner (POMP) that maintains a belief over configuration space and searches for paths Pareto-optimal in path length and collision measure. We particulary advocate for using POMP as the underlying search algorithm used in conjunction with a densification strategy.

POMP is well-suited for being run several times on the same configuration space and obstacle setting as the C-space belief model gets more and more informed as more edges are evaluated. In practice, using the Geometric Near-neighbour Access Tree or GNAT [Brin, 1995] datastructure, the model updates become reasonably efficient and the increased information from them is typically worth the computational cost of updation. The GNAT data structure is optimized for fast querying which we typically plan to do more than updation via collision checks. The fail-fast nature of POMP is useful for when the next batch does not have an improved feasible solution. Using densification

with POMP leads to an algorithm with multi-level anytime behaviour which is even better for the effort-to-quality tradeoff. POMP itself has no sub-optimality guarantees for intermediate solutions but as it obtains the shortest feasible path for the given roadmap that it searches, the overall guarantees via densification do stand.

6.1 Future Work

An immediate direction of future work would be to implement anytime roadmap planners which use densification strategies and POMP as the underlying search algorithm, and benchmark it against the same densification strategies with a different search algorithm (the one used in chapter 4). The behaviour of POMP as the batches get larger and larger, and the belief model gets more and more data points, would also be interesting to observe and analyse.

In the roadmap densification regime, at the end of each batch, the samples to be added for the next batch are already decided beforehand based on the generating sequence. However, as the belief model is continuously being updated, it induces a prior over the samples yet to be added. Whether this prior can be useful while adding the next batch is another interesting question to explore.

6.2 Final Remarks

Despite the recent advances in deep reinforcement learning for control and the advent of sensorimotor approaches [Levine et al., 2016, Pinto and Gupta, 2016], a large number of industrial applications for manipulation and mobile robotics rely on efficient motion planning algorithms. While theoretical properties like asymptotic optimality and probabilistic completeness are interesting and informative, we would like to have good quality solutions over a range of problems and be able to analyse the finite-time behaviour of our algorithms. With this thesis, we have made some amount of progress towards that end.

Bibliography

Oktay Arslan and Panagiotis Tsiotras. Dynamic programming guided exploration for sampling-based motion planning algorithms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4819–4826. IEEE, 2015.

Robert G Bartle. *The elements of integration and Lebesgue measure*. John Wiley & Sons, 2014.

Subhrajit Bhattacharya, Robert Ghrist, and Vijay Kumar. Persistent homology for path planning in uncertain environments. *IEEE Transactions on Robotics*, 31(3):578–590, 2015.

Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. IEEE, 2000.

Michael S. Branicky, Steven M. LaValle, Kari Olson, and Libo Yang. Quasi-randomized path planning. In *IEEE International Conference on Robotics and Automation*, pages 1481–1487, 2001.

Sergey Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 574–584. Morgan Kaufmann Publishers Inc., 1995.

Brendan Burns and Oliver Brock. Information theoretic construction of probabilistic roadmaps. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 650–655. IEEE, 2003.

Brendan Burns and Oliver Brock. Sampling-based motion planning using predictive models. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3120–3125. IEEE, 2005a.

Brendan Burns and Oliver Brock. Toward optimal configuration space sampling. In *Robotics: Science and Systems*, pages 105–112. Citeseer, 2005b.

Sanjiban Choudhury, Jonathan D Gammell, Timothy D Barfoot, Siddhartha S Srinivasa, and Sebastian Scherer. Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4207–4214. IEEE, 2016a.

Shushman Choudhury, Christopher M Dellin, and Siddhartha S Srinivasa. Pareto-optimal search over configuration space beliefs for anytime motion planning. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3742–3749. IEEE, 2016b.

Shushman Choudhury, Oren Salzman, Sanjiban Choudhury, and Siddhartha S Srinivasa. Densification strategies for anytime motion planning over large dense roadmaps. *arXiv preprint arXiv:1611.00111*, 2016c.

João Carlos Namorado Climaco and Ernesto Queirós Vieira Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11(4):399–404, 1982.

Benjamin J. Cohen, Mike Phillips, and Maxim Likhachev. Planning single-arm manipulations with n-arm robots. In *RSS*, 2014.

Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 2012.

Christopher M Dellin. Completing manipulation tasks efficiently in complex environments. 2016.

Christopher M Dellin and Siddhartha S Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *International Conference on Automated Planning and Scheduling*, pages 459–467, 2016.

Martin Desrochers and François Soumis. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR Information Systems and Operational Research*, 1988.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Andrew Dobson, George V. Moustakides, and Kostas E. Bekris. Geometric probability results for bounding path quality in sampling-based roadmaps after finite computation. In *IEEE International Conference on Robotics and Automation*, 2015.

Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, 2014a.

Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. *arXiv preprint arXiv:1405.5848*, 2014b.

Roland Geraerts and Mark H Overmars. Creating high-quality paths for motion planning. *The International Journal of Robotics Research*, 26 (8):845–863, 2007.

J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numer. Math.*, 2(1): 84–90, 1960.

John H Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, 1964.

Gabriel Y Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.

Pierre Hansen. Bicriterion path problems. In *Multiple criteria decision making theory and application*, pages 109–127. Springer, 1980.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

Lucas Janson, Brian Ichter, and Marco Pavone. Deterministic sampling-based motion planning: Optimality, complexity, and performance. *CoRR*, abs/1505.00023, 2015a.

Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *I. J. Robotics Res.*, pages 883–921, 2015b.

Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *RSS VI*, 104, 2010.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *I. J. Robotics Res.*, 30(7):846–894, 2011.

Lydia E Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

Lydia E. Kavraki, Mihail N. Kolountzakis, and Jean-Claude Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Trans. Robotics and Automation*, 14(1):166–171, 1998.

Michal Kleinbort, Oren Salzman, and Dan Halperin. Collision detection or nearest-neighbor search? on the computational bottleneck in sampling-based motion planning. *arXiv preprint arXiv:1607.04800*, 2016.

Ross A Knepper and Matthew T Mason. Real-time informed path sampling for motion planning search. *The International Journal of Robotics Research*, page 0278364912456444, 2012.

Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning A*. *Artificial Intelligence*, 155(1):93–146, 2004.

Richard E. Korf. Iterative-deepening-A*: An optimal admissible tree search. In *Joint Conference on Artificial Intelligence*, pages 1034–1036, 1985.

James J Kuffner and Steven 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. IEEE, 2000.

Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

Steven M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. In *IEEE International Conference on Robotics and Automation*, pages 473–479, 1999.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*, pages 767–774, 2004.

Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: an anytime, replanning algorithm. In *Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling*, pages 262–271. AAAI Press, 2005.

Tomas Lozano-Perez. Spatial planning: A configuration space approach. *IEEE transactions on computers*, (2):108–120, 1983.

Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

Patrycja E Missiuro and Nicholas Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1261–1267. IEEE, 2006.

Ian M Mitchell and Shankar Sastry. Continuous path planning with multiple constraints. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 5, pages 5502–5507. IEEE, 2003.

Harald Niederreiter. *Random Number Generation and quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1992.

Christian L Nielsen and Lydia E Kavraki. A two level fuzzy prm for manipulation planning. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 1716–1721. IEEE, 2000.

Jia Pan, Sachin Chitta, and Dinesh Manocha. Faster sample-based motion planning using instance-based learning. In *Algorithmic Foundations of Robotics X*, pages 381–396. Springer, 2013.

Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3406–3413. IEEE, 2016.

Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

Markus Rickert, Oliver Brock, and Alois Knoll. Balancing exploration and exploitation in motion planning. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2812–2817. IEEE, 2008.

Oren Salzman and Dan Halperin. Asymptotically-optimal motion planning using lower bounds on cost. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4167–4172. IEEE, 2015.

Oren Salzman and Dan Halperin. Asymptotically near-optimal RRT for fast, high-quality motion planning. *IEEE Trans. Robotics*, 32(3): 473–483, 2016.

Oren Salzman, Michael Hemmer, Barak Raveh, and Dan Halperin. Motion planning via manifold samples. *Algorithmica*, 67(4):547–565, 2013.

Oren Salzman, Kiril Solovey, and Dan Halperin. Motion planning for multi-link robots by implicit configuration-space tiling. *IEEE Robotics and Automation Letters*, 2016.

Gildardo Sánchez and Jean-Claude Latombe. On delaying collision checking in prm planning: Application to multi-robot coordination. *The International Journal of Robotics Research*, 21(1):5–26, 2002.

Jacob T Schwartz and Micha Sharir. On the âĂIJpiano movers'âĂİ problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on pure and applied mathematics*, 36(3):345–398, 1983.

Faye Smith and Brian Rooks. The harmonious robot. *Industrial Robot: an international journal*, 33(2):125–130, 2006.

Kiril Solovey, Oren Salzman, and Dan Halperin. New perspective on sampling-based motion planning via random geometric graphs. In *RSS*, 2016.

Siddhartha S Srinivasa, Dave Ferguson, Casey J Helfrich, Dmitry Berenson, Alvaro Collet, Rosen Diankov, Garratt Gallagher, Geoffrey Hollinger, James J. Kuffner, and Michael Vande Weghe. HERB: a home exploring robotic butler. *Autonomous Robots*, 28(1):5–20, 2010.

Siddhartha S Srinivasa, Aaron M Johnson, Gilwoo Lee, Michael C Koval, Shushman Choudhury, Jennifer E King, Christopher M Dellin, Matthew Harding, David T Butterworth, Prasanna Velagapudi, et al. A system for multi-step mobile manipulation: Architecture, algorithms, and experiments. In *International Symposium on Experimental Robotics*, pages 254–265. Springer, 2016.

Ioan Sucan, Maciej Moll, Lydia E Kavraki, et al. The open motion planning library. *Robotics & Automation Magazine, IEEE*, 19(4): 72–82, 2012.

Jur van den Berg, Rajat Shah, Arthur Huang, and Kenneth Y. Goldberg. Anytime nonparametric A. In *Association for the Advancement of Artificial Intelligence*, pages 105–111, 2011.

Christopher Makoto Wilt and Wheeler Ruml. When does weighted A* fail? In *Symposium on Combinatorial Search*, pages 137–144, 2012.