Motion Planning under Obstacle Uncertainty

by

Luke Shimanuki

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author . . . . . .
Department of Electrical Engineering and Computer Science
May 12, 2020

Certified by . . . . . .
Leslie P. Kaelbling
Panasonic Professor of Computer Science and Engineering
Thesis Supervisor

Certified by . . . . . .
Tomás Lozano-Pérez
School of Engineering Professor in Teaching Excellence
Thesis Supervisor

Accepted by . . . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Motion Planning under Obstacle Uncertainty

by

Luke Shimanuki

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We consider the problem of motion planning in the presence of uncertain or estimated obstacles. A number of practical algorithms exist for motion planning in the presence of known obstacles by constructing a graph in configuration space, then efficiently searching the graph to find a collision-free path. We show that such a class of algorithms is unlikely to be practical in the domain with uncertain obstacles. In particular, we show that safe 2D motion planning among polytopes with Gaussian-distributed faces (PGDF) obstacles is $NP-$hard with respect to the number of obstacles, and remains $NP-$hard after being restricted to a graph. Our reduction is based on a path encoding of MAXQHORNSAT and uses the risk of collision with an obstacle to encode variable assignments and literal satisfactions. This implies that, unlike in the known case, planning under uncertainty is hard, even when given a graph containing the solution. We further show by reduction from 3-SAT that both safe 3D motion planning among PGDF obstacles and the related minimum constraint removal problem remain NP-hard even when restricted to cases where each obstacle overlaps with at most a constant number of other obstacles. We then identify a parameter that determines the hardness of this problem. When this parameter is small, we present a polynomial time algorithm to find the minimum risk trajectory. When this parameter is larger, we are still able to guarantee an efficient runtime in exchange for an approximately optimal solution. This parameter seems to be small for most real world problems, suggesting that the tradeoff between computational efficiency and strong guarantees is not necessary. We also explore a connection between our work and previous work on the minimum constraint removal problem (MCR).

Thesis Supervisor: Leslie P. Kaelbling
Title: Panasonic Professor of Computer Science and Engineering

Thesis Supervisor: Tomás Lozano-Pérez
Title: School of Engineering Professor in Teaching Excellence

Acknowledgments

First, I would like to thank Leslie Pack Kaelbling and Tomás Lozano-Pérez for their patient mentorship and support. They have given so much since I joined the Learning and Intelligent Systems group (LIS) a bit over three years ago. I am grateful for the opportunity to work on exciting problems, the constant encouragement, and the guidance in research and academic matters. I would also like to thank the rest of LIS for welcoming me into the group. In particular, I would like to thank Brian Axelrod for taking me on as a freshman and continuing to work with me even after he left. None of this would have been done without his collaboration, and I thank him for helping me take this project from start to finish.

My time at MIT would not have been the same without the community I found in Asian Christian Fellowship (ACF). I am so grateful to everyone in ACF for welcoming me into the family and walking alongside me in all aspects of life and faith. In the context of this thesis, I would especially like to thank Christine Soh for being 6.UAR buddies, Chris Wang for sitting with me in CSAIL for so many hours, and Oliver Ren for opening his home to me in the midst of the COVID-19 situation – this thesis would not have been completed without them. I cannot emphasize enough how grateful I am for all the friendships and memories we made over the last four years. ACF has transformed my life in so many ways, and I will treasure every moment we had together.

I would also like to thank those who were a part of my journey before MIT. In particular, I thank Marco Pavone, Rick Zhang, and Federico Rossi from Stanford Autonomous Systems Lab for their mentorship and introduction into robotics research. I am especially grateful that they took me in as a high schooler without prior experience and gave me the opportunity to experience what research is like. I would also like to thank my high school robotics team AVBotz, where I first found my passion for robotics. Even before then, I have to thank my math teacher Randy Lomas for igniting at a young age the love for problem solving and analytical thinking that has served as a foundation for much of the work I do today.

I am incredibly grateful to my family for all of their support the past 22 years. I thank my parents for everything they put into raising me, loving me, and encouraging me to pursue the things that are important to me. I also thank my brother Brian for paving the way forward

from childhood to MIT, and inspiring me to grow into the person I have become.

Finally, I give thanks to God. To Him belongs the credit for every accomplishment, every word on this page, every part of who I am. I thank Him for creating a world with so many beautiful concepts to discover, and for giving us the opportunity to search for them in wonder and excitement. I thank Him for all the ways he has blessed me and provided for me, both before and during my time at MIT. And I thank Him for placing in my life all of the people mentioned above, that they may impact me in all the wonderful ways that they have.

Contents

List of Figures

Chapter 1

Introduction

This work is based on [39] and unpublished manuscripts that are jointly written with Brian Axelrod.

1.1 Motivation

Planning in an uncertain environment is one of the fundamental problems facing autonomous systems. Drones and autonomous cars, for example, must all navigate environments that are not known a priori. They must estimate the environment with noisy sensors. Unfortunately, this problem presents many difficulties when one wants to guarantee a low collision probability. One must reconstruct a model of the environment from sensor observations in a systematic way, quantify uncertainty in the estimation process and then compute a path with the appropriate information. While there are many methods to plan a trajectory between two points in a known environment with strong theoretical guarantees, few of them generalize to obstacles with locations estimated by noisy sensors. It has proven much harder to provide strong completeness, runtime, and optimality guarantees in this setting. For the rest of the paper, when the term safe is used to refer to a trajectory with low collision probability over the randomness of the uncertain environment.

There has been a long series of works in this area. One line of work models uncertainty in estimation of the environment as a partially observable Markov decision problem (POMDP) [9]. Unfortunately, solving POMDPs have proven difficult in practice for compli-

cated problem instances, despite large advances in POMDP solvers [26, 41]. Even solving finite horizon POMDPs is PSPACE hard, implying that an efficient, general algorithm does not exist [30].

However, there are many works that suggest that navigation among uncertain obstacles is an easier problem than solving general POMDPs. A long line of work approaches this problem, trading off performance, safety guarantees, completeness, and limitations of the model [5, 22, 29, 20, 42, 31, 32, 10, 15].

In this thesis, we focus on navigation among uncertain obstacles drawn from known distributions. In the scenario we examine, the robot first fixes a trajectory based on a known obstacle distribution. Then obstacles are drawn exactly once from the distribution and the robot executes the plan. The "risk" of the plan is based on the probability that the trajectory collides with an obstacle. Ideally, an algorithm would be always efficient, always finding the the optimal solution, and never returning a solution that is less safe than advertised.

There are several straightforward approaches to this problem, that while efficient, do not necessarily yield solutions that meet practical needs. The first would be to penalize each waypoint with its unconditional likelihood of collision. This approach corresponds to the assumption that collision probabilities at different waypoints are independent. The sum of the individual risks can be taken as a proxy for the total risk. Unfortunately, this has several undesirable properties. It is sensitive to the coarseness of the discretization. Taking a trajectory and discretizing more finely increases the computed risk bound even though the real risk does not change. At one extreme, using this calculation would show that a robot that doesn't change position is certain to collide! In reality, one expects collision probabilities at different points of a trajectory to potentially have significant correlation making a union bound a poor strategy by which to bound the collision probability.

This can be remedied by using the concept of a shadow. For every obstacle, a "shadow" is identified that contains the obstacle with a fixed probability. As long the robot follows a trajectory that avoids the shadows, the risk of the trajectory is at most the sum of the probabilities that each shadow contains the obstacle. If one can decide the probabilities corresponding to each shadow ahead of time, it becomes identical to a standard motion planning problem. Shadows effectively capture the notion that collision probabilities are

strongly correlated between points that are close together. Unfortunately, it is important to choose the probabilities at the same time the path is chosen. If it is not known a priori which obstacles the optimal trajectory must pass near, the shadows must be chosen conservatively in a way that has many undesired consequences. For example, even obstacles far away from the final trajectory which do not have a sizeable effect on the true collision probability may end up affecting the choice of trajectory and computed risk bound.

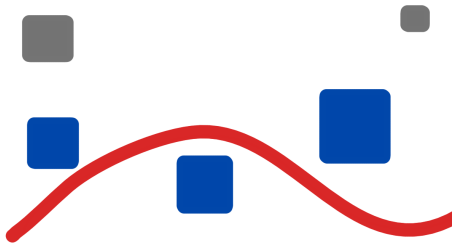

Figure 1-1: In the above example the dark blue estimated obstacles have a significant effect on the risk of collision, but the light gray obstacles do not.

Axelrod, Kaelbling, and Lozano-Pérez [4] posed the following question: Is there an efficient algorithm that, given a graph embedded in $\mathbb{R}^n$ and a set of obstacles, can find the path with minimal risk as computed via a shadow bound? The cost function derived from the shadow approximation is only influenced by the portion of the trajectory close to the obstacle and has submodular structure with respect to the graph. The fact that similar approximations have worked well for motion planning, and the existence of efficient algorithms for certain classes of submodular minimization problems gave the hope that it might be possible to find an efficient algorithm for this problem as well.

While motion planning is hard in general, practical and efficient algorithms have proven very successful under some assumptions [27]. One such body of work are the sampling-based motion-planning methods. These algorithms often have the assumption that the problem can be split into two pieces: First use a practically (though often not worst-case) efficient method to generate a small graph that contains a solution; then use a standard, efficient graph search algorithm to find the solution in this graph. Algorithms based on this scheme have been successful even for high-dimensional planning problems for robots with many degrees of freedom.

There are several other classes of practically efficient algorithms (including grid based

and optimization-based planners) that rely on the assumption that part of the problem may be solved much more efficiently in the average case than in the worst case. We discuss this further in the background section.

1.2 Contributions

This paper answers the question posed by Axelrod, Kaelbling, and Lozano-Pérez [4] in the negative.

Theorem 1. *Safe path planning in the presence of uncertain obstacles in 2 dimensions is NP-hard.*

A more formal statement of this result is presented in Section 2. We prove this by reducing from MAXQHORNSAT using a construction based on and very similar to that used by Erickson and LaValle [13] for the minimum constraint removal problem (MCR). We also show, via reduction from 3-SAT, that both safe path planning and MCR remain hard in three dimensions even when each obstacle overlaps with only a constant number of other obstacles, answering the question posed by Hauser [18]. Our first contribution is modifying the reduction to 2D MCR by Erickson and LaValle [13] to instead reduce to the 2D safe path planning problem. Our second contribution is presenting a new reduction from 3-SAT to a restricted version of safe path planning and MCR in 3D.

The proofs presented in this thesis illuminate what makes this problem more difficult than the standard motion-planning problem with known obstacles. Searching for the minimum-risk path does not have a Markov-like structure. Unlike in the shortest-path problem on a graph, the risk of the second half of a trajectory is very much affected by the first half. This means that the problem is lacking the Bellman property, as identified by Salzman, Hou, and Srinivasa [38]. The absence of a Markov-like property for the risk over the path has important ramifications for the complexity of the problem. In particular, the collision risk at different points along a trajectory can be highly correlated.

Because of this result, we cannot hope to find a provably efficient algorithm that works in general. Instead we identify a parameter which controls long range correlations between

collision probabilities, and in turn, the hardness of the planning problem. When this parameter is a small constant, we demonstrate a provably efficient, optimal, algorithm with rigorous guarantees. Fortunately, this parameter seems to be small for most practical instances. When the parameter cannot be controlled, we demonstrate a tradeoff between computation time and the suboptimality of the result.

As mentioned above, planning under obstacle uncertainty is hard because the risk of collision at potentially distant segments of a trajectory can be correlated. However, in many domains these correlated risks tend to be somewhat localized to nearby portions of the trajectory. We identify a parameter that determines this, the *collision horizon* parameter. It functions as a measure of how far apart these correlated collisions are in terms of the number of obstacle shadows entered in between them. Intuitively, this parameter captures the number of obstacles that are interacting with each other in such a way that a planner must reason about their collision risks jointly; or alternatively, how much collision history is needed to define a Markov state, that is, one that fully determines the marginal risk at future states.

We present a parameterized polynomial-time algorithm M_h, for finding the mininum risk path in a graph, for which the following informal theorems hold (the formal statements are presented later in the paper):

Theorem 2. *On problems with collision horizon at most h, M_h returns the minimal collision risk plan.*

Theorem 3. *On problems with collision horizon h', where $h' > h$, M_h produces a solution that is suboptimal at most by the risk incurred by correlated collisions in the optimal trajectory that are separated by more than h other collisions.*

We also show that minimum constraint removal, the problem of finding trajectories with the least number of collisions, is also solved by our algorithm with the same guarantees.

Theorem 4. *The greedy and exact algorithms presented by Hauser [18] are equivalent to M_0 and M_∞, respectively, applied to MCR.*

Hence, M_h can be seen as interpolating between the greedy and exact algorithms on MCR problems (but not on minimum-risk planning).

These definitions and theorems are stated formally in sections 2.3.1 and 4.2.

1.3 Background

Motion planning for robotics has been extensively studied in many different settings. One important high-level distinction between settings is whether the environment and state are known exactly or estimated.

1.3.1 Complexity in Motion Planning

The story of motion-planning algorithms in robotics has been one of walking the fine boundaries of complexity classes. On one hand, motion planning is PSPACE-hard in $\mathbb{R}^3$ [35] and $\mathbb{R}^2$ [11, 19] with respect to the number of degrees of freedom of a robot (and thus dimension of its configuration space). However, while Canny's [1988] work on singly-exponential time (with respect to number of degrees of freedom) roadmaps leads to a polynomial-time algorithm when the number of degrees of freedom is fixed, a different set of algorithms is used in practice. The robotics community has been able to find practically efficient methods that provide meaningful theoretical guarantees weaker than completeness (finding a solution if one exists). Sampling-based planners such as Rapidly-Exploring Random Trees (RRTs) [28, 27] and Probabilistic Roadmaps (PRMs) [25] are both practically efficient and *probabilistically complete* under some regularity conditions. Given effective heuristics, graph-based planners have also proved efficient and provide *resolution completeness* [27].

Searching for optimal plans, as opposed to simply feasible plans, further increases the difficulty. In a classic result, Canny and Reif [8] show that the 3-d Shortest-Path Problem is

NP-hard for a simple robot in terms of the number of obstacles. This ruled out results of the form of Canny's [1991] roadmap algorithm that showed fixed parameter tractability in the feasible motion planning case.

However, the community has been able to find practically efficient algorithms regardless of these worst-case results. A modified version of the original sampling-based algorithms allows them to return nearly optimal solutions in the limit [24] and graph-based planning algorithms are able to provide bounds on the suboptimality of their solutions [1].

Another motion-planning problem that lacks a Markov property is the minimum constraint removal problem (MCR), where the objective is to find a path that collides with the fewest obstacles. This problem was shown to be NP-hard in Cartesian spaces of dimension 3 [17, 18], and shortly later, in dimension 2 [13]. Eiben, Gemmell, Kanj, and Youngdahl [12] improve on these results by showing that MCR remains hard when obstacles are restricted to line segments or axis-aligned rectangles. Hauser [18] observes that MCR is in P when obstacles are connected and non-overlapping, and he suggests that the hardness seen in MCR is caused when an obstacle intersects with $O(n)$ other obstacles. Erickson and LaValle [13], Hauser [18], and Eiben, Gemmell, Kanj, and Youngdahl [12] further pose the open question of whether MCR remains hard when each obstacle overlaps with only a constant number of other obstacles. We then ask an analogous question: Is safe path planning in the presence of uncertain obstacles tractable when obstacles intersect only a constant number of other obstacles? These questions are stated more formally and answered in the negative in Section 2.

1.3.2 Planning under Uncertainty

While planning under uncertainty has been broadly studied in robotics, few methods have formal guarantees on solution quality and efficient runtime. We survey some of the related work below.

Many works assume some sort of uncertainty about the environment, but do not propose a model in which to rigorously quantify the uncertainty in the environment and provide guarantees about the success probability of the trajectory. Instead they often rely on heuristics

that seem to provide the desired behavior.

One line of work focuses on uncertainty in the robot's position. Here the model of the robot itself is "inflated" before the collision checking, ensuring that any slight inaccuracy in the position estimate or tracking of the trajectory does not result in a collision.

Work that focuses on uncertainty in the environment sometimes does the exact opposite. They often inflate the occupied volume of the obstacle with a "shadow" and ensure that any planned trajectory avoids the shadow [22, 29].

Both of these approaches work well when the obstacles are spaced out because there is still room to pass between them, but they become incomplete in more crowded domains when a trajectory must choose to risk collision with some obstacle, since the planner does not know beforehand how much it can afford to expand each shadow while still allowing a trajectory between nearby obstacles.

A more general approach that handles either or both of localization and obstacle uncertainty is belief-space planning. Belief space is the set of all possible beliefs about or probability distributions over the current state. Belief-space planning converts the uncertain domain in state space to belief space, then plans in belief space using trees [34, 5] or control systems [33].

Another line of work uses synthesis techniques to construct a trajectory intended to be safe by construction. If the system is modeled as a Markov decision process with discrete states, a safe plan can be found using techniques from formal verification [10, 15]. Other authors have used techniques from Signal Temporal Logic combined with an explicitly modeled uncertainty to generate plans that are heuristically safe [36].

Recent work by Hauser [18, 17] applies an approximate minimum constraint removal algorithm to motion planning under obstacle uncertainty by randomly sampling many draws for each obstacle and finding the path that intersects with the fewest samples. With this approach, he demonstrates low runtime and error on average although with poor worst case performance. He notes that his greedy minimum constraint removal algorithm is optimal when the optimal plan does not require entering an obstacle multiple times, and similarly his solution for motion planning under uncertainty is optimal when it is not required to risk collision with an obstacle multiple times.

Building on previous works using "shadows", Axelrod, Kaelbling, and Lozano-Pérez [3, 4] formalized the notion of a shadow in a way that allowed the construction of an efficient algorithm to bound the probability that a trajectory will collide with the estimated obstacles. However, the proposed RRT-based planning method making use of this algorithm does not return a solution with probability approaching 1 as the number of iterations approaches infinity (i.e. it lacks probablistic completeness). This is because this problem lacks optimal substructure: subpaths of an optimal path are not necessarily optimal, since the collision risk at different points along a trajectory can be highly correlated if it passes near the same obstacle multiple times.

We can now define a shadow rigorously:

Definition 1 (ϵ-shadow)**.** *A set $S \subseteq \mathbb{R}^d$ is an ϵ-shadow of a random obstacle $O \subseteq \mathbb{R}^d$ if $Pr[O \subseteq S] \geq 1 - \epsilon$.*

Shadows are important because they allow for an efficient method to upper-bound the probability of collision. If there exists an $\epsilon-$shadow of an obstacle that does not intersect a given trajectory's swept volume, then the probability of the obstacle intersecting with the trajectory is at most ϵ. An example of a shadow for an obstacle is shown in figure 1-2.

Shimanuki and Axelrod [39] show that planning in this domain is NP-hard in fixed dimension, and that even the problem of searching a graph for a safe path is NP-hard. The reduction they construct forces the planner to solve 3-SAT by encoding variable assignment and clause satisfaction into collision risks for the different obstacles, taking advantage of the fact that collisions can be correlated even between distant portions of a trajectory. Hence, it is believed that the hardness of the problem stems from long-distance dependencies between potential collisions.

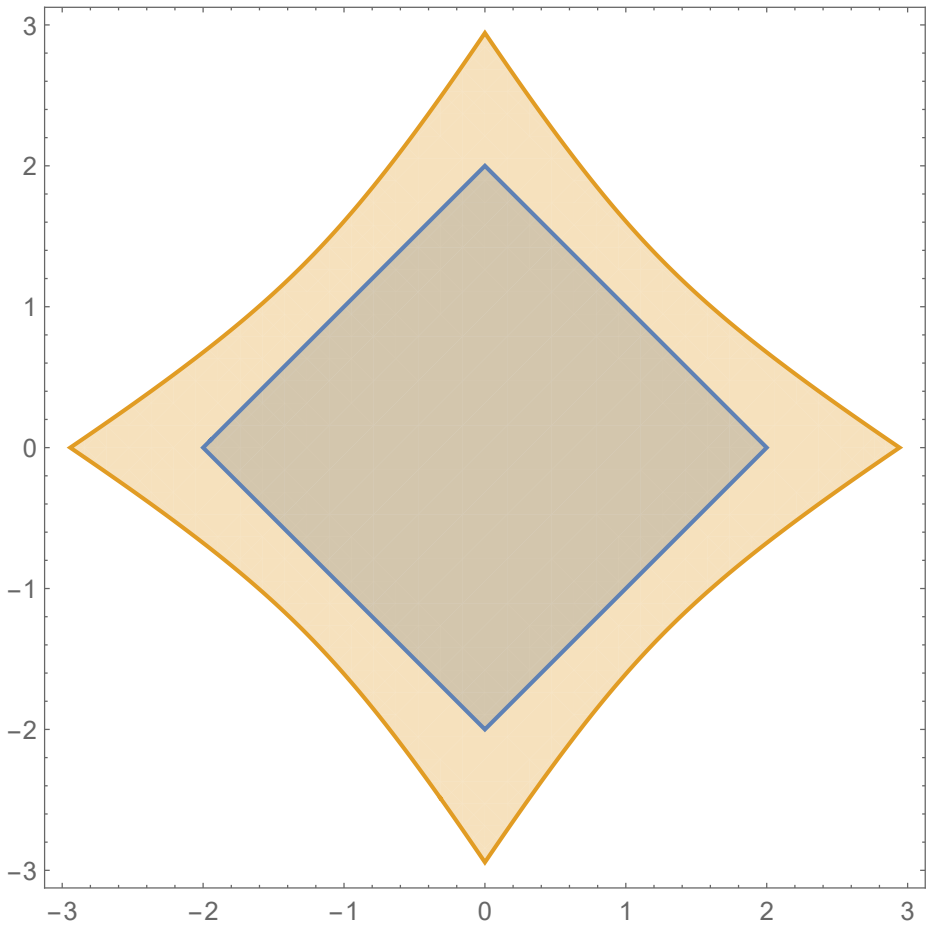

Figure 1-2: The orange set is a shadow of the obstacle. The blue set is the obstacle represented by the mean parameters.

Chapter 2

Problem Definition

2.1 Setup

2.1.1 Notation

In this section we will cover definitions and notation conventions that will be used in this thesis. A vector will be marked in bold as in $\mathbf{u}$, in contrast to a scalar u. $\wedge$, $\vee$, and $\neg$ are the logical AND, OR, and NOT operators, respectively. The power set (set of all subsets) of S is denoted by $\mathcal{P}(S)$. A function mapping into the power set of $\mathbb{R}^n$ outputs subsets of $\mathbb{R}^n$. We will use e_i to denote the ith standard basis vector.

2.1.2 Random Obstacle Model

Up until this point, we have been talking about uncertain obstacles in general. In this section, we define the abstraction we use to work with uncertain obstacles. Note that we are not working with uncertainty in the robot pose, only in the environment. Further, we will provide additional constraints on methods that first construct a graph embedded in configuration space, such as a PRM [25] or RRG [23], then search the graph for safe plans.

It is important that the obstacle distribution captures the fact that collision probabilities in different locations can be correlated. Consider the following toy example where a range sensor reports that an obstacle is 10 meters in front of the robot. At time $t = 1$ the robot drives forward 10 meters and then at $t = 2$ drives backwards 2 meters. If the robot did

not crash at time $t = 1$, it is unlikely to collide at time $t = 2$! Using a realistic model that captures correlations allows systems to be both safer and less conservative.

A useful reference here are the shadows defined by Axelrod et al. [4]. The paper defined shadows controlled by a single parameter (the probability that the shadow contains the obstacle). Shadows more likely to contain the obstacle strictly contain shadows less likely to contain the obstacle. The algorithm in our paper works with the *monotone obstacle model*, which includes the shadows used in [4]. Under this model, every node in the planning graph is assigned a risk for each obstacle. For a given path and particular obstacle, the risk of the path colliding with the obstacle is bounded by the maximum risk. Note that it does not matter if risks are associated with nodes or edges since a graph with risks at edges may be transformed via constructing a node for every edge.

Definition 2 (Monotone Risk Model)**.** *Given a graph $G = (E, V)$ annotated with "risks" r_i at node i, a function $f : \mathcal{P}(E) \to \mathbb{R}$ is a* Monotone Risk Model *if $f(E') = \max_{i \in E'} r_i$* or *$f$ is a monotone accumulation of Monotone Risk Models.*

We can examine how to compute such risk values under the model presented in [4]. Under this model, the risk for every node, for a given obstacle, corresponds to the probability of the minimal shadow that contains the node. A risk level is then a shadow with a particular, discrete risk value. A natural monotone accumulation would be the OR operation over collision probabilities. In practice – and in this work – a union-bound accumulation, that is, summation over probabilities, is often preferred for its simplicity. We note that our theorems and algorithms apply to both of these, as well as any other monotone accumulation model. Henceforth we will use $R[x \cup y]$ to denote this accumulation over the risks of events x and y.

We note this model is not limited to the shadows constructed in [4]. For example, it applies to shadows computed for different distributions using a similar techniques.

Furthermore, the discrete minimum constraint removal problem (MCR), can also be expressed as a minimization of a monotone risk model.

In the deterministic case, an obstacle is typically a set of points in task space that the robot would like to avoid. An equivalent view is to define an obstacle as a mapping from

robot trajectories to 1 or 0, indicating whether the swept volume of a robot executing that trajectory collides with the obstacle or not. We extend this notion of an obstacle to the uncertain case, defining it as a mapping instead from robot trajectories to collision *probabilities*.

Definition 3 (Obstacle)**.** *An obstacle is defined as a volume in task space (usually $\mathbb{R}^3$). We note that an obstacle also corresponds to a volume in configuration space (all volumes that result in the robot colliding with the obstacle).*

We are concerned with uncertain models, that is *random* obstacles drawn from a known distribution. We quantify the risk of collusion through monotone risk models corresponding to each obstacle. We let $f_o : E \to \mathbb{R}$ denote the risk model for a particular obstacle. We note that the values of this function can be computed using shadows.

It is convenient to work with trajectories as swept volumes in task space (i.e. the space that a robot moves through when following a given trajectory). For a given set of distributions over obstacles, the risk of a trajectory is the accumulation of risks over every obstacle. We frequently abuse notation, representing trajectories either in configuration space, task space or nodes and edges in a graph. This allows us to define an $\epsilon-$safe trajectory.

Definition 4 (ϵ-safe trajectory)**.** *Given joint distributions over random obstacles O, a trajectory τ is ϵ-safe for a sample $s_1..s_n \sim O$, $R[\tau \cap s_i \neq \varnothing \; \forall \; i] \leq \epsilon$. That is, a trajectory is ϵ-safe if the accumulated risk over all obstacles is less than ϵ.*

We note that this formulation differs from the notion of “risk-zones" evaluated by Salzman and Srinivasa [37] and Salzman, Hou, and Srinivasa [38], where the cost of a trajectory is proportional to the amount of time spent within a risk-zone. These problems share the lack of an optimal substructure—the subpaths of an optimal path are not necessarily optimal. Salzman, Hou, and Srinivasa [38] provide a generalization of Dijkstra’s algorithm that finds minimum-risk plans in their domain efficiently, but as we will show, there are no such techniques for our problem.

In our hardness proofs, we use a particular kind of monotone risk model: shadows over polytopes with Gaussian-distributed faces (PGDFs). Under the PGDF assumption, obstacles are the intersections of halfspaces with parameters drawn from multivariate normal

distributions. More formally a PGDF $O \subset \mathbb{R}^n$ is $O = \bigcap^{i} {\alpha_\mathbf{i}}^T \mathbf{x} \leq 0$, where $\alpha_\mathbf{i} \sim \mathcal{N}(\mu_\mathbf{i}, \mathbf{\Sigma_i})$. We can use homogeneous coordinates to create obstacles not centered about the origin.

One reason that PGDF obstacles are important is that we have methods of computing shadows for PGDF obstacles efficiently [3]. It is also able to capture collision correlations [3], which forms the basis for the hardness we encounter. This kind of model makes sense when constructing obstacle distributions from noisy sensor data, such as when estimating obstacle faces using Bayesian regression on point clouds from LIDAR input. One important property of this model is that the exact space of collisions is unknown, but the event of a collision is binary. In contrast, the "risk-zones" evaluated by Salzman and Srinivasa [37] are a better fit in domains where the area of risk is known, but the degree of risk is not absolute, e.g. a slippery floor.

2.2 Algorithmic Question

This leads to the following algorithmic question, of finding safe plans for a known distribution of obstacles.

Problem 1 (ϵ-safe Planning Problem)**.** *Given the obstacle distributions O and initial and end points $\mathbf{s}, \mathbf{t}$ in configuration space, find an ϵ-safe trajectory from $\mathbf{s}$ to $\mathbf{t}$ if one exists, otherwise FAIL.*

Note that there exists reductions between the safe planning problem and finding a path that minimizes the risk of collision. Since the risk ϵ is confined to $[0, 1]$, a binary search over ϵ yields an efficient algorithm that can approximately compute the minimum risk given an $\epsilon-$safe planner. For convenience, our proofs will consider the approximate minimum-risk planning problem, though the construction applies directly to $\epsilon-$safe planning as well.

Problem 2 ($(1 + \alpha)$-approximate minimum-risk planning problem)**.** *Given the parameters of PGDF distributions for each obstacle and initial and end points $\mathbf{s}, \mathbf{t}$ in*

configuration space, return a $((1+\alpha)\epsilon_*)$*-safe trajectory from* $\mathbf{s}$ *to* $\mathbf{t}$*, where* ϵ_* *is the minimum* ϵ *for which an* ϵ*-safe trajectory exists.*

2.2.1 Graph Restriction

We now consider the class of motion-planning algorithms that first construct a graph embedded in the robot's configuration space, and then run a graph-search algorithm to find a path within the graph. This class of algorithms has been shown to be practical in the known environment, with sampling-based planners such as RRT and PRM. Conditioned on there being a nonzero probability of sampling a solution, these algorithm are guaranteed to find a collision-free path with probability approaching 1 as the number of iterations approaches infinity. [28, 24, 25].

More formally this condition can be articulated as the existence of a path in the δ-interior of the free space X_{free}.

Definition 5 (δ-interior [24])**.** *A state* $x \in X_{free}$ *is in the* δ*-interior of* X_{free} *if the closed ball of radius* δ *around* x *lies entirely in* X_{free}*.*

This condition is necessary because it guarantees that finding a plan does not require waiting for a zero probability event. However this formulation does not extend well to the domain with uncertain obstacles; there is no concept of "free space" because the locations of the obstacles are not known. Instead we will use the equivalent view of inflating the path instead of shrinking the free space.

Definition 6. *The* δ*-inflation of the set* X *is the set* $Y = \bigcap_{x \in X} \{y \mid d(x,y) \leq \delta\}$.

where $d(x,y)$ is the distance between x and y.

We note that in the deterministic setting, if a trajectory is in the δ-interior of X_{free}, then the δ-inflation of the trajectory is entirely in X_{free}. This allows us to consider problems with the following regularity condition: there exists a δ-inflated trajectory that has a low risk of collision.

Definition 7 (ϵ-safe δ-inflated trajectory)**.** *A trajectory* τ *is an* ϵ*-safe* δ*-inflated trajectory if the accumulation over the risk that its* δ*-inflation intersects each obstacle is at most* ϵ*.*

We want to find an algorithm that satisfies the completeness and safety guarantees defined below.

Definition 8 (Probabilistically Complete ϵ-safe Planning Algorithm)**.** *A planning algorithm takes a set of obstacles, a start state* $\mathbf{s}$*, and a goal state* $\mathbf{t}$ *as input and generates an* ϵ*-safe trajectory as output. A planning algorithm is safe and probabilistically complete if, with* n *samples, the probability that it finds a safe trajectory approaches* 1 *as* n *approaches* ∞*.*

We also consider a special case where each obstacle overlaps with only a constant number of other obstacles.

Definition 9. *Two obstacles* O_i, O_j *overlap if there exists any point in space that both obstacles have a significant probability of intersecting. That is, there exists* $\mathbf{x} \in \mathbb{R}^d$ *such that* $Pr[\mathbf{x} \in O_i] \geq \epsilon$ *and* $Pr[\mathbf{x} \in O_j] \geq \epsilon$ *for* $\epsilon = \epsilon_*/|O|$*.*

Definition 10 (Probabilistically Complete κ-overlap $(1+\alpha)$-approximate Safe Planning Algorithm)**.** *A probabilistically complete* κ*-overlap* $(1+\alpha)$*-approximate safe planning algorithm is a planning algorithm that is probabilistically complete and* $(1+\alpha)$*-approximate safe for cases where the number of other obstacles that each obstacle overlaps with is at most* κ*.*

Axelrod, Kaelbling, and Lozano-Pérez [3] provide an extension of the RRT algorithm to the probabilistic domain using the shadow approximation. The uniqueness of paths between any two vertices in a tree makes finding the optimal (restricted to the tree) path trivial. However, while the paths it generates are indeed safe, the algorithm is not probabilistically complete, since it only considers the first path it finds to each point, even though it may not be the optimal subpath in order to reach the goal.

The following extension of the RRG algorithm is probabilistically complete, since all possible path are eventually generated [2].

Algorithm 1 SAFE_RRG

Input: End points $\mathbf{s}, \mathbf{t} \in \mathbb{R}^d$, set of obstacles O, and number of samples n.

Output: A ϵ-safe trajectory from $\mathbf{s}$ to $\mathbf{t}$ or FAIL.

1: G = CONSTRUCT_RRG(s, t, n)

2: τ = GRAPH_SEARCH(G, O, s, t)

3: **return** τ if RISK$(\tau) \leq \epsilon$ else NONE

SAFE_RRG requires as subroutine GRAPH_SEARCH, which is a ϵ-safe graph-search algorithm as defined below.

Definition 11 (minimum-risk graph-search algorithm)**.** *A minimum-risk graph-search algorithm is a procedure $\phi(G, O, \mathbf{s}, \mathbf{t})$, where G is a graph, O is a set of obstacles, and $\mathbf{s}$ and $\mathbf{t}$ are the start and end nodes in G, respectively. It returns a ϵ_*-safe trajectory in G, where ϵ_* is the minimum ϵ for which an ϵ-safe trajectory exists.*

Theorem 5 ([2])**.** *SAFE_RRG is probabilistically complete and ϵ-safe as long as GRAPH_SEARCH is complete.*

However, no graph-search procedure, beyond the naïve, exponential-time search procedure, is provided [2]. This means that, while the probability of success increases with more samples, the worst-case running time is exponential. Sampling-based motion-planning algorithms work in practice in the known environment because efficient graph-search algorithms can quickly find collision-free paths within a graph. In order for the $SAFE\_RRG$ class of algorithms to be practical, we would need a corresponding graph-search algorithm in the probabilistic domain. Because the cost of a path depends on what set of shadows it intersects, the state space of the graph search is not just the current node, but it also includes the accumulated risk incurred due to each obstacle. This means that the typical approaches for searching graphs with known obstacles, which make use of dynamic programming, cannot be applied in the same manner to graphs with unknown obstacles.

2.2.2 Complexity Results

Unfortunately, as will be shown in the remainder of this paper, Problem 1 and Problem 2 are NP-HARD with respect to $n = \Theta(|G| + |O|)$, the size of the input, even with a point robot two dimensions and given a graph containing the solution.

Theorem 6. *Unless $P = NP$, there is no $(1+\alpha)$-approximate ϵ-safe graph-search algorithm that runs in $POLY(n)$, time when restricted to graphs that embed in $\mathbb{R}^d$, $d = O(k)$ where k is the number of obstacles and $\alpha = \Theta(\frac{1}{n})$.*

We can strengthen this result to show that the minimum-risk planning problem is hard in general, that is, even when not restricted to a graph.

Theorem 7. *The $(1+\alpha)$-approximate minimum-risk planning problem is NP-hard. That is, unless $P = NP$, there is no $(1+\alpha)$-approximate Safe Planning Algorithm for $\mathbb{R}^2$ that runs in $POLY(n)$ when $\alpha = \Theta\left(\frac{1}{n^2}\right)$, even when provided a graph containing the solution.*

We show Theorem 6 and Theorem 7 by constructing a minimum risk planning problem in 2 dimensions which solves MAXQHORNSAT (an NP-complete problem). The proof follows the outline of the MCR hardness proof presented by Erickson and LaValle [13]. The main contribution of the work in this theorem is the connection to the minimum risk planning problem and the construction of the uncertain obstacles. However, the construction is not natural in the sense that certain constructed obstacles are used to correlate collision probabilities in disparate parts of the space. Some obstacles will be split by others and there is a high degree of overlap. We also show that the problem remains hard in 3D even when each obstacle overlaps with only a constant number of other obstacles.

Theorem 8. *The κ-overlap $(1+\alpha)$-approximate minimum-risk planning problem is NP-hard for $\kappa = O(1)$ in 3 dimensions. That is, unless $P = NP$, there is no κ-overlap $(1+\alpha)$-approximate Safe Planning Algorithm for $\mathbb{R}^3$ that runs in $POLY(n)$ when $\alpha = \Theta(\frac{1}{n^2})$, even when provided a graph containing the solution and when restricted to*

cases where each obstacle overlaps with at most κ other obstacles.

Furthermore, the proof can be extended to apply to MCR as well.

Theorem 9. *The κ-overlap minimum constraint removal problem is NP-hard for $\kappa = O(1)$ in 3 dimensions.*

We show Theorems 8 and 9 by constructing a minimum risk planning problem and related MCR problem in 3 dimensions which solves 3-SAT. We believe the construction presented here is of particular value because it is very natural – the construction is simple and does not require that any obstacle be immediately adjacent to more than a constant number of other obstacles.

2.3 Parameterized Algorithm

2.3.1 Formal Definition of Collision Horizon

Here we identify a parameter that drives the hardness of these problems. More specifically, we define the collision horizon which, loosely speaking, captures the length of dependencies between obstacles.

We first define the collision coverage, which describes the distance between correlated collisions for a single obstacle.

Definition 12 (collision coverage)**.** *The collision coverage $H_o^{(\tau)}$ for a given trajectory τ and obstacle $o \in O$ in minimum-risk graph-search problem instance $(G, O, \mathbf{s}, \mathbf{t})$ is the number of obstacles (including o itself) for which τ enters a higher risk level between the first time it enters a given risk level of o and the last time. More formally, treating τ as a sequence of edges, let $\zeta_{o'}^{(\tau)}$ denote the set of indices t for edges for which $\tau(t)$ enters a higher risk level of obstacle o', ie $f_{o'}(\tau(1..t) > f_{o'}(\tau(1..t-1)))$. Then*

$$H_o^{(\tau)} = \sum_{o' \in O} \mathbb{1}(\exists t \in \zeta_{o'}^{(\tau)} \; s.t. \; \min \zeta_o^{(\tau)} \leq t \leq \max \zeta_o^{(\tau)}).$$

This leads us to define the collision horizon for the overall problem instance.

Definition 13 (collision horizon)**.** *The collision horizon h of a given minimum-risk graph-search problem instance $(G, O, \mathbf{s}, \mathbf{t})$ is the maximum collision coverage of any obstacle of a minimum-risk trajectory, or more formally, $h = \min_{\tau \in T_*} \max_{o \in O} H_o^{(\tau)}$, where T_* is the set of minimal-risk trajectories from $\mathbf{s}$ to $\mathbf{t}$.*

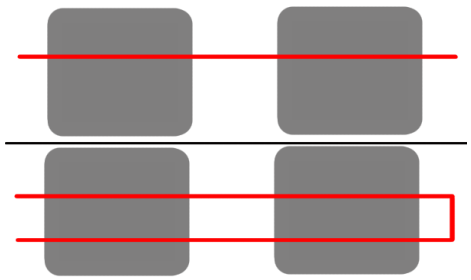

Figure 2-1: In the first setting the red path doesn't return to any earlier obstacles, leading to a collision horizon of 0. In the second setting the red path returns to the first obstacle after going through the second, leading to a collision horizon of 1.

2.3.2 Fixed-parameter Result

We can now define a parameterized graph-search algorithm.

Definition 14 (h-horizon minimum-risk graph-search algorithm)**.** *A h-horizon minimum-risk graph-search algorithm is a procedure $\phi(G, O, \mathbf{s}, \mathbf{t})$, where G is a graph, O is a set of obstacles, and $\mathbf{s}$ and $\mathbf{t}$ are the start and end nodes in G, respectively. When the collision horizon of the problem is at most h, it returns a ϵ_*-safe trajectory in G, where ϵ_* is the minimum ϵ for which an ϵ-safe trajectory exists.*

Later in this thesis we present a polynomial-time h-horizon minimum-risk graph-search algorithm.

Chapter 3

Complexity

3.1 Hardness Results in $\mathbb{R}^2$

3.1.1 Maximum Quadratic Horn Clause Satisfiability

Maximum Quadratic Horn Clause Satisfiability (MAXQHORNSAT) is an NP-complete problem whose input is a Boolean formula given in conjunctive normal form. It consists of the intersection of many clauses, each consisting of at most two literals (i.e. is quadratic), and each clause contains at most one positive literal (i.e. is a Horn clause) [21]. In other words, it is of the form $((x_0 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_2) \wedge ...)$. While QHORNSAT, the decision problem of determining the satisfiability of the input formula, is in P [14], MAXQHORNSAT, the problem of determining the maximum number of clauses that can be satisfied, is NP-hard [21]. MAXQHORNSAT was used by Erickson and LaValle [13] to show that minimum constraint removal (MCR), a similar problem, is NP-hard. Our reduction is based on that used by Erickson and LaValle and will use a similar construction modified to apply to the safe planning problem. We will consider a formula with n_v variables, n_n clauses with two negative literals, n_p clauses with one positive literal and one negative literal, and n_s clauses with only a single literal. We also define the total number of clauses $n_c = n_s + n_p + n_n$ and the total size of the problem $n = n_v + n_c$.

3.1.2 Proof Outline

We prove Theorem 6 using a reduction from MAXQHORNSAT. Given a MAXQHORNSAT instance, we construct an $\mathbb{R}^2$ $(1+\alpha)$-approximate minimum-risk planning problem and graph containing the solution. Our construction will have two kinds of obstacles. High-risk obstacles will induce a sufficiently high risk that any "reasonable" solution will go through the minimal number of these obstacles. Low-risk obstacles will affect the collision risk much less and will be used to count how many clauses are satisfied. The sum of the potential risk of all low-risk obstacles will be less than that of a single high-risk obstacle. This means the optimal solution will always choose to avoid a high-risk obstacle whenever possible, regardless of how many low-risk obstacles it must pass in order to do so.

1. Construct a portion of the graph for the algorithm to assign every variable by taking either the *left* or *right* branch, corresponding to setting the value of each variable to *true* or *false*, respectively. A high-risk obstacle corresponding to each branch will ensure that the optimal path only goes down one of the branches.

2. Construct a portion of the graph for the algorithm to select a literal from each clause to try to satisfy by taking either the *left* or *right* branch, corresponding to selecting the first or second literal, respectively. Choosing a branch which corresponds to a different assignment than in the first part would result in passing by an extra high-risk obstacle.

3. Construct low-risk obstacles such that there will be additional collision risk each time the selected literal is not satisfied by the chosen variable assignment.

The solution to this planning problem can then be transformed into a solution to the original MAXQHORNSAT instance in polynomial time (via observing which nodes were visited), demonstrating that $(1+\alpha)$-approximate safe graph search is at least as hard as MAXQHORNSAT.

3.1.3 Obstacle Templates

Throughout this reduction we will construct a number of obstacles using a few common templates, so for simplicity of notation we will define a few parameterized types of obstacles. These obstacle templates will fall into two categories based on how much we want them to affect the cost of a trajectory: low-risk obstacles and high-risk obstacles. In figures, high-risk obstacles will be denoted in blue and low-risk obstacles will be denoted in green.

The first kind of obstacle, shown in Figure 3-1, is a low-risk obstacle parameterized by a line segment $(\mathbf{u}, \mathbf{v})$. It is a long, thin obstacle that runs parallel to $(\mathbf{u}, \mathbf{v})$ and has one edge with uncertain position such that there is a risk of collision with points along $(\mathbf{u}, \mathbf{v})$.

$$\widehat{C}(\mathbf{u}, \mathbf{v}, \alpha) = \left\{ \mathbf{x} \;\middle|\; \begin{array}{l} \mathbf{x} \in \mathbb{R}^2 \\ \dfrac{1}{|\mathbf{v}-\mathbf{u}|}(\mathbf{v}-\mathbf{u})^T(\mathbf{x}-\mathbf{u}) \geq -\epsilon_C \\ \dfrac{1}{|\mathbf{u}-\mathbf{v}|}(\mathbf{u}-\mathbf{v})^T(\mathbf{x}-\mathbf{v}) \geq -\epsilon_C \\ \alpha \leq \dfrac{1}{|\mathbf{v}-\mathbf{u}|}(R_{\frac{\pi}{2}}(\mathbf{v}-\mathbf{u}))^T(\mathbf{x}-\mathbf{u}) \leq \epsilon_C \end{array} \right\}$$

for small constant ϵ_C, and where R_θ is the 2D rotation matrix for a clockwise rotation with angle θ. Note that $\widehat{C}$ defines a rectangular obstacle, where the position of one edge is parameterized by α. Then we can define a distribution over such obstacles

$$C(\mathbf{u}, \mathbf{v}) = \widehat{C}(\mathbf{u}, \mathbf{v}, \alpha) \text{ where } \alpha \sim \mathcal{N}(\mu_C, \sigma_C^2)$$

for small constants μ_C and σ_C. It guarantees that any point within distance ϵ_C of $(\mathbf{u}, \mathbf{v})$ has a risk of collision of at most $r_c = \Phi\left(-\frac{1}{\sigma_C}(\mu_C - \epsilon_C)\right)$ and at least $r'_c = \Phi\left(-\frac{1}{\sigma_C}(\mu_C + \epsilon_C)\right)$, and any point with distance further than $z_C\epsilon_C$ from $(\mathbf{u}, \mathbf{v})$ for some constant $z_C > 1$ has a risk of collision of at most $r_f = \Phi\left(-\frac{1}{\sigma_C}(\mu_C + \frac{1}{\sqrt{2}}z_C\epsilon_C)\right)$ (lower bounded by $r'_f = 0$ because risk becomes arbitrarily small as distance from the obstacle increases), where Φ is the cumulative distribution function of the standard normal distribution. Note that given some value of ϵ_C we can set μ_C, σ_C, and z_C to achieve any desired values of r_c, r'_c, and r_f. In particular, we can make r'_c/r_c arbitrarily close to 1 by decreasing ϵ_B, and make r_f/r_c arbitrarily close to 0 by increasing z_B.

The next kind of obstacle is identical to the line-segment obstacle defined above and shown in Figure 3-1 except it is a high-risk obstacle, so it has a higher risk of intersecting with points near the line segment (so it can be thought of as having a higher weight in terms of affecting the risk of a path).

$$\widehat{V}(\mathbf{u},\mathbf{v}) = \left\{ \mathbf{x} \;\middle|\; \begin{array}{l} x \in \mathbb{R}^2 \\ \frac{1}{|\mathbf{v}-\mathbf{u}|}(\mathbf{v}-\mathbf{u})^T(\mathbf{x}-\mathbf{u}) \geq -\epsilon_V \\ \frac{1}{|\mathbf{u}-\mathbf{v}|}(\mathbf{u}-\mathbf{v})^T(\mathbf{x}-\mathbf{v}) \geq -\epsilon_V \\ \alpha \leq \frac{1}{|\mathbf{v}-\mathbf{u}|}(R_{\frac{\pi}{2}}(\mathbf{v}-\mathbf{u}))^T(\mathbf{x}-\mathbf{u}) \leq \epsilon_V \\ \alpha \sim \mathcal{N}(\mu_V, \sigma_V^2) \end{array} \right\}$$

for small constant ϵ_V. Note that $\widehat{V}$ defines a rectangular obstacle, where the position of one edge is parameterized by α. Then we can define a distribution over such obstacles

$$V(\mathbf{u},\mathbf{v}) = \widehat{V}(\mathbf{u},\mathbf{v},\alpha) \text{ where } \alpha \sim \mathcal{N}(\mu_V, \sigma_V^2)$$

for small constants μ_V and σ_V. As before, it guarantees that any point within distance ϵ_V of $(\mathbf{u},\mathbf{v})$ has a risk of collision of at most $r_{Vc} = \Phi(-\frac{1}{\sigma_V}(\mu_V - \epsilon_V))$ and at least $r'_{Vc} = \Phi(-\frac{1}{\sigma_V}(\mu_V + \epsilon_V))$, and any point with distance further than $z_V\epsilon_V$ from $(\mathbf{u},\mathbf{v})$ for some constant $z_V > 1$ has a risk of collision of at most $r_{Vf} = \Phi(-\frac{1}{\sigma_V}(\mu_V + \frac{1}{\sqrt{2}}z_V\epsilon_V))$ (lower bounded by $r'_{Vf} = 0$). Again, given some value of ϵ_V we can set μ_V, σ_V, and z_V to achieve any desired values of r_{Vc}, r'_{Vc}, and r_{Vf}, and so let us set the constants such that

$$r_{Vc} = 5n_c r_c$$
$$r'_{Vc} = 5n_c r'_c$$
$$r_{Vf} = 5n_c r_f.$$

The final type of obstacle, shown in Figure 3-2, is another low-risk obstacle parameterized by a single point $\mathbf{q}$ and a horizontal direction h (either 1 or -1, corresponding to right and left, respectively). It is a small obstacle that sits to the side of $\mathbf{q}$ in the direction specified by h and has one edge with uncertain position such that there is risk of collision

Figure 3-1: The illustrations of the obstacle template $C(u, v)$. Note that it is a PGDF obstacle with the height of the obstacle being the only part that is random. Figure 3-1a illustrates a probability density function of collision. The distance between u and the top line or between v and the top line is ϵ_C. Figure 3-1b illustrates several obstacles drawn from the obstacle template. Note that if u in collision so is v and vice versa.

with $\mathbf{q}$ and points nearby q.

$$\widehat{B}(\mathbf{q}, h, \alpha) = \left\{ x \;\middle|\; \begin{array}{l} \mathbf{x} \in \mathbb{R}^2 \\ \mathbf{e_2}^T\mathbf{q} - \epsilon_B \leq \mathbf{e_2}^T\mathbf{x} \leq \mathbf{e_2}^T\mathbf{q} + \epsilon_B \\ (\mathbf{e_1}^T\mathbf{q} + \alpha)h \leq \mathbf{e_1}^T\mathbf{x}h \leq (\mathbf{e_1}^T\mathbf{q} + \epsilon_B)h \end{array} \right\}$$

for small constant ϵ_B (also recall that $\mathbf{e_i}$ refers to the ith standard basis vector). Note that $\widehat{B}$ defines a rectangular obstacle, where the position of one edge is parameterized by α. Then we can define a distribution over such obstacles

$$B(\mathbf{q}, h) = \widehat{B}(\mathbf{q}, h, \alpha) \text{ where } \alpha \sim \mathcal{N}(\mu_B, \sigma_B^2)$$

for small constants μ_B and σ_B. It guarantees that any point within a ball of radius ϵ_B around $\mathbf{q}$ has a risk of collision of at most $r_c = \Phi\left(-\frac{1}{\sigma_B}(\mu_B - \epsilon_B)\right)$ and at least $r_c' = \Phi\left(-\frac{1}{\sigma_B}(\mu_B + \epsilon_B)\right)$, and any point outside a ball of radius $z_B\epsilon_B$ around $\mathbf{q}$ for some constant $z_B > 1$ has a risk of collision of at most $r_f = \Phi\left(-\frac{1}{\sigma_B}(\mu_B + \frac{1}{\sqrt{2}}z_B\epsilon_B)\right)$. As before, note

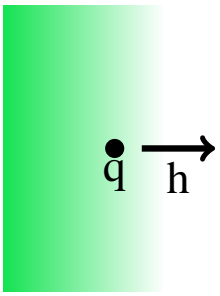

Figure 3-2: An example of the $B(q, h, \alpha)$ obstacle template. The distance between q and the left edge of the obstacle is $epsilon_B$

that given some value of ϵ_B we can set μ_B, σ_B, and z_B to achieve any desired values of r_c, r'_c, and r_f. Since these will also be low-risk obstacles, let us say that they will take on the same risk values as with the obstacles defined by C above.

3.1.4 Variable Gadgets

First, for each variable i in the MAXQHORNSAT problem, we construct a section of the graph where the choice of path corresponds to choosing either a true or false value of variable i. We illustrate this in Figure 3-3 and formalize this below. For variable i we construct vertices

$$\mathbf{u_i^v} = (0, 3i)$$
$$\mathbf{a_i^v} = (-(n_v - i) - 1, 3i + 1)$$
$$\mathbf{b_i^v} = (n_v - i + 1, 3i + 1)$$
$$\mathbf{v_i^v} = (0, 3i + 2)$$

and edges

$$(\mathbf{u_i^v}, \mathbf{a_i^v})$$
$$(\mathbf{u_i^v}, \mathbf{b_i^v})$$
$$(\mathbf{a_i^v}, \mathbf{v_i^v})$$
$$(\mathbf{b_i^v}, \mathbf{v_i^v}).$$

Each pair of consecutive loops is connected by an additional edge $(\mathbf{v_i^v}, \mathbf{u_{i+1}^v})$ for all i..

This entire set of variable gadgets will be mirrored at the bottom, with the positive-negative clause gadgets (see Section 3.1.5 between them. The bottom set of variable gadgets

will be needed for the negative-negative clause gadgets (see Section 3.1.5). Then for each variable i we construct a mirrored loop with vertices

$$\begin{aligned}
\mathbf{u_i^{v\prime}} &= (0, 7n_v + 3n_p - 3i) \\
\mathbf{a_i^{v\prime}} &= (-(n_v - i) - 1, 7n_v + 3n_p - 3i + 1) \\
\mathbf{b_i^{v\prime}} &= (n_v - i + 1, 7n_v + 3n_p - 3i + 1) \\
\mathbf{v_i^{v\prime}} &= (0, 7n_v + 3n_p - 3i + 2)
\end{aligned}$$

and edges

$$\begin{aligned}
&(\mathbf{u_i^{v\prime}}, \mathbf{a_i^{v\prime}}) \\
&(\mathbf{u_i^{v\prime}}, \mathbf{b_i^{v\prime}}) \\
&(\mathbf{a_i^{v\prime}}, \mathbf{v_i^{v\prime}}) \\
&(\mathbf{b_i^{v\prime}}, \mathbf{v_i^{v\prime}})\,.
\end{aligned}$$

Likewise, consecutive loops are connected by an additional edge $(\mathbf{v_i^{v\prime}}, \mathbf{u_{i+1}^{v\prime}})$ for all i.

In order to ensure that the resulting path selects the same variable assignment in the top set and the mirrored set, for each variable i, we construct an obstacle that has risk of colliding with the *true* path in both versions, and another obstacle that has a risk of colliding with the *false* path in both versions. These obstacles are given by

$$\begin{aligned}
&V(\mathbf{a_i^{v}}, \mathbf{a_i^{v\prime}}) \\
&V(\mathbf{b_i^{v\prime}}, \mathbf{b_i^{v}}).
\end{aligned}$$

Because the collision risks are correlated, selecting the same value in the bottom gadget as in the top gadget will incur no additional risk of colliding with the corresponding obstacle, but selecting a different value will incur the additional risk of colliding with the other obstacle.

3.1.5 Clause Gadgets

There are three types of clause gadgets, each of which will need to be handled separately: single literals, each with only a single positive or negative literal, positive-negative clauses, each with one positive literal and one negative literal, and negative-negative clauses, each

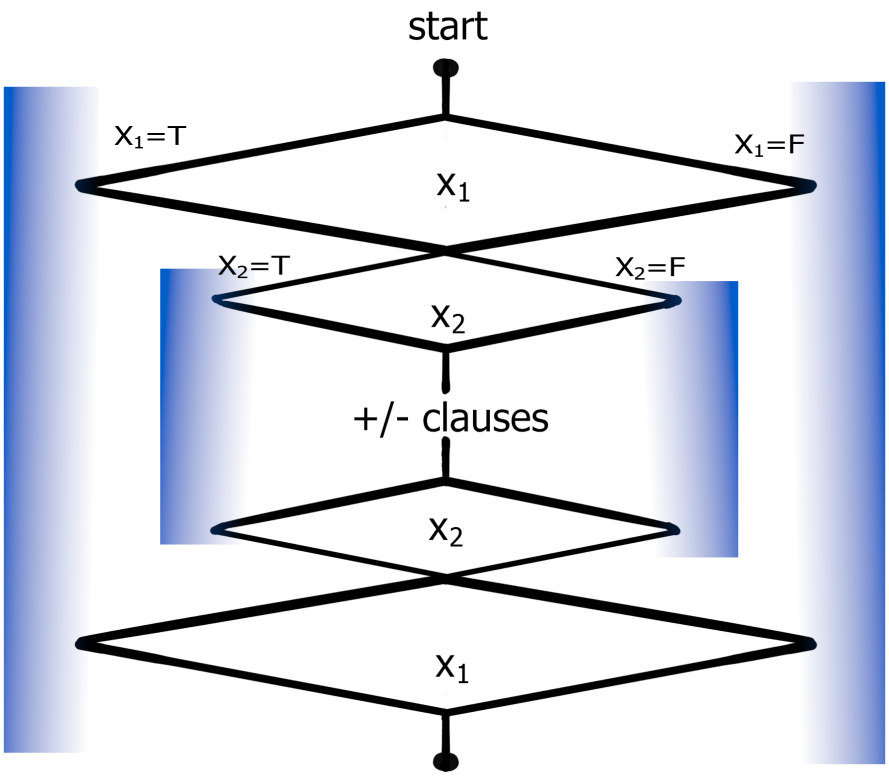

Figure 3-3: The variable gadget loops (solid lines) and obstacles (gradient-shaded rectangles). A path assigns a *true* or *false* value to each variable by selecting a branch through the variable gadget loop to traverse. It must also select the same variable assignment in the mirrored gadgets at the bottom in order to avoid additional collision risk. Notice that only high-risk obstacles are used in these gadgets, as they are constructed with the V template. There are two mirrored copies of each loop, with the positive-negative clause (or $+/-$ clause) gadgets in the center. The positive-negative clause gadgets will be constructed in Section 3.1.5. Also notice how loops closer to the center have smaller width, so a straight line can be drawn from any loop to the center without intersecting any other loops.

with two negative literals.

Single Literal

This first case is the simplest, as there is no choice to make about which literal to satisfy, so we do not need to add any additional components to the graph. For each single-literal clause j, let c_j^s denote the variable specified by the literal. Then we construct obstacles such that setting variable c_j^s to the opposite value in the variable assignment gadgets will incur additional risk. For a positive literal, the obstacles are constructed as

$$C(\mathbf{a}^{\mathbf{v}}_{\mathbf{c_j^s}}, \mathbf{u}^{\mathbf{v}}_{\mathbf{c_j^s+1}})$$
$$B(\mathbf{b}^{\mathbf{v}}_{\mathbf{c_j^s}}, 1)$$

and for a negative literal, the obstacles are constructed as

$$C(\mathbf{u}^{\mathbf{v}}_{\mathbf{c_j^s+1}}, \mathbf{b}^{\mathbf{v}}_{\mathbf{c_j^s}})$$
$$B(\mathbf{a}^{\mathbf{v}}_{\mathbf{c_j^s}}, -1).$$

Notice that each path through the variable gadget loop will incur risk of colliding with one of these obstacles, but at the end of the loop it will pass near the same obstacle that is near the branch corresponding to a satisfying assignment. Therefore selecting a variable assignment that satisfies this clause will risk collision with only one of these obstacles, whereas selecting a variable assignment that does not satisfy this clause will risk collision with both obstacles.

Positive and Negative Literal

For each clause j with one positive literal and one negative literal, we construct a loop below the top set of variable gadgets, with vertices

$$\mathbf{u}^{\mathbf{p}}_{\mathbf{j}} = (0, 4n_v + 3j)$$
$$\mathbf{a}^{\mathbf{p}}_{\mathbf{j}} = (-1, 4n_v + 3j + 1)$$
$$\mathbf{b}^{\mathbf{p}}_{\mathbf{j}} = (1, 4n_v + 3j + 1)$$
$$\mathbf{v}^{\mathbf{p}}_{\mathbf{j}} = (0, 4n_v + 3j + 2)$$

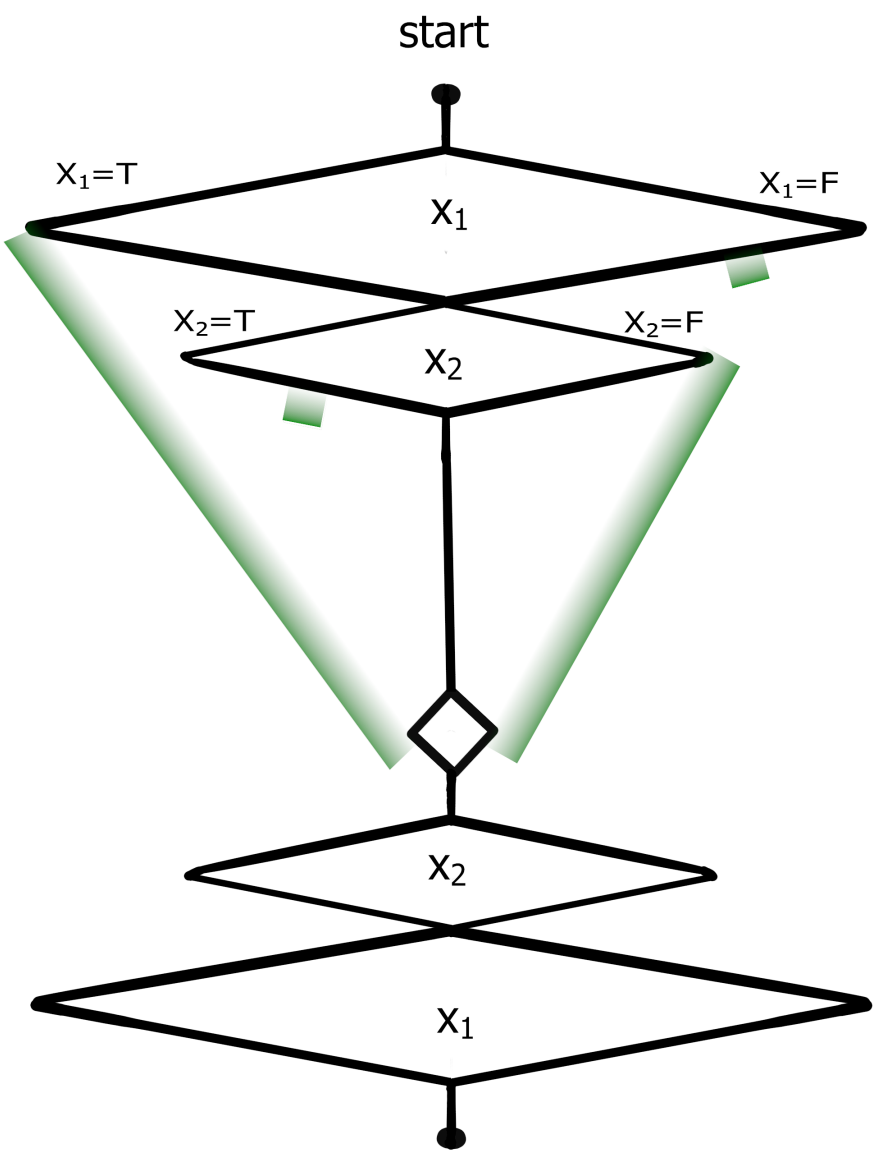

Figure 3-4: An example positive-negative clause gadget, for $X_1 \vee \neg X_2$, illustrating the variable gadget loops (obstacles not shown) and positive-negative clause gadget loops and obstacles. A path assigns a *true* or *false* value to each variable by selecting a branch through the variable gadget loop, thereby risking collision with an obstacle. Then, there is no additional risk for taking the branch through the positive-negative clause gadget loop that corresponds to a satisfied literal. Notice that only low-risk obstacles are used in this gadget. For clarity, we have drawn the B-template obstacles along the edge of the branch rather than at the corner of the branch, as it is constructed in the template.

and edges

$$(\mathbf{u_i^n}, \mathbf{a_i^n})$$
$$(\mathbf{u_i^n}, \mathbf{b_i^n})$$
$$(\mathbf{a_i^n}, \mathbf{v_i^n})$$
$$(\mathbf{b_i^n}, \mathbf{v_i^n}).$$

As before, we also construct edges connecting consecutive loops $(\mathbf{v_i^n}, \mathbf{u_{i+1}^n})$ for all j, an edge from the end of the last variable gadget in the top set to the first positive-negative clause loop $(\mathbf{v_{n_v}^v}, \mathbf{u_0^n})$, and an edge from the last positive-negative clause loop to the first variable gadget in the mirrored set $(\mathbf{v_{n_n}^n}, \mathbf{u_{n_v}^{v\prime}})$.

Each loop gives a planning algorithm two paths corresponding to two literals to try to satisfy. Arbitrarily, for each such positive-negative clause j, let the left path correspond to the positive literal c_{jp}^p and the right path correspond to the negative literal c_{jn}^p. For each one, we construct two obstacles, each near one of the two paths of the corresponding variable gadget loop. However, for the obstacle near the path corresponding to assigning a value to the variable that satisfies the literal, we extend it to also be near the path for this literal in this clause gadget loop.

$$C\left(\mathbf{a_j^p}, \mathbf{a_{c_{jp}^p}^v}\right)$$
$$C\left(\mathbf{b_{c_{jn}^p}^v}, \mathbf{b_j^p}\right)$$

$$B\left(\mathbf{b_{c_{jp}^p}^v}, 1\right)$$
$$B\left(\mathbf{a_{c_{jn}^p}^v}, -1\right)$$

Therefore, taking a path corresponding to a satisfied literal risks collision with just that one obstacle, whereas a path corresponding to a non-satisfied literal risks collision with both obstacles.

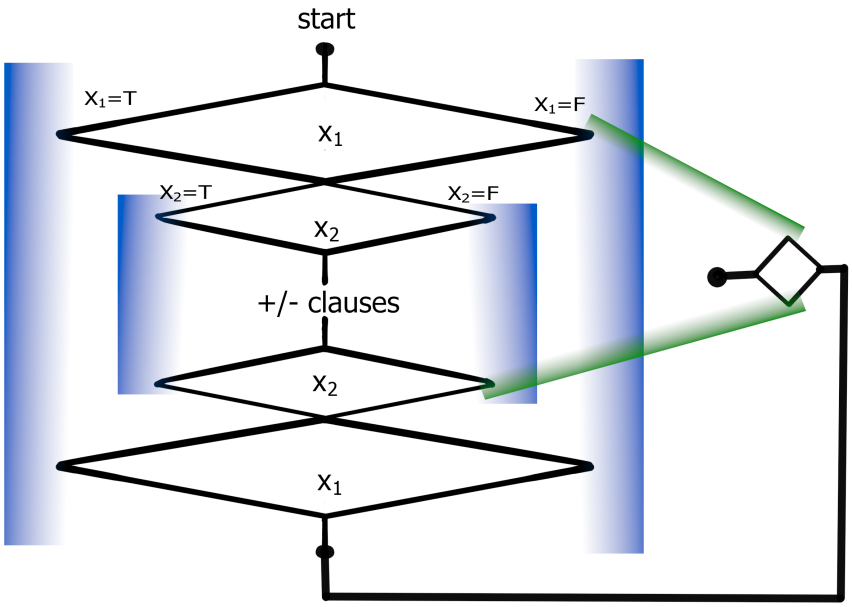

Figure 3-5: An example negative-negative clause gadget, for $\neg X_1 \vee \neg X_2$, illustrating the variable gadget loops and obstacles (blue) and negative-negative clause gadget loops and obstacles (green). A path assigns a *true* or *false* value to each variable by selecting a branch through the variable gadget loop. It must also select the same variable assignment in the mirrored gadgets at the bottom in order to avoid additional collision risk. Then, there is no additional risk for taking the branch through the negative-negative clause gadget loop that corresponds to a satisfied literal.

Two Negative Literals

For each clause j with two negative literals, we construct a loop to the side of the other gadgets, with vertices

$$\mathbf{u_j^n} = (2n_v + 3n_s - 3j, 4n_v)$$
$$\mathbf{a_j^n} = (2n_v + 3n_s - 3j - 1, 4n_v + 1)$$
$$\mathbf{b_j^n} = (2n_v + 3n_s - 3j - 1, 4n_v - 1)$$
$$\mathbf{v_j^n} = (2n_v + 3n_s - 3j - 2, 4n_v)$$

and edges

$$(\mathbf{u_i^p}, \mathbf{a_i^p})$$
$$(\mathbf{u_i^p}, \mathbf{b_i^p})$$
$$(\mathbf{a_i^p}, \mathbf{v_i^p})$$
$$(\mathbf{b_i^p}, \mathbf{v_i^p}).$$

We construct edges connecting consecutive loops $(\mathbf{v_i^p}, \mathbf{u_{i+1}^p})$ for all j. We also construct an intermediate vertex $\mathbf{d} = (2n_v + 3n_s, 7n_v + 3n_p + 2)$ to connect the last variable gadget in the mirrored set to the first negative-negative clause loop with two edges $(\mathbf{v_0^v}, \mathbf{d})$ and $(\mathbf{d}, \mathbf{u_0^p})$.

Each loop gives a planning algorithm two paths corresponding to two literals to try to satisfy. For each such negative-negative clause j, let c_{j1}^n denote the variable specified by the first literal and c_{j2}^n denote the variable specified by the second literal. For each literal, we construct two obstacles, each near one of the two paths of the corresponding variable gadget loop (for the literal corresponding to the top path of the negative-negative clause loop, we will use the top set of variable gadgets, and for the literal corresponding to the bottom path of the negative-negative clause loop, we will use the mirrored set of variable gadgets). However, for the obstacle near the path corresponding to a negative assignment assigning (which satisfies the literal), we extend it to also be near the path for this literal in

this clause gadget loop.

$$C(\mathbf{b}^{\mathbf{v}}_{\mathbf{c}^{\mathbf{n}}_{\mathbf{j1}}}, \mathbf{a}^{\mathbf{n}}_{\mathbf{j}})$$

$$C(\mathbf{b}^{\mathbf{n}}_{\mathbf{j}}, \mathbf{a}^{\mathbf{v\prime}}_{\mathbf{c}^{\mathbf{n}}_{\mathbf{j2}}})$$

$$B(\mathbf{a}^{\mathbf{v}}_{\mathbf{c}^{\mathbf{n}}_{\mathbf{j1}}}, -1)$$

$$B(\mathbf{a}^{\mathbf{v\prime}}_{\mathbf{c}^{\mathbf{n}}_{\mathbf{j2}}}, -1)$$

Therefore, taking a path corresponding to a satisfied literal risks collision with just that one obstacle, whereas a path corresponding to a non-satisfied literal risks collision with both obstacles.

3.1.6 Path Risk Encoding MAXQHORNSAT

We define the $(1+\alpha)$-approximate safe graph search problem as $\phi(G, O, \mathbf{s}, \mathbf{t})$, where G is the set of vertices and edges constructed in the variable and clause gadgets above, O is the set of obstacles constructed in the variable and clause gadgets above, and

$$\mathbf{s} = \mathbf{u}^{\mathbf{v}}_{\mathbf{0}}$$

$$\mathbf{t} = \mathbf{v}^{\mathbf{n}}_{\mathbf{n_s}}.$$

G and O were constructed such that there will exist a gap between the risk of a path corresponding to an optimal assignment and a path corresponding to a suboptimal assignment. Each variable gadget loop in the top set passes near a single high-risk obstacle, and there will exist a path through the mirrored set that does not pass near any additional high-risk obstacles, and passing near an additional high-risk obstacle incurs more risk than passing near every low-risk obstacle, so a minimum-risk path will pass near exactly n_v high-risk obstacles. Any path must also pass near at least one obstacle for each clause gadget, and it will pass near an additional obstacle for each unsatisfied clause, so a minimum-risk path will pass near $(n_s + n_p + n_n + \delta)$ low-risk obstacles, where δ is the minimum number of unsatisfied clauses. Recall that there exists a gap between the induced risk close to an obstacle r_c and far away from the obstacle r_f (or r_{Vc} and r_{Vf} in the case of high-risk

obstacles). Then a path corresponding to an optimal solution to the MAXQHORNSAT problem will incur risk at most

$$n_v r_{Vc} + n_v r_{Vf} + (n_c + \delta) r_c + (3n_c - \delta) r_f$$

and a path corresponding to a suboptimal solution to the MAXQHORNSAT problem will incur risk at least

$$n_v r'_{Vc} + n_v r'_{Vf} + (n_c + \delta + 1) r'_c + (3n_c - \delta - 1) r'_f.$$

This allows us to compute a lower bound on the ratio of the risks between a suboptimal solution and an optimal solution as

$$\begin{aligned}
&\frac{n_v r'_{Vc} + n_v r'_{Vf} + (n_c + \delta + 1) r'_c + (3n_c - \delta - 1) r'_f}{n_v r_{Vc} + n_v r_{Vf} + (n_c + \delta) r_c + (3n_c - \delta) r_f} \\
=&\frac{n_v r'_{Vc} + (n_c + \delta + 1) r'_c}{n_v r_{Vc} + n_v r_{Vf} + (n_c + \delta) r_c + (3n_c - \delta) r_f} \\
=&\frac{5 n_c n_v r'_c + (n_c + \delta + 1) r'_c}{5 n_c n_v r_c + 5 n_c n_v r_f + (n_c + \delta) r_c + (3n_c - \delta) r_f} \\
\geq&\left(\frac{r'_c}{r_c \beta} + \frac{r'_c}{20 n_c n_v r_c} \right) + \frac{r'_c}{20 n_c n_v r_c} \\
\geq&1 + \theta \left(\frac{1}{n^2} \right)
\end{aligned}$$

if we set the obstacle constants such that $20 n_c n_v + \beta \geq 20 n_c n_v \beta$, where $\beta = 1 + 3\frac{r_f}{r_c}$

Each gadget can be constructed in polynomial time, and the number of gadgets is polynomial, so this reduction can be constructed in polynomial time. Thus, any algorithm that can approximate the minimum-risk planning problem in a graph to a factor better than $1 + \Theta(\frac{1}{n^2})$ can also solve MAXQHORNSAT with polynomial overhead. $\square$

3.1.7 Hardness of Continuous Planning Problem

We prove Theorem 7 by extending the above reduction to still apply even without the graph restriction (thereby reducing to $(1 + \alpha)$-approximate minimum-risk planning). Given the

graph G and set of obstacles O constructed above, we surround G with additional obstacles such that a path cannot deviate from G by more than $2\epsilon_P$, for some small constant ϵ_P. We divide the space of $\mathbb{R}^2$ into a grid with cells of size $\epsilon_P \times \epsilon_P$ and construct a square obstacle in every cell that does not intersect with the graph and is within a window containing all of the gadgets.

$$C_{ij} = \left\{ \mathbf{x} \;\middle|\; \begin{array}{l} \mathbf{x} \in \mathbb{R}^2 \\ i\epsilon_P \leq \mathbf{e_1}^T \mathbf{x} \leq (i+1)\epsilon_P \\ j\epsilon_P \leq \mathbf{e_2}^T \mathbf{x} \leq (j+1)\epsilon_P \end{array} \right\}$$

$$\text{for all } i, j \in \mathbb{Z}$$

$$O' = O \cup \left\{ C_{ij} \;\middle|\; \begin{array}{l} i, j \in \mathbb{Z} \\ -\frac{1}{\epsilon_P} 2n_v \leq i, j \leq \frac{1}{\epsilon_P}(8n_v + 4n_p) \\ C_{ij} \cap G = \emptyset \end{array} \right\}$$

Note that there are a polynomial number of such obstacles, so if ϵ_P is sufficiently small, the solution to the resulting $(1+\alpha)$-approximate minimum-risk planning problem or lack thereof is approximately equivalent to that for the original $(1+\alpha)$-approximate safe graph search problem, and so $(1+\alpha)$-approximate minimum-risk planning is also NP-hard. □

3.2 Hardness with Constraints on Overlapping Obstacles

Hauser [18] observed that his 3D MCR reduction as well as the 2D MCR reduction presented by Erickson and LaValle [13] required that each obstacle be allowed to overlap with $O(n)$ other obstacles. Similarly, we note that the reduction we present above for 2D motion planning under obstacle uncertainty also requires that each obstacle overlap with $O(n)$ other obstacles. This is a relatively unnatural problem instance, as most real-world problems will not have this degree of overlap. In this section, we show that the problem remains hard in 3D even when each obstacle only overlaps with a constant number of other obstacles.

3.2.1 3SAT

3SAT is an NP-complete problem that is commonly used to prove the hardness of other problems [40]. The problem input is a Boolean formula given in conjunctive normal form, where each clause consists of three literals, or in other words, it is of the form $((x_0 \vee \neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge \ldots)$. The algorithm must then decide whether there exists any variable assignment that satisfies the formula. We will consider a 3SAT problem with k variables $x_0, x_1, \ldots$ and m clauses, where each clause j is of the form $(x_{j_u} \vee \neg x_{j_v} \vee x_{j_w})$.

3.2.2 Proof Outline

We prove Theorem 8 using a reduction from 3SAT. Given a 3SAT instance, we construct a κ-overlap $(1 + \alpha)$-approximate minimum-risk planning problem as follows.

1. Construct a set of variable assignment layers where each branch corresponds to a variable assignment.

2. Construct a set of clause layers where each branch corresponds to selecting a literal to satisfy.

3. For each variable assignment layer, construct a pair of obstacles for each variable that will encode whether the variable is set to *true* or *false*. There will be additional collision risk for a path that selects different values in each variable assignment layer, as well as for a path that selects a literal that is not satisfied by the value selected in the preceding variable assignment layer.

The solution to the planning problem can then be transformed into a solution to the 3SAT instance in polynomial time, demonstrating that the κ-overlap $(1+\alpha)$-approximate minimum-risk planning problem is at least as hard as 3SAT.

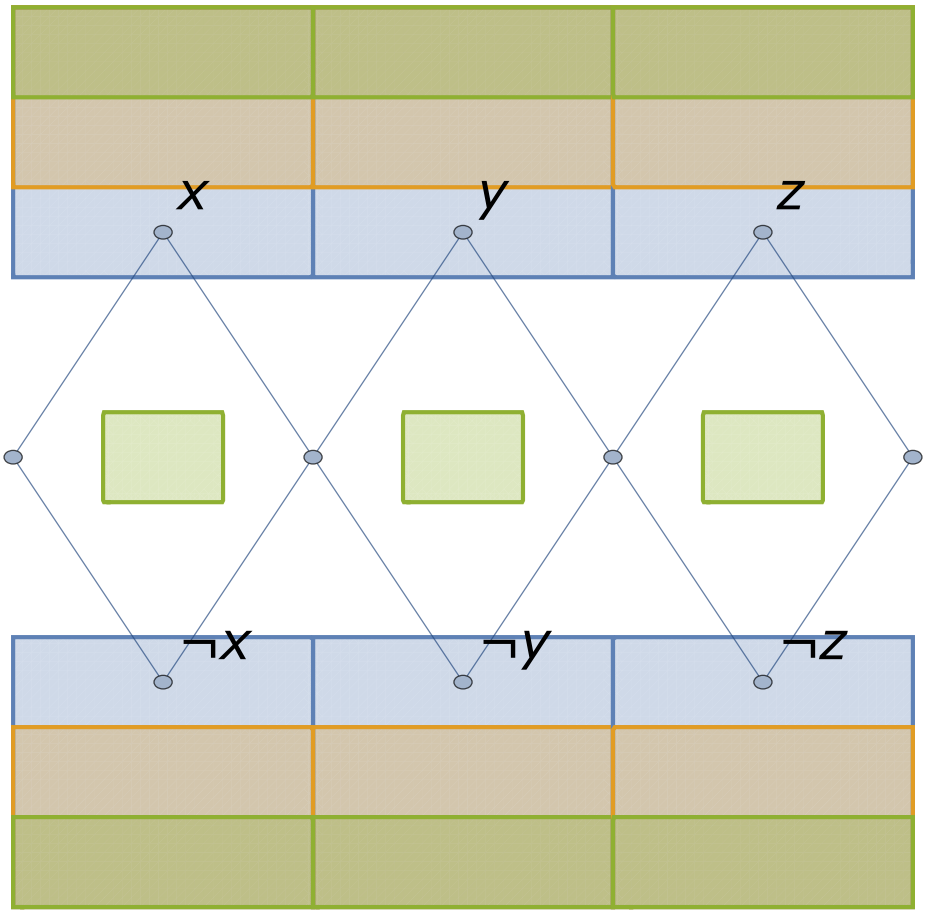

Figure 3-6: A path through this gadget must go near either the *true* or *false* obstacle for each variable, thereby selecting a variable assignment.

3.2.3 Proof

Variable Gadgets

First, we will construct a variable assignment layer for each clause j. A variable assignment layer consists of two PGDF obstacles for each variable i in the 3SAT problem.

Note that we define a PGDF obstacle as the intersection of halfspaces of the form $\alpha^T x \leq 0$ for α normally distributed and x represented in homogeneous coordinates. Here we will work with just one face and standard coordinates for convenience. That is, each obstacle i will be defined as

$$o = \left\{ \mathbf{x} \;\middle|\; {\alpha_\mathbf{i}}^T \mathbf{x} \leq 1, \alpha_\mathbf{i} \sim \mathcal{N}(\mu_\mathbf{i}, \mathbf{\Sigma_i}) \right\}.$$

For obstacle i, the *true* obstacle will be defined as the intersection of

$$\begin{aligned} {\alpha_\mathbf{i}}^T \mathbf{x} &\leq 1 \\ i \leq {\mathbf{e_2}}^T \mathbf{x} &\leq i+1 \\ 2j - \frac{3}{2} < {\mathbf{e_3}}^T \mathbf{x} &\leq 2j + \frac{3}{2} \end{aligned}$$

where $\alpha_\mathbf{i} \sim \mathcal{N}(2\mathbf{e_1}, \mathbf{e_1}{\mathbf{e_1}}^T)$. The "negative" obstacle will similarly be defined with

$$\begin{aligned} {\beta_\mathbf{i}}^T \mathbf{x} &\leq 1 \\ i \leq {\mathbf{e_2}}^T \mathbf{x} &\leq i+1 \\ 2j - \frac{3}{2} < {\mathbf{e_3}}^T \mathbf{x} &\leq 2j + \frac{3}{2} \end{aligned}$$

where $\beta_\mathbf{i} \sim \mathcal{N}(-2\mathbf{e_1}, \mathbf{e_1}{\mathbf{e_1}}^T)$.

Intuitively the covariance $\mathbf{e_1}{\mathbf{e_1}}^T$ means that $\alpha_\mathbf{i}$ has variance 1 in the direction of the normal of the face. This is important because it means that there is no variance in the orientation of the face. Also note that each obstacle overlaps with the corresponding obstacle in the layer below and the layer above, which we will later show to be important in ensuring that variable assignments are consistent across layers.

Then we will construct the variable assignment graph, as illustrated in figure 3-6. Said

formally, indexing over the variable with index i, we embed nodes in locations

$$(2j-1)\mathbf{e_3} + i\mathbf{e_2}$$
$$(2j-1)\mathbf{e_3} + (i+\frac{1}{2})\mathbf{e_2} \pm \mathbf{e_1}$$
$$(2j-1)\mathbf{e_3} + (i+1)\mathbf{e_2}.$$

We then draw edges from $(2j-1)\mathbf{e_3} + i\mathbf{e_2}$ to both of $(2j-1)\mathbf{e_3} + (i+\frac{1}{2})\mathbf{e_2} \pm \mathbf{e_1}$, and from both of $(2j-1)\mathbf{e_3} + (i+\frac{1}{2})\mathbf{e_2} \pm \mathbf{e_1}$ to $(2j-1)\mathbf{e_3} + (i+1)\mathbf{e_2}$.

Clause Gadgets

For each clause j we will construct an additional graph "layer" in between consecutive pairs of variable layers that lets the algorithm choose which literal to satisfy, as illustrated in figure 3-7.

Recall that each clause j is of the form $x_{j_u} \vee \neg x_v \vee x_w$. Without loss of generality, let $j_u < j_v < j_w$. Indexing over j, construct nodes at

$$2j\mathbf{e_3}, 2j\mathbf{e_3} + \left(j_u + \frac{1}{2}\right)\mathbf{e_2}$$
$$\left(2j + \frac{1}{3}\right)\mathbf{e_3} + \left(j_u + \frac{1}{2}\right)\mathbf{e_2} \pm \mathbf{e_1}$$
$$\left(2j + \frac{2}{3}\right)\mathbf{e_3} + \left(j_u + \frac{1}{2}\right)\mathbf{e_2}$$
$$\left(2j + \frac{2}{3}\right)\mathbf{e_3}$$

drawing edges between consecutive nodes, and letting '$\pm$' represent '-' if x_{j_u} is given in

negated form and '+' otherwise. Then construct nodes at

$$
\begin{aligned}
&2j\mathbf{e_3} + \left(j_u + \frac{1}{2}\right)\mathbf{e_2} \\
&2j\mathbf{e_3} + \left(j_v + \frac{1}{2}\right)\mathbf{e_2} \\
&\left(2j + \frac{1}{3}\right)\mathbf{e_3} + \left(j_v + \frac{1}{2}\right)\mathbf{e_2} \pm \mathbf{e_1} \\
&\left(2j + \frac{2}{3}\right)\mathbf{e_3} + \left(j_v + \frac{1}{2}\right)\mathbf{e_2} \\
&\left(2j + \frac{2}{3}\right)\mathbf{e_3} + \left(j_u + \frac{1}{2}\right)\mathbf{e_2}
\end{aligned}
$$

(the first and last were already constructed previously), drawing edges between consecutive nodes, and similarly setting '$\pm$' based on the negation of literal x_{j_v}. Then construct nodes at

$$
\begin{aligned}
&2j\mathbf{e_3} + \left(j_v + \frac{1}{2}\right)\mathbf{e_2} \\
&2j\mathbf{e_3} + \left(j_w + \frac{1}{2}\right)\mathbf{e_2} \\
&\left(2j + \frac{1}{3}\right)\mathbf{e_3} + \left(j_w + \frac{1}{2}\right)\mathbf{e_2} \pm \mathbf{e_1} \\
&\left(2j + \frac{2}{3}\right)\mathbf{e_3} + \left(j_w + \frac{1}{2}\right)\mathbf{e_2} \\
&\left(2j + \frac{2}{3}\right)\mathbf{e_3} + \left(j_v + \frac{1}{2}\right)\mathbf{e_2}
\end{aligned}
$$

(the first and last were already constructed previously), drawing edges between consecutive nodes, and similarly setting '$\pm$' based on the negation of literal x_{j_w}. Intuitively, this creates three possible routes through the graph, each going near the obstacle corresponding to a particular value assigned to a variable.

A path through this gadget must pick one of the literals in the clause to satisfy and pass near the obstacle that corresponds to that variable and the value the literal requires it to have. In doing so, it may incur risk of intersecting with the obstacle. If this variable was assigned to the value the literal specifies, then the path would have already gone near this obstacle so no further risk is incurred. However, if the literal contradicts the variable assignment, the path will incur additional risk for going near this obstacle.

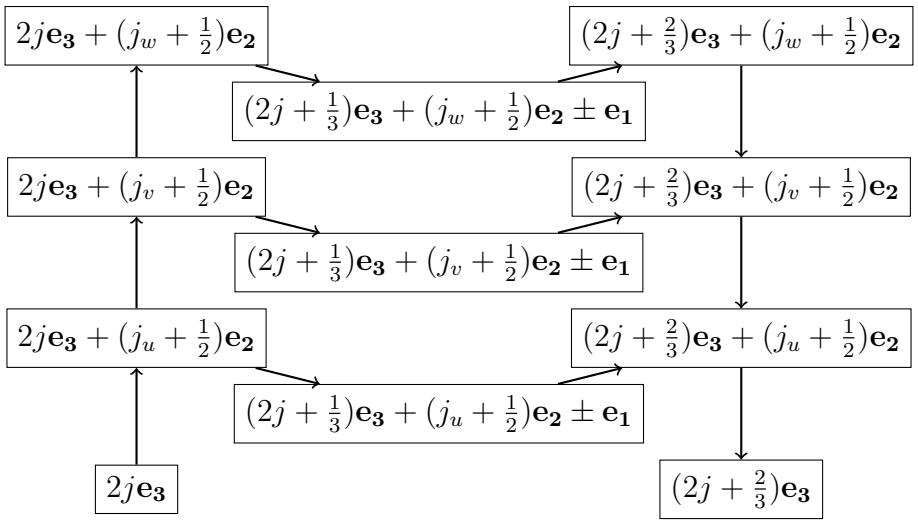

Figure 3-7: A path through this gadget must select one of three paths to go through, each going near the obstacle for the corresponding literal.

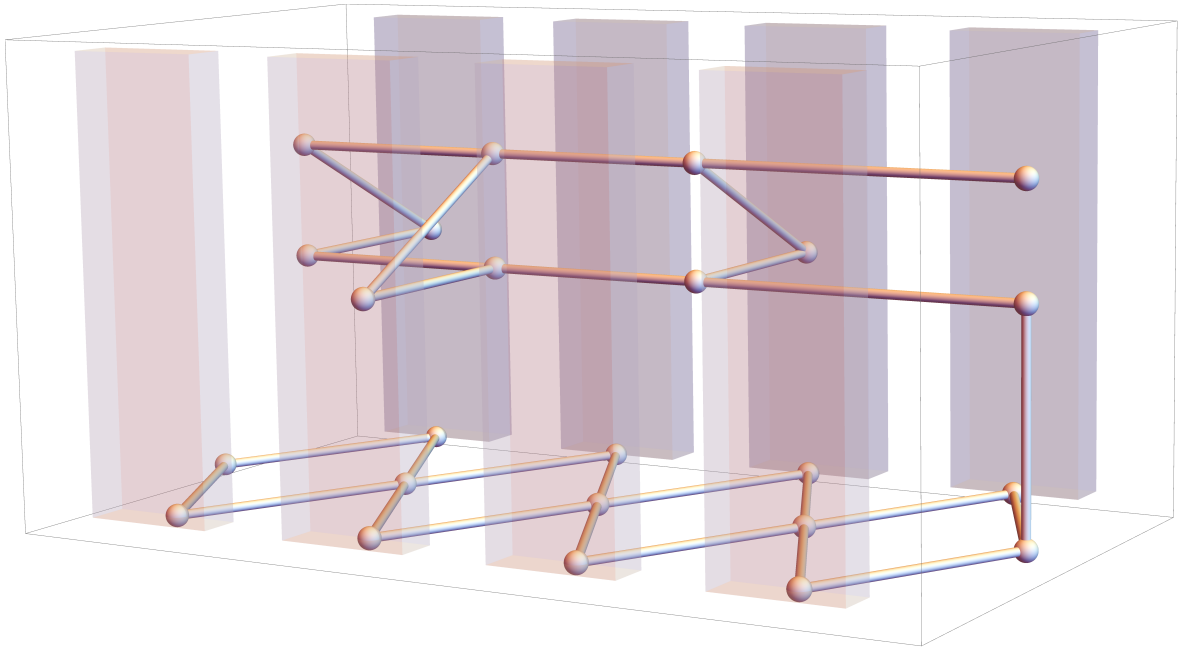

Figure 3-8: The bottom layer is the first variable assignment layer. The top layer is the first clause gadget. There would usually be many more clause gadgets stacked on top with additional variable gadget layers in between.

Full Reduction

Now we combine the variable and clause gadgets, as seen in figure 3-8. As in Section 3.1.7, we construct a grid of deterministic obstacles to force the path to remain near the graph. Given the graph G and set of obstacles O constructed above, we surround G with additional obstacles such that a path cannot deviate from G by more than $2\epsilon_P$, for some small constant ϵ_P. We divide the space of $\mathbb{R}^{\not\vdash}$ into a grid with cells of size $\epsilon_P \times \epsilon_P \times \epsilon_P$ and construct a cubic obstacle in every cell that does not intersect with the graph and is within a window containing all of the gadgets.

$$C_{\mathbf{z}} = \left\{ \mathbf{x} \;\middle|\; \begin{array}{l} \mathbf{x} \in \mathbb{R}^3 \\ \\ z_i \epsilon_P \leq x_i \leq (z_i + 1)\epsilon_P \;\forall\; i \in \{1, 2, 3\} \end{array} \right\}$$

$$\text{for all } \mathbf{z} \in \mathbb{Z}^3$$

$$O' = O \cup \left\{ C_{\mathbf{z}} \;\middle|\; \begin{array}{l} z \in \mathbb{Z}^3 \\ -\frac{4}{\epsilon_P} \leq \frac{1}{\sqrt{3}}|\mathbf{z}| \leq \frac{1}{\epsilon_P}(k + m + 4) \\ C_{\mathbf{z}} \cap G = \emptyset \end{array} \right\}$$

Path Risk Encoding 3SAT

This graph was constructed such that there will exist a gap between the risk of a satisfying assignment and of a non-satisfying assignment. First we note that for the PGDF obstacle model as well as most reasonable alternative formulations, there exists a gap between the induced risk close to the obstacle and far away from the obstacle. In particular, there is some r_c that lower-bounds the risk computed from the shadow approximation for the closer points and r_f that upper-bounds the computed risk for the further point. A path through the variable assignment portion of the graph will go near km obstacles for the k variable assignments it makes, each repeated m times. Then it will be "close" to km obstacles and "far" from the other km obstacles. Therefore, it will incur risk $kmr_c + kmr_f$.

If a path through the variable assignment portion encodes a satisfying assignment to the 3SAT problem, there will exist a path through the remainder of the graph that will not incur any additional cost. If there is no satisfying assignment, then any path through the

remaining portion must go near an obstacle that it did not go near in the variable assignment portion, so for some variable i, the optimal path must go close to both the *true* and *false* obstacles, incurring cost at least $(km+1)r_c + (km-1)r_f$. This allows us to compute a lower bound on ratio between the two risks:

$$\begin{aligned}
&\frac{(km+1)r_c + (km-1)r_f}{kmr_c + kmr_f} \\
=&\frac{kmr_c + kmr_f + r_c - r_f}{kmr_c + kmr_f} \\
=&1 + \frac{r_c - r_f}{kmr_c + kmr_f} \\
=&1 + \Theta\left(\frac{1}{km}\right).
\end{aligned}$$

We note that each obstacle only overlaps with at most a constant number of other obstacles. In particular, each obstacle will overlap with the two corresponding obstacles in the layer above and the layer below, as well as the constant number of deterministic obstacles forming dividers between layers.

Each gadget can be constructed in polynomial time, and the number of gadgets is polynomial, so this reduction can be constructed in polynomial time. Thus any algorithm that can approximate the κ-overlap $(1+\alpha)$-approximate minimum-risk planning problem (regardless of whether a graph containing the solution is provided) to a factor better than $1 + \Theta\left(\frac{1}{km}\right)$ can also solve 3SAT with polynomial overhead. □

Extension for Minimum Constraint Removal

This reduction can also be extended to apply to the Minimum Constraint Removal (MCR) Problem, proving Theorem 9 and answering the question posed by Hauser [18]. We replace each uncertain obstacle in the ϵ-safe planning problem with an MCR obstacle covering the r_c-shadow of the obstacle. As before, a path corresponding to a satisfying assignment will collide with km obstacles, whereas a path corresponding to a nonsatisfying assignment must collide with at least $km+1$ obstacles. Then the ratio between the costs of a nonsatisfying

path and a satisfying path is lower bounded by

$$\frac{km+1}{km} = 1 + \Theta\left(\frac{1}{km}\right).$$

Therefore, MCR remains NP-hard even when each obstacle is connected and intersects no more than κ obstacles (κ held constant).

Chapter 4

Fixed-parameter Algorithm

4.1 Parameterized Tractability

When a problem is found to be intractable in general, the natural next step is to ask if there are any special cases that might be tractable. We have already shown that the case where each obstacle overlaps with at most a constant number of other obstacles remains hard. Instead, we look at the underlying factor that appeared to be caused by obstacle overlap: obstacle reentry. Both the 3SAT and MAXQHORNSAT reductions rely on the robot going near the same obstacle multiple times, at distant parts of the trajectory. In particular, we see that the robot must pass near an unbounded number of other obstacles in the intervening time. This number is captured by the *collision horizon* parameter. We find that this parameter drives the hardness of the problem, and so when the collision horizon is fixed, the problem becomes tractable.

4.2 Algorithm

In this section, we present a polynomial time h-horizon minimum-risk graph-search algorithm. The algorithm is generally structured similarly to $A*$ search [16] minimizing collision risk, but with the state augmented to include the collision memory in addition to the current vertex. The collision memory can be thought of as the maximum reached for each obstacle in order to get to the current state. Then, the visited set contains all visited

states for each vertex. When deciding whether to skip a state, instead of just checking whether the exact state has previously been visited, it instead checks whether every set of at most h remembered collisions are contained in the collision memory of some previously visited state at this vertex. This is necessary in order to bound the number of states visited. Pseudocode for this algorithm is provided in Algorithm 2. With k obstacles, L risk levels, and E edges, it can visit at most $(kL)^h$ unique states for each vertex, so it can query at most $E(kL)^h$ edges. Since each edge can take $O\left(\binom{k}{h}(kL)^h k\right)$ to check whether it can be skipped, the overall algorithm runs in time $O\left(\binom{k}{h}(kL)^{2h} hEk \log VkL\right)$, which is polynomial when h is fixed.

Algorithm 2 MEMORY_SEARCH (M_h)

Input: Graph $G = (V, E)$, obstacles O, end points $\mathbf{s}, \mathbf{t}$, and collision horizon h.
Output: A trajectory from $\mathbf{s}$ to $\mathbf{t}$ through G.

visited = {}
queue = PriorityQueue({$(\mathbf{s}, [], \{\})$})
while not queue.empty() **do**
  $\mathbf{u}, \tau, C$ = queue.pop()
  // return if reached goal
  if $\mathbf{u} = \mathbf{t}$ **then**
    return τ
  end if
  // check whether every subset of M of size at most h
  // is contained in the memory of some visited state
  if $\exists M \subseteq C \; s.t. \; (|M| \leq h$ and $\neg\exists \; C'' \in$ visited$[u] \; s.t. \; M \subseteq C'')$ **then**
    visited.insert(u, C)
    // add each outgoing edge to the queue
    for $\mathbf{v} \in E[\mathbf{v}]$ **do**
      $C' = \{(o, \max(C[o], f_o((u, v))) \text{ for each } o \in O\}$
      queue.insert($\sum_{o \in O} C'[o]$, $(\mathbf{v}, \tau + (\mathbf{u}, \mathbf{v}), C')$)
    end for
  end if
end while

Intuitively, the algorithm is performing an exact graph search that prunes states that can only be useful if the collision horizon is larger. The augmented state with the remembered collisions is sufficient to fully predict the marginal cost of future states. However, the potential number of such states is exponentially large in the collision horizon. We know that a state with a higher cost than an existing state for the same vertex can only be part of a lower-risk path if the remembered collisions reduce future marginal risk due to correlated

collisions. But if the collision horizon is h, then only h collisions must be remembered in order to describe the state. One approach would be to restrict the collision memory to h collisions. Due to the large branching factor stemming from the choice of which collision to forget, we do not limit the collisions we record. Instead we simply check whether *any* subset of size at most h is not already contained by a previously visited state, thereby treating each state as a collection of states with memory size h. If the obstacles have multiple risk levels, the new history is kept only when it has at least h obstacles with a lower risk level than each of the previous visits to the current node. This algorithm correctly computes the cost of any trajectory segment that does not incur correlated collision risk at points separated by more than h other obstacles. Thus, the following theorem bounding the suboptimality of the algorithm holds.

Theorem 10. *Let ϵ be the associated cost of the trajectory generated by $M_h(G, O, \mathbf{s}, \mathbf{t})$ and let T_* be the set of optimal trajectories, with associated cost ϵ_*. Then*

$$\epsilon \leq \sum_{o \in O, \tau_i \in S_o^{(\tau_*)}} f_o(\tau_i) \leq \epsilon_* + \sum_{o \in O} (|S_o^{(\tau_*)}| - 1) f_o(\tau_*)$$

where $S_o^{(\tau_*)}$ is the trajectory τ_* split into the fewest segments such that for each $\tau_i \in S_o^{(\tau_*)}$, $H_o^{(\tau_i)} \leq h$.

Proof. Let ϵ be the associated cost of the trajectory generated by $M_h(G, O, \mathbf{s}, \mathbf{t})$ and let T_* be the set of optimal trajectories, with associated cost ϵ_*. Because each obstacle is distributed independently from other obstacles, we begin by considering the risk incurred by each one separately. For a given obstacle o, suppose $S_o^{(\tau_*)}$ is the trajectory τ_* split into the fewest segments such that for each $\tau_i \in S_o^{(\tau_*)}$, $H_o^{(\tau_i)} \leq h$. For each time τ_i enters a risk level with edge (u, v), the planning tree generated by M_h must contain a state $\hat{s}_u$ at vertex u with memory containing the preceding h collisions in τ_i since such a state is reachable (given that τ_i reaches it) and would not be skipped unless another previously visited state at u already contained the preceding h collisions. Because $H_o^{(\tau_i)} \leq h$, we know that the preceding h collisions are sufficient to determine the marginal risk of each collision. Then M_h will at some point expand edge $(\hat{s}_u, \hat{s}_v)$, where $\hat{s}_v$ is the state at vertex v still with

memory containing the preceding h collisions in τ_i and with cost from o no more than $f_o(\tau_i)$. Hence, M_h computes the marginal risk of this obstacle for this subtrajectory correctly. Then the total computed risk from obstacle o for trajectory τ_* is at most

$$\sum_{\tau_i \in S_o^{(\tau_*)}} f_o(\tau_i)$$

Hence, the algorithm would assign the overall risk of trajectory τ_* as at most

$$\widetilde{\epsilon} \leq \sum_{o \in O, \tau_i \in S_o^{(\tau_*)}} f_o(\tau_i)$$

Because M_h greedily expands nodes in order of computed cost, it would only select a different trajectory if its computed cost $\hat{\epsilon} \leq \widetilde{\epsilon}$. We know that M_h can only overestimate the cost of a trajectory (due to not taking into account the optimal set of past collisions), not underestimate, so the cost of the trajectory it returns is at most $\hat{\epsilon}$. Finally, since $f_o(\tau_*) = \max_{\tau_i \in S_o^{(\tau_*)}} f_o(\tau_i)$ and

$$\epsilon_* = \sum_{o \in O} f_o(\tau_*) = \sum_{o \in O} \max_{\tau_i \in S_o^{(\tau_*)}} f_o(\tau_i)$$

we are left with the following bound:

$$\epsilon \leq \hat{\epsilon} \leq \widetilde{\epsilon} \leq \epsilon_* + \sum_{o \in O} (|S_o^{(\tau_*)}| - 1) f_o(\tau_*)$$

□

And the following consequence holds when the collision horizon is fixed.

Theorem 11. *M_h returns an optimal trajectory when the collision horizon of the problem is less than h.*

Thus, the problem is tractable when the collision horizon is small, and when it is unbounded, we have an algorithm that produces a solution whose suboptimality is limited by the how much the optimal trajectory exceeds a collision horizon of h. This bound is especially

helpful for domains where obstacle extents have infinite support, because while the collision horizon of these problems can be fairly large due to significant overlap of shadows, they impact the quality of the solution by the amount of change in risk level only over the areas that interact with more than h other obstacles, and typically only the large, low-risk shadows interact with large numbers of obstacles. Hence, we can still find a near-optimal solution.

Some examples of the behavior of this algorithm on different kinds of problems are in Figure 4-1.

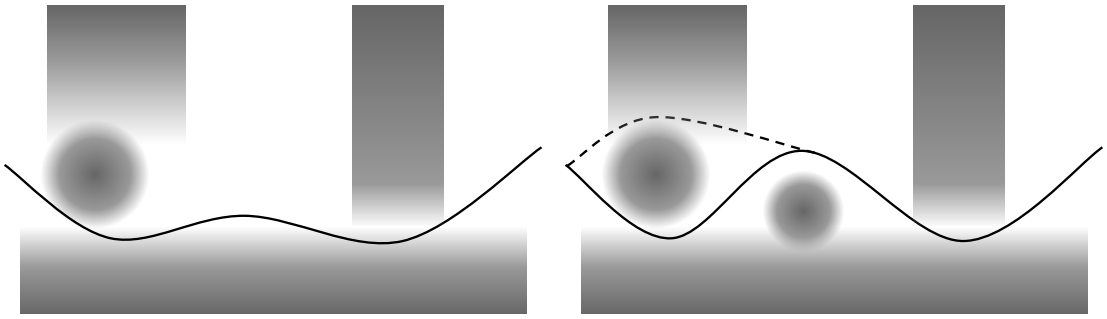

Figure 4-1: An example where M_0 is optimal (left) and one where it is suboptimal (right). The shading indicates the obstacle shadows, so the probability a trajectory collides with a given obstacle is given by the darkness of the maximally shaded point it goes through. In both cases the optimal trajectory is the solid black line, which risks collision with the long obstacle at the bottom – note that even though the trajectory on the right risks collision with the bottom obstacle twice, these collisions are correlated (if the first "dip" is in collision, then so is the second, and vice versa), so the overall collision risk is the same as on the left. However, in the suboptimal case, M_0 will instead pick the dotted subtrajectory, because by itself it is safer than the corresponding subpath of the optimal trajectory. Hence, this problem instance has collision horizon 1, and so M_1 will solve it correctly.

4.2.1 Application to MCR

We would now like to consider the related problem of *minimum constraint removal*, or the problem of finding a plan that collides with the smallest number of obstacles. This problem was studied by Hauser [18], who presented two algorithms for solving it. The first is an exact solver, which retains the set of past collisions in the state space of the search, leading to optimal solutions but a potentially exponentially large search space. The other algorithm is a greedy solver, which is restricted to visiting each vertex at most once. As a result, the greedy method is faster, but can be suboptimal in certain cases.

We so far have described an algorithm for planning under obstacle uncertainty. However,

as noted by Hauser [18] and by Shimanuki and Axelrod [39], there appear to be strong connections between planning under obstacle uncertainty and minimum constraint removal. In fact, a minimum constraint removal problem can be described as a planning under obstacle uncertainty problem, where each obstacle only has a single and a fixed cost for colliding with it. Intuitively, this can be thought of as treating the objective of minimizing collisions as equivalent to the objective of minimizing collision risk when each obstacle has some fixed probability of existing.

As such, our algorithm can also be used to solve minimum constraint removal problems. Moreover, M_0 is equivalent to the greedy algorithm proposed by Hauser [18], and M_∞ is equivalent to his exact algorithm. Thus, the collision horizon parameter of our algorithm can be seen as interpolating between the greedy and exact algorithms, providing a tradeoff between optimality and runtime based on the application.

4.3 Empirical Results

We now present experimental results comparing our algorithm to various baselines as well as illustrating how the collision horizon parameter influences behavior. We consider two baselines: First, we implemented the naive and commonly used approach of simply setting all the shadows to be equal, often referred to as constructing buffers, and then running a normal motion planner on it. Our implementation iteratively adapts the buffer size to select the smallest risk level that allows a solution. Second, we compare to a method proposed by Hauser [18] which samples obstacle instances from the distribution and then uses an approximate minimum constraint removal planner to find the path that collides with the fewest sampled obstacles. Note that this approach does not guarantee safe plans because a trajectory constructed conditioned on a specific sample of obstacle instances can still be likely to collide with an independent sample of obstacle instances. As such, we slightly modify it to construct each risk level as an individual obstacle rather than sampling actual obstacle instances. This ensures the soundness of the algorithm while leading to a runtime improvement due to needing fewer obstacles, that leads to a fair comparison. We note that in order to guarantee a low risk a very large number of samples is required.

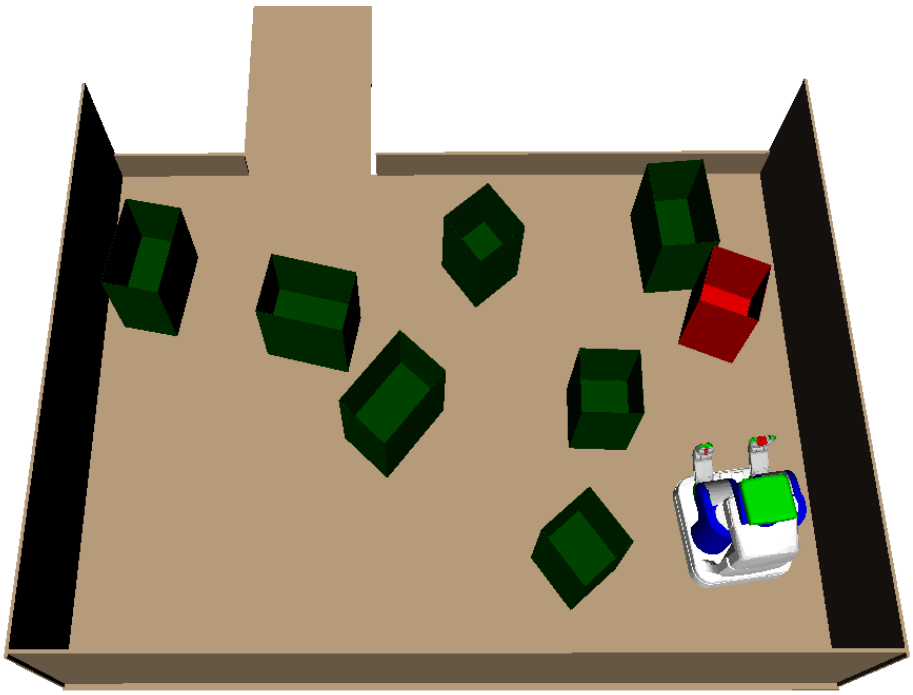

Figure 4-2: The robot's task is to pick up the red box and carry it out of the room through the hallway at the top. The green boxes (of which there are between 8 and 24) are obstacles with known position but Gaussian distributed extents. This is then discretized into 3 risk levels.

4.3.1 Moving Boxes Domain

The first domain we evaluate our algorithm on is a pick-and-place motion planning problem among uncertain obstacles. An example problem instance is depicted in Figure 4-2.

We sample approximately 600 robot base poses and draw edges to form a graph embedded in the configuration space of the robot base. We then sample up to 4 feasible grasps, each of which creates an edge to a copy of the original graph (a copy is necessary because the collisions at each node are different based on whether the robot is holding a box and how it is grasping it). The performance of each method on this domain is compared in Figure 4-3.

We also measured the effect of the collision horizon on these performance metrics, depicted in Figure 4-4.

We find that our algorithm is already near-optimal at $h = 0$, and the gap becomes entirely closed with $h = 1$. This suggests that this domain tends to have a very small collision

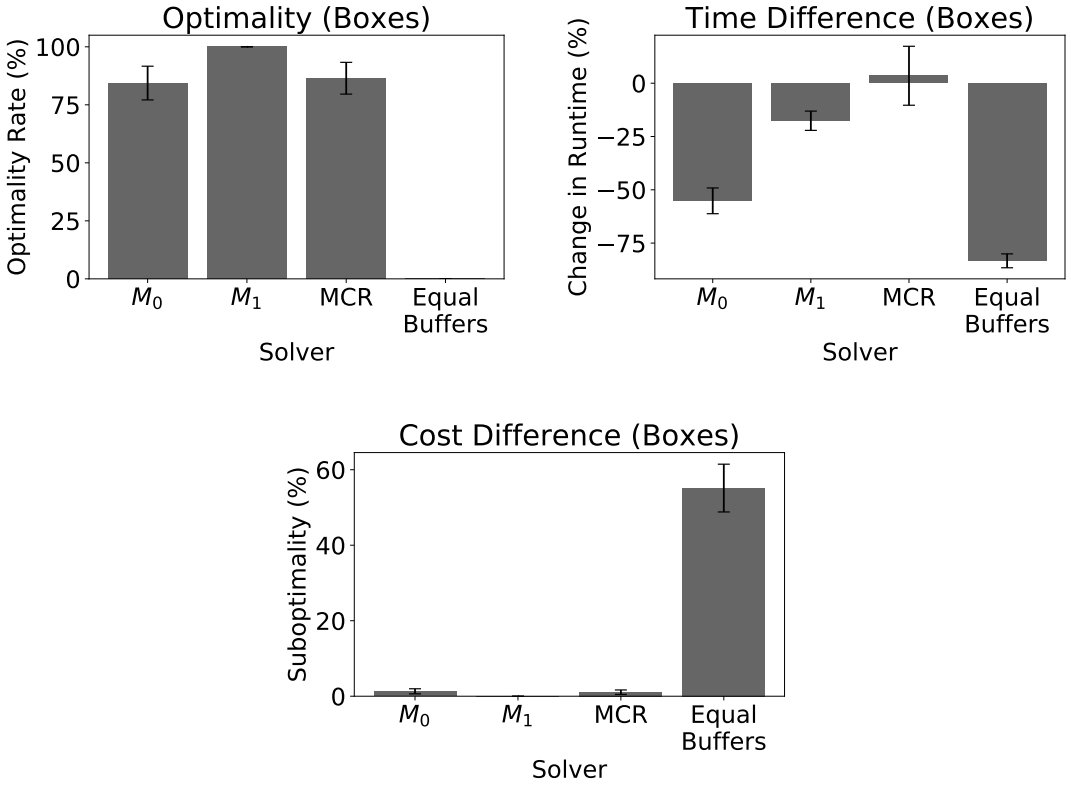

Figure 4-3: The optimality rate (percent of problem instances where the algorithm returns an optimal solution), runtime, and planning risk of each method. Runtime and cost are depicted as the difference compared to the optimal planner M_{24} to control for the variance in difficulty of different problem instances.

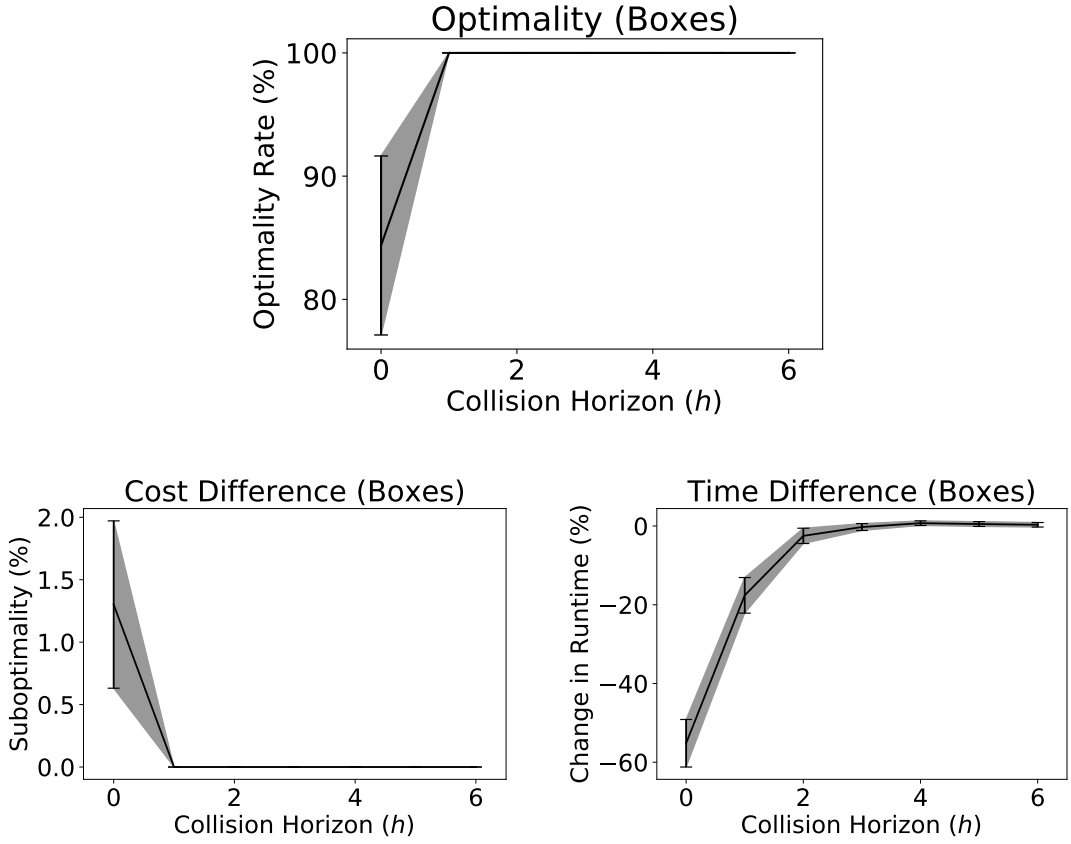

Figure 4-4: The optimality, runtime, and planning risk of M_h for each collision horizon. Runtime and cost are depicted as the difference compared to the optimal planner M_{24} to control for the variance in difficulty of different problem instances. Increasing the collision horizon past 6 up to 24 shows no noticeable change in behavior, so we have cropped the graphs for clarity.

horizon. The obstacle-sampling approach behaves similarly to M_0, which is unsurprising since the minimum constraint removal planner used is equivalent to M_0. However, it runs slower, likely because it has to handle a larger number of sampled obstacles instead of reasoning about the differents directly. Setting the shadows to be equal is a fast solution, but is very suboptimal. We suspect this is because due to the crowdedness of the domain, the robot only actually risks collision with a fraction of the obstacles, and so expanding all the shadows by an equal amount is overly conservative. Overall, M_1 appears to be the most generally attractive option, as it is optimal in every instance and runs slightly faster than the exact search.

4.3.2 Driving Domain

We also evaluate our method on a driving domain. In this problem, the vehicle is attempting to make an unprotected left turn, and there is both cross and oncoming traffic. An example of this domain is depicted in Figure 4-5.

The geometric curve the vehicle will follow is fixed, but the vehicle has the option to proceed forward or wait at each timestep. Hence, the graph is a 2D lattice where one dimension is progress along the curve (40 steps) and the other dimension is time (100 timesteps). This graph is fed as input to the graph search algorithms, each of which returns a trajectory that indicates when the robot should be moving and when it should be stopping. Practically speaking, a solution trajectory makes two choices: which vehicles to cut between when entering the intersection, and which vehicles to cut between when leaving the intersection. The performance of these methods are compared in Figure 4-6.

We also measured the effect of the collision horizon on these performance metrics, depicted in Figure 4-7. We find that M_0 and running MCR on sampled obstacles produce plans of similar quality, although M_0 is significantly faster. As before, setting the shadows to be equal is suboptimal in nearly all of the problems in this domain because there are too many obstacles, so it must choose an overly conservative shadow for each one. Overall, M_1 appears to be the most generally attractive option, as it is optimal in every instance, although in this domain increasing the collision horizon does not appear to increase runtime

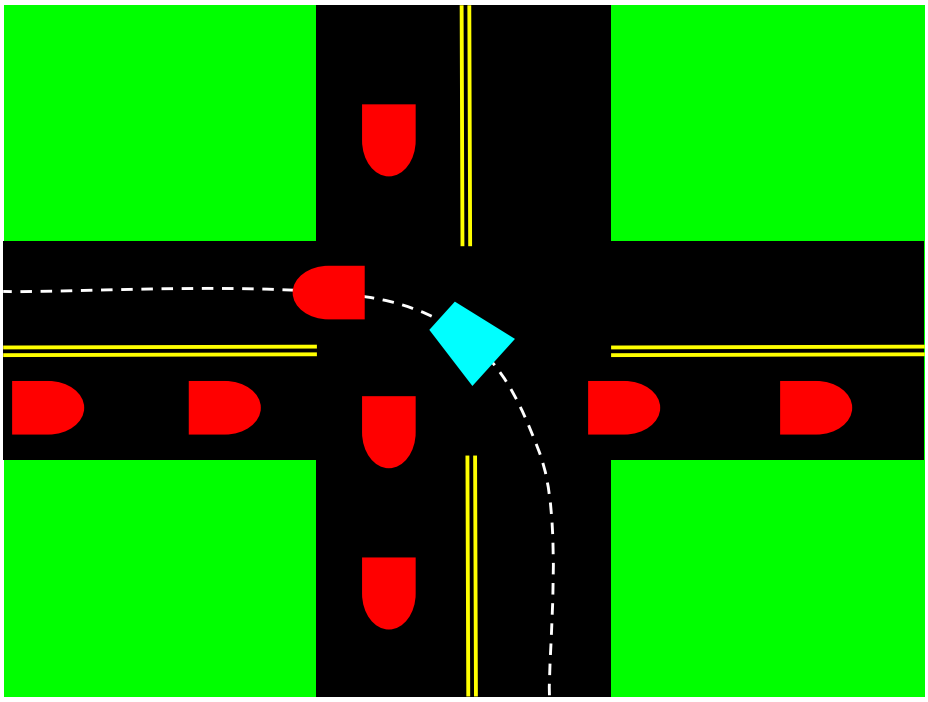

Figure 4-5: The robot (blue trapezoidal vehicle) making an unprotected left turn along the dotted white curve. Each obstacle (red rounded vehicles) exists in space-time and has uncertain speed. There is cross traffic going to the right blocking the robot's path before entering the intersection, oncoming traffic going downwards blocking the robot's path before exiting the intersection, and an obstacle vehicle in front. The robot must choose when it is safest to cut between vehicles, keeping in mind that going too fast risks collision with the front vehicle. There are a total of 12 obstacle vehicles.

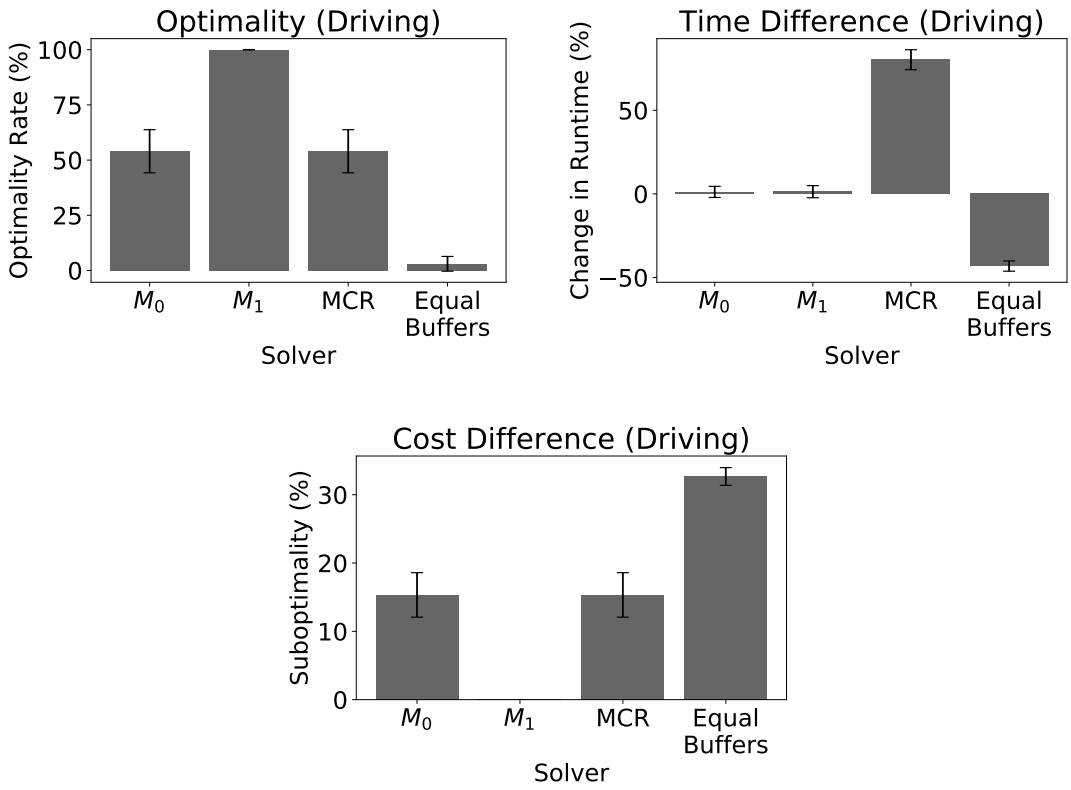

Figure 4-6: The optimality rate (percent of problem instances where the algorithm returns an optimal solution), runtime, and planning risk of each method. Runtime and cost are depicted as the difference compared to the optimal planner M_{12} to control for the variance between problem instances.

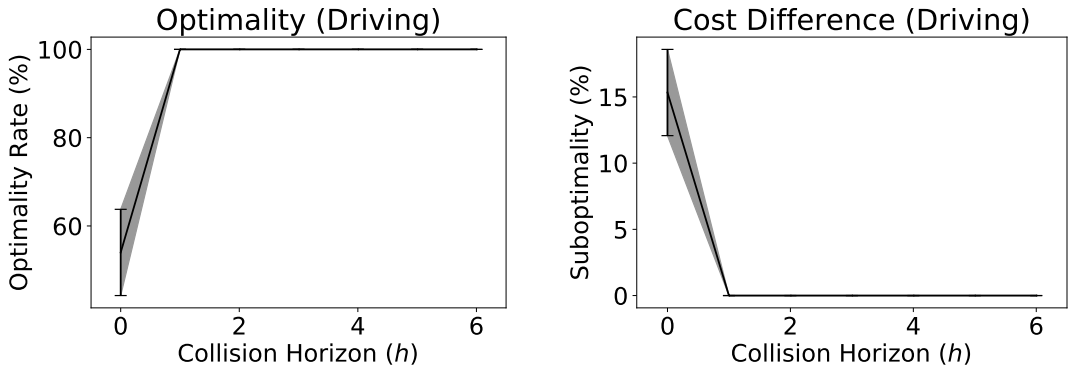

Figure 4-7: The optimality and planning risk of M_h for each collision horizon. Cost is depicted as the percent difference compared to the optimal planner M_{12} to control for the variance in difficulty between problem instances. There was no significant difference in runtime across different values of h, so the runtime graph is omitted. Increasing the collision horizon past 6 shows no noticeable change in behavior, so we have cropped the graphs for clarity.

significantly.

4.3.3 Minimum Constraint Removal for Manipulation Planning

We also evaluate our algorithm as a minimum constraint removal planner. Our experimental domain is identical to the one in Section 4.3.1, but the obstacles are deterministic and the task is instead to find the path with the fewest collisions. This task is very practically relevant in manipulation planning domains to determine which obstacles must be moved out of the way in order to perform a given operation.

As described before, Hauser [18] presented two algorithms, a greedy planner and an exact planner, which are equivalent to M_0 and M_{24}, respectively (note that M_{24} is equivalent to M_∞ when there are at most 24 obstacles). Our algorithm is compared for different settings of the collision horizon in Figure 4-8.

Similar to minimum-risk planning, we find that M_0 is already near optimal, and that M_1 closes the gap. As a result, it is unclear whether the collision horizon is bounded for this domain, or if it is just highly likely to be small. As before, M_1 strikes a good balance of optimal performance and quick runtime.

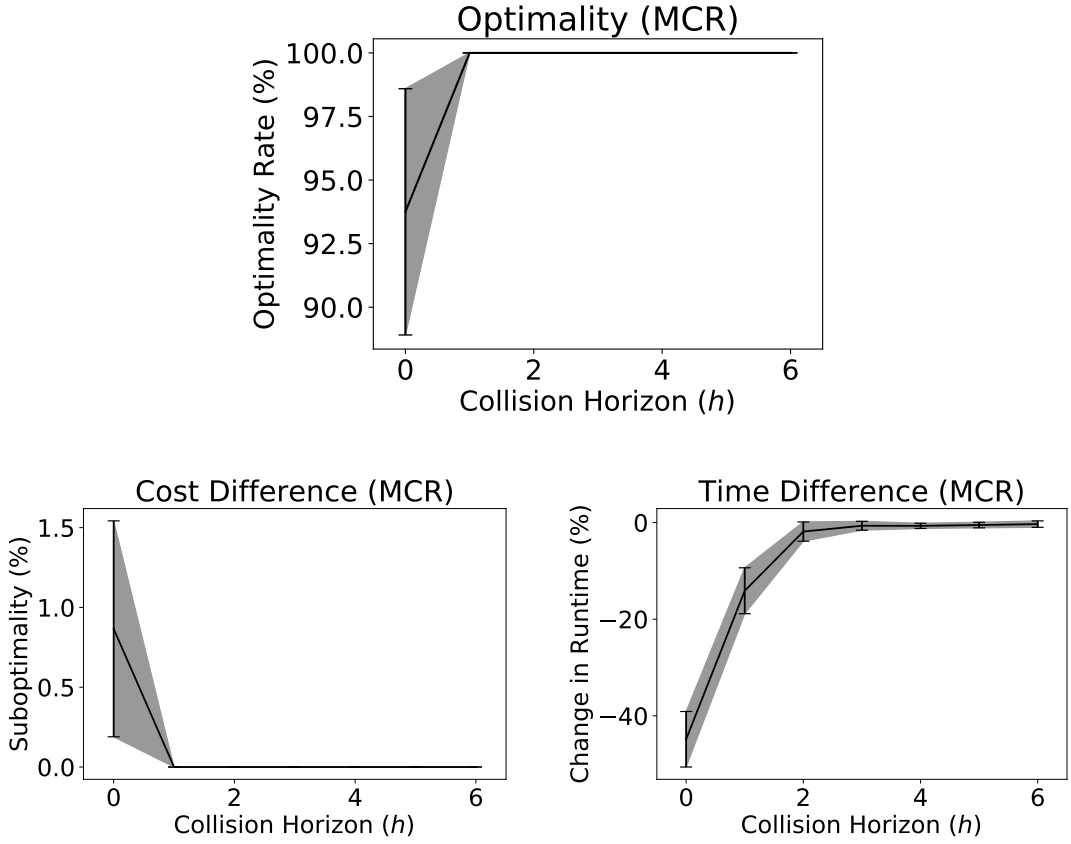

Figure 4-8: The optimality, runtime, and planning risk of M_h for each collision horizon. Runtime and cost are depicted as the difference compared to the optimal planner M_{24} to control for the variance in difficulty of different problem instances. Increasing the collision horizon past 6 up to 24 shows no noticeable change in behavior, so we have cropped the graphs for clarity.

Chapter 5

Conclusion

We have shown that the minimum-risk planning problem on graphs is NP-hard, even in dimension 2. However, it becomes tractable when the collision horizon is bounded. Furthermore, we present a practical algorithm that efficiently finds optimal plans for fixed collision horizon and finds approximately optimal plans with a natural suboptimality bound when the collision horizon is higher.

This demonstrates that approximate planning under obstacle uncertainty is tractable in practical domains, which can lead to improved robustness in many robotic planning domains. Furthermore, it shows that the collision horizon is the source of the hardness of the problem, suggesting that the field should look towards this parameter to find further improvements and applications.

One caveat is that the runtime of the algorithm scales poorly with the collision horizon. A potential direction for future work would be to further explore the relationship between the collision horizon and the hardness of the problem. Either finding more efficient algorithms that reduces the exponential runtime to one with a fixed base, ideally something like 2^h rather than the n^h our algorithm has, or showing that such an algorithm does not exist would be a significant step in understanding the nature of these problems.

Another limitation is that our algorithm operates on problems with discrete risk levels. In practice, this is reasonable because the obstacle shadows can be discretized based on the precision that is required. However, the number of risk levels has a substantial effect on runtime especially with higher collision horizons. A line of future work would be to

generalize this approach to a continuous notion of risk levels.

There is also the related direction of investigating models of uncertainty over obstacles. We focus on the PGDF model in our hardness proofs because it captures certain desirable characteristics and has been used in prior work. However, the PGDF model has certain surprising characteristics, particularly near the tails of the distribution [3]. Perhaps there is a model that is a better fit for obstacle estimates in practice. In exploring this direction, it is important to note that we do not strongly invoke the structure of PGDF obstacles. Interesting directions for future work also include finding a good minimal condition on the obstacle distribution to make the problem NP-hard.

Finally, we have shown that some realistic domains tend to have small collision horizons. An interesting open question is what properties of a problem domain determine this, and whether we can define special classes of problems that have provably fixed collision horizon.

Bibliography

[1] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic A*. *The International Journal of Robotics Research*, 35(1-3):224–243, 2016.

[2] Brian Axelrod. Algorithms for Safe Robot Navigation. Master's thesis, Massachusetts Institute of Technology, 2017.

[3] Brian Axelrod, Leslie Kaelbling, and Tomás Lozano-Pérez. Provably safe robot navigation with obstacle uncertainty. *Robotics Science and Systems*, 13, 2017. URL `http://lis.csail.mit.edu/pubs/axelrod-rss-17.pdf`.

[4] Brian Axelrod, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Provably safe robot navigation with obstacle uncertainty. *The International Journal of Robotics Research*, 2018.

[5] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 723–730. IEEE, 2011. URL `http://ieeexplore.ieee.org/abstract/document/5980508/`.

[6] J. F. Canny. Computing roadmaps of general semi-algebraic sets. In Harold F. Mattson, Teo Mora, and T. R. N. Rao, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 94–107, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg. ISBN 978-3-540-38436-6.

[7] John Canny. Some algebraic and geometric computations in PSPACE. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 460–467, 01 1988.

[8] John Canny and John Reif. New lower bound techniques for robot motion planning problems. In *Foundations of Computer Science*, pages 49 – 60, 11 1987. ISBN 0-8186-0807-2.

[9] A.R. Cassandra, L.P. Kaelbling, and James Kurien. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *International Conference on Intelligent Robots and Systems*, volume 12, pages 963 – 972 vol.2, 12 1996. ISBN 0-7803-3213-X.

[10] Xu Chu Ding, Alessandro Pinto, and Amit Surana. Strategic planning under uncertainties via constrained Markov decision processes. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4568–4575. IEEE, 2013. URL http://ieeexplore.ieee.org/document/6631226/.

[11] John E. Hopcroft, Deborah Joseph, and Sue Whitesides. Movement problems for 2-dimensional linkages. In *SIAM Journal on Computing*, volume 13, pages 610–629, 08 1984.

[12] Eduard Eiben, Jonathan Gemmell, Iyad Kanj, and Andrew Youngdahl. Improved results for minimum constraint removal. In *AAAI Conference on Artificial Intelligence*, 2018. URL https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16615.

[13] L. Erickson and S. LaValle. A simple, but NP-hard, motion planning problem. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[14] William F. Dowling and Jean Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1:267–284, 10 1984. doi: 10.1016/0743-1066(84)90014-1.

[15] Seyedshams Feyzabadi and Stefano Carpin. Multi-objective planning with multiple high level task specifications. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 5483–5490. IEEE, 2016. URL http://ieeexplore.ieee.org/document/7487762/.

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

[17] Kris Hauser. The minimum constraint removal problem with three robotics applications. In *Workshop on the Algorithmic Foundations of Robotics*, 2012.

[18] Kris Hauser. The minimum constraint removal problem with three robotics applications. In *The International Journal of Robotics Research*, volume 33, 2014.

[19] John Hopcroft, Deborah Joseph, and Sue Whitesides. On the movement of robot arms in 2-dimensional bounded regions. In *SIAM Journal on Computing*, volume 14, pages 280 – 289, 12 1982.

[20] Lucas Janson, Edward Schmerling, and Marco Pavone. Monte carlo motion planning for robot trajectory optimization under uncertainty. In *International Symposium on Robotics Research*, Sestri Levante, Italy, September 2015. URL https://web.stanford.edu/~pavone/papers/JSP.ISRR15.pdf.

[21] Brigitte Jaumard and Bruno Simeone. On the complexity of the maximum satisfiability problem for Horn formulas. *Information Processing Letters*, 26:1–4, 9 1987. doi: 10.1016/0020-0190(87)90028-7.

[22] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 2013. URL http://journals.sagepub.com/doi/abs/10.1177/0278364913484072.

[23] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *IEEE Conference on Decision and Control*, 2009.

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

[25] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[26] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, 2008.

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

[28] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. In *ICRA*, 1999.

[29] Alex Lee, Yan Duan, Sachin Patil, John Schulman, Zoe McCarthy, Jur van den Berg, Ken Goldberg, and Pieter Abbeel. Sigma hulls for Gaussian belief space planning for imprecise articulated robots amid obstacles. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5660–5667. IEEE, 2013. URL http://ieeexplore.ieee.org/document/6697176/.

[30] Christos H Papadimitriou and John N Tsitsiklis. The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[31] Chonhyon Park, Jae Sung Park, and Dinesh Manocha. Fast and bounded probabilistic collision detection in dynamic environments for high-dof trajectory planning. *CoRR*, abs/1607.04788, 2016. URL https://arxiv.org/abs/1607.04788.

[32] Jae Sung Park, Chonhyon Park, and Dinesh Manocha. Efficient probabilistic collision detection for non-convex shapes. *CoRR*, abs/1610.03651, 2016. URL https://arxiv.org/abs/1610.03651.

[33] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010. doi: 10.15607/RSS.2010.VI.037. URL http://www.roboticsproceedings.org/rss06/p37.html.

[34] Sam Prentice and Nicholas Roy. The belief roadmap: Efficient planning in linear pomdps by factoring the covariance. In *Proceedings of the 13th International Symposium of Robotics Research (ISRR)*, Hiroshima, Japan, November 2007.

[35] John Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 421–427, 11 1979.

[36] Dorsa Sadigh and Ashish Kapoor. Safe control under uncertainty with probabilistic signal temporal logic. In *Proceedings of Robotics: Science and Systems*, AnnArbor, Michigan, June 2016. doi: 10.15607/RSS.2016.XII.017. URL `http://www.roboticsproceedings.org/rss12/p17.html`.

[37] Oren Salzman and Siddhartha Srinivasa. Open problem on risk-aware planning in the plane. *arXiv preprint arXiv:1612.05101*, 2016.

[38] Oren Salzman, Brian Hou, and Siddhartha Srinivasa. Efficient motion planning for problems lacking optimal substructure. *arXiv preprint arXiv:1703.02582*, 2017.

[39] Luke Shimanuki and Brian Axelrod. Hardness of 3d motion planning under obstacle uncertainty. In *Workshop on the Algorithmic Foundations of Robotics*, 2018.

[40] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996. ISBN 053494728X.

[41] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online POMDP planning with regularization. In *Advances in Neural Information Processing Systems*, volume 58, 01 2013.

[42] Wen Sun, Luis G Torres, Jur Van Den Berg, and Ron Alterovitz. Safe motion planning for imprecise robotic manipulators by minimizing probability of collision. In *Robotics Research*, pages 685–701. Springer, 2016. URL `http://link.springer.com/chapter/10.1007/978-3-319-28872-7_39`.