Efficient Motion Planning under Obstacle Uncertainty with Local Dependencies

Brian Axelrod[1*] and Luke Shimanuki[2*]

[1] Stanford University, Stanford CA 94305, USA
baxelrod@cs.stanford.edu
[2] Magna Electronics [†], Boston MA 02210, USA
luke@shimanuki.cc

Abstract. Minimum Risk Motion Planning (MRMP) has been shown to remain NP-Hard even under the conditions that lead to the practical efficiency of sampling-based and grid-based motion planning algorithms [25]. However, the hardness proof does not eliminate the importance of finding a practical solution to this problem. In this paper we identify a parameter which directly controls the hardness of MRMP. We present experiments that suggest this parameter is small for many practical MRMPs and present an algorithm guaranteed to efficiently yield high-quality solutions whenever this parameter is small. When the parameter is large, the algorithm fails gracefully—it returns a solution with bounded suboptimality. We also explore a connection between our work and previous work on the minimum constraint removal problem (MCR).

Keywords: motion and path planning, probabilistic reasoning, completeness and complexity, collision avoidance

1 Introduction

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. Even after they are able to observe the environment, observations are often noisy and incomplete. Unfortunately, motion planning in this context is especially difficult when one wants to guarantee a low collision probability. One must reconstruct a model of the environment from sensor observations, quantify uncertainty in this estimation process, and then compute a path that ensures a low collision probability. For the rest of the paper, the term *safe* is used to refer to a trajectory with low collision probability over the randomness of the uncertain environment. Then the goal of Minimum Risk Motion Planning (MRMP) is to find such a safe trajectory. Note that this differs from many other works in planning under uncertainty which focus on uncertainty in estimation of the robot position, orientation, or ability to execute a trajectory precisely.

In this paper we build on previous work characterizing collision probabilities and focus on developing methods for computing safe trajectories. Unfortunately, since many

[†] Work performed while at MIT. This work represents the opinions of the authors only.
[*] Authors listed in alphabetical order.

instances of this problem are NP hard, we cannot hope to find a provably efficient algorithm that works in general. Instead we identify a parameter that 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 many practical instances*. When the parameter cannot be controlled, we demonstrate a tradeoff between computation time and the suboptimality of the result. The key contributions of this paper are twofold:

1. Defining a parameter that controls the hardness of MRMP and allows us to identify solvable MRMP problems. We demonstrate experimentally that this parameter is small for certain problems of interest.
2. Describing an algorithm that efficiently solves MRMP when the parameter is small. When the parameter is large, the algorithm allows trading off runtime and optimality.

In particular, this represents a different paradigm of working with computational complexity in robotics. Many works in robotics can be divided into two categories based on the hardness of the problem they are trying to solve: (a) the problem is efficiently (polynomial-time) solvable and the work presents an algorithm that guarantees an efficient runtime and optimal solution (b) the problem is provably computationally difficult so the work presents an algorithm that depends on heuristics and may only sometimes be efficient and/or produce an optimal solution. This paper takes a more fine grained approach. By identifying a parameter that quantifies how hard a problem instance is, we can understand the exact tradeoff between runtime and solution quality. As a result, we are able to present an algorithm with strong theoretical guarantees that are, as demonstrated in the experimental section, also practical and efficient at solving instances which could not be solved by prior approaches.

1.1 Approximations in Planning with Environment Uncertainty

The hardness of planning with environmental uncertainty has led to various approximations to make the problem more computationally feasible. One line of work models this uncertainty as a partially observable Markov decision problem (POMDP) [5]. Unfortunately, solving POMDPs is PSPACE-hard, implying that an efficient, general algorithm does not exist [18]. Furthermore, even solving POMDPs in practice has proven difficult for complicated problems, despite advances in approximate POMDP solvers [15, 26].

However, there are many works that suggest that navigation among uncertain obstacles is an easier problem than solving general POMDPs. There is a long line of work that approaches this problem, trading off performance, safety guarantees, completeness, and limitations of the model [4, 11, 17, 10, 27, 19, 20, 6, 8]. One way to avoid making these tradeoffs is to find a restricted setting where the problem is easier to solve. Unfortunately, positive results in this area are sparse. Consider the following two negative results for context. The first is that planning on a graph with uncertain obstacles remains hard, even in two dimensions and even if obstacles are guaranteed to not overlap with more than a fixed number of other obstacles [25]. The second concerns the related minimum constraint removal problem. Finding the minimal set of obstacles to remove to make travel between two points feasible remains NP-hard even if (a) the obstacles are constrained to

be axis-aligned rectangles or (b) obstacles are line segments such that no three intersect at one point [7]. In contrast to this prior work, this paper will present conditions under which the problem *is* efficiently solvable.

In this paper, 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 defined as the probability that the trajectory collides with an obstacle. Ideally, an algorithm would always efficiently find the least-risk solution and never return a solution that is less safe than advertised.

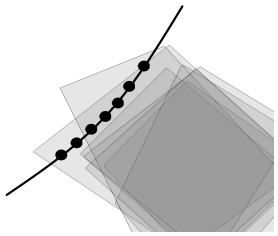

Fig. 1. Estimating the risk by summing the risk of individual waypoints can overestimate the risk since the collision probability of adjacent waypoints is often correlated in practice. In the above cartoon, we can see that if a draw contains one waypoint, it is likely to contain many. Even though the collision risk of the trajectory is 20%, the sum of the risks of the waypoints is greater than 100%.

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 along a discretized path 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 waypoint 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 does not change position is certain to collide! In reality, one expects collision probabilities at different points of a trajectory to potentially have significant correlations, making a union bound a poor strategy by which to bound the collision probability. This issue is illustrated in Figure 1.

This dependence on the discretization can be remedied by using the concept of a shadow. For every obstacle, a "shadow" (depicted in Figure 2) is a volume that contains the obstacle with a fixed probability. As long as 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 its obstacle. If one can decide the probabilities corresponding to each shadow ahead of time, MRMP 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, choosing these probabilities a priori is crucial for the reduction to the standard motion planning problem. 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 can have 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. The incompleteness resulting from allowing equal risk for every obstacle is illustrated in Figure 3. Unfortunately, planning without fixed shadows is harder than deterministic motion planning, even

when allowing for certain approximations. Even when the obstacles are polytopes composed of Gaussian-distributed faces (defined in [3]), with paths restricted to a small graph and with a point robot in a low dimensional space, the problem is NP-hard [25].

In this work we identify a parameter that determines the hardness of such problems. When this parameter is small, we compute an optimal solution efficiently. We believe this parameter is small for most practical instances of planning among uncertain obstacles.

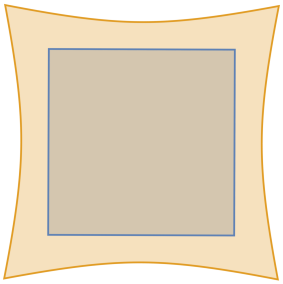

Fig. 2. When the true position of the blue square is unknown, we can identify the orange "shadow" as a region that contains the blue square with some probability [3].

1.2 Related Work

Since planning under uncertainty is both ubiquitous and provably hard, many works rely on heuristics, rarely providing formal guarantees for both solution quality and runtime.

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 mirror image. They often inflate the occupied volume of the obstacle with a "shadow" and ensure that any planned trajectory avoids the shadow [1, 11, 17]. 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 reason about which objects force the robot to incur more risk. The planner does not know beforehand how much it can expand each shadow while still allowing a feasible trajectory.

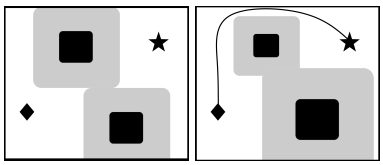

Fig. 3. Planning a safe trajectory requires deciding how to balance the risk among possible obstacles. In the first image, the robot is allowed to incur equal risk of colliding with each obstacle and no path between the diamond and the star exists. In the second, the robot is allowed more risk of colliding with the upper obstacle and less for the lower one, creating a passage for the robot.

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 [22, 4] or control systems [21]. However, the dimensionality of the belief space can become quite large in domains with many uncertain variables such as obstacles.

Another line of work uses synthesis techniques to construct trajectories that are 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 [6, 8]. Other authors have used techniques from Signal Temporal Logic combined with an explicitly modeled uncertainty to generate plans that are safe [24], though the ϵ in said paper does not correspond to the notion of safety used in this paper.

Recent work by Hauser [9] 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 suboptimality 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] 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.

Shimanuki and Axelrod [25] 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 MAXQHORNSAT 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, this suggests that the hardness of the problem stems from long-distance dependencies between potential collisions.

1.3 Summary of Formal and Experimental Results

As mentioned in section 1.2, 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. In this paper we identify a parameter that controls this, the *collision horizon*. 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, the collision horizon 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 1. *On problems with collision horizon at most h, M_h returns the minimal collision risk plan.*

Theorem 2. *On problems with collision horizon h^0, where $h^0 > h$, 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 fewest collisions, is also solved by our algorithm with the same guarantees.

Theorem 3. *The greedy and exact algorithms presented by Hauser [9] are equivalent to M_0 and M_1, 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 2.2 and 3.

We demonstrate M_h on a manipulation domain and an autonomous driving domain. We find that M_0 is near-optimal and that M_1 is optimal on all problem instances, indicating that the collision horizon tends to be small in these domains. We find similarly on an MCR manipulation domain.

2 Definitions

2.1 Formal Problem Definition

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 be restricting our discussion to methods that first construct a graph embedded in configuration space, such as a PRM [14] or RRG [12], then search the graph for safe plans.

A useful reference here are the shadows defined by Axelrod et al. [3]. 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 this paper works with the *monotone risk model*, which includes the shadows used in [3]. 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 any node colliding with the obstacle is bounded by a weakly monotone submodular function over the nodes in the path. When using shadows, this function becomes simply the maximum over the risks for each individual node being in collision with that obstacle. Note that it does not matter if risks are associated with nodes or edges since a graph with risks at edges may be transformed to one with risks at nodes by constructing a node for every edge.

Definition 1 (Monotone Risk Model). *Given a graph $G = (V, E)$ (with edges E and vertices V), a function $f : \mathsf{P}(E) \to L$ (with P denoting the power set, and $L \subseteq \mathbb{R}$ is a finite set over which the risk is discretized) is a* Monotone Risk Model *if f is a weakly monotone submodular set function* or *f is a weakly monotone accumulation of Monotone Risk Models, i.e. $f(E') = g(f_1(E'), f_2(E'))$, where f_1, f_2 are Monotone Risk Models and g is weakly monotone, that is, $g(w, x) \leq g(y, z)$ if $w \leq y$ and $x \leq z$. Typically, each $f_i(E')$ would represent the risk that a given path, represented as a set of edges $E' \subseteq E$, is in collision with obstacle i. Then the overall risk of the path can be obtained by combining the risk for each obstacle in some monotonic manner.*

A natural monotone accumulation would be the OR operation, treating the inputs as collision probabilities for independent events (i.e. $g(x, y) = x+y \quad xy$). In practice —and in our experiments —a union-bound accumulation, that is, summation over probabilities ($g(x, y) = x + y$), 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 let $R[x \ [\ y]$ denote this accumulation over the risks of events x and y.

Furthermore, the discrete minimum constraint removal problem (MCR), can also be expressed as a minimization of a monotone risk model. Each obstacle i corresponds to a monotone risk model f_i such that $f_i(E^0) = 1$ if any edge $e \ 2 \ E^0$ is in collision with obstacle i, and $f_i(E^0) = 0$ otherwise. Then the sum of the risk models for all obstacles, which corresponds exactly to the cost function defined in the MCR problem, is also 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. A useful abstraction is to define for each obstacle 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 to the uncertain case, defining a random obstacle and then constructing a mapping instead from robot trajectories to collision *probabilities*.

Definition 2 (Obstacle). *An obstacle is defined as a random volume in task space (usually* R^3*) drawn from a known distribution. We note that an obstacle also corresponds to a random volume in configuration space (all configurations that result in the robot colliding with the obstacle). Each obstacle o is associated with a mapping* $f_o : P(E) \ ! \ [0, \ 1 \ )$*, where* f_o *is a Monotone Risk Model denoting the risk of a path colliding with obstacle o.*

There are many ways to describe the risk of collision with a given obstacle as a weakly monotone submodular set function over edges, but just as an example, under the model presented in [3], the risk for a given edge corresponds to the risk of the minimal shadow that intersects with the edge. Then the risk of an entire path, or the risk of the minimal shadow that intersects with the path, is equal to the maximum over the risks for the individual edges. In this paper, we will be working with a discrete set of shadows for each obstacle. Then we define a risk level as the risk associated with a particular shadow.

Definition 3 (Risk Level). *A risk level* l_o^i *is the probability that obstacle o is not fully contained within its i'th shadow.*

Informally, the shadows for a set of risk levels can be thought of as regions enclosed by contour lines encircling an obstacle, with higher risk levels associated with smaller shadows of higher risk. We note this model is not limited to the shadows constructed in [3]. For example, it applies to shadows computed for different distributions using a similar technique.

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 trajectory). For a given set of distributions over obstacles, the risk of a trajectory is the probability that the trajectory intersects *any* obstacle. We frequently abuse notation, representing trajectories either in configuration space, task space, or nodes and edges in a graph. We now define an ϵ safe trajectory.

Definition 4 (ϵ-safe trajectory). *Given a joint distribution over random obstacles O, a trajectory τ is ϵ-safe if, for any sample $s_1..s_n$ O, $R[\tau \setminus s_i \in ? \in i]$ ϵ. That is, a trajectory is ϵ-safe if the probability that it collides with* any *obstacle is less than ϵ.*

Note that for obstacles $o_1...o_n$, the probability of collision is bounded by the sum of the individual collision probabilities. Thus if $\sum f_{o_i}(\tau)$ ϵ, τ is ϵ safe. This allows us to quantify the safety of a trajectory by using a monotone risk model.

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.*

While the problem is known in general to be intractable [25], and risk-constrained extensions to practical sampling-based motion planning algorithms that build up a tree lack probabilistic completeness [3], Axelrod [2] proposed a risk-constrained extension to the RRG algorithm [12] which is probabilistically complete. Unfortunately, that algorithm requires executing a risk-constrained graph search, which is also known to be intractable if completeness is required [25]. (The conditions and guarantees for probabilistic completeness of the algorithm presented in this paper are discussed in detail in Appendix C). Nevertheless, we would still like to find a practical algorithm that can solve this problem.

Definition 5 (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 an ϵ -safe trajectory in G, where ϵ is the minimum ϵ for which an ϵ-safe δ-inflated trajectory exists.*

The δ-inflation condition informally means that there must be a nonzero probability of eventually sampling a satisfying trajectory, and is more formally discussed in Appendix C. We note that an algorithm optimizing risk can be directly used in place of an algorithm satisfying bounded risk. Simply run the risk-minimizing algorithm and do not return the solution if the risk is too high.

2.2 Formal Definition of Collision Horizon

Because the minimum-risk graph-search problem is intractable, we would now like to define a parameterized version of the problem that is tractable when the parameter is fixed. We identify a parameter that drives the hardness of the minimum-risk graph-search problem. More specifically, we define the collision horizon which, loosely speaking, captures the length of dependencies between obstacles.

We first define the collision coverage, illustrated in Figure 4, which describes the distance between correlated collisions for a single obstacle.

Definition 6 (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 shadow between the first time it enters a given shadow 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 shadow of obstacle o', i.e. $\zeta_{o'}^{(\tau)} = \{t \mid f_{o'}(\tau(t)) > f_{o'}(\tau(t-1))\}$. Then

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

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

Definition 7 (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 $\mathbf{t}$.*

Definition 8 (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 an ϵ-safe trajectory in G, where ϵ is the minimum ϵ for which an ϵ-safe δ-inflated trajectory exists.*

We will now present such an algorithm.

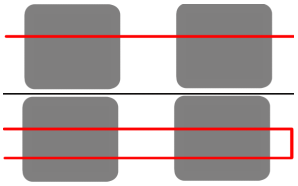

Fig. 4. In the upper half of the figure the path does not return to any obstacles and has a collision coverage of 0. The second path re-enters the first obstacle after going through the second, leading to a coverage of 2.

3 Algorithm

In this section, we present a polynomial-time h-horizon minimum-risk graph-search algorithm. The algorithm is based on Dijkstra's algorithm minimizing the collision risk, but instead of nodes corresponding to robot configurations, nodes correspond to a robot configuration and a risk-level memory limited to h obstacles. The other differentiator between our algorithm and a standard graph search is an update rule that takes advantage of the structure of shadows to expand multiple nodes simultaneously.

Recall that shadows are split up into risk levels. Once the robot enters a higher risk-level shadow for a particular obstacle, it does not incur any additional risk for time spent in lower risk-level shadows for that same obstacle. The risk memory keeps track of the maximum risk-level shadow entered for each of some number of obstacles. Then we define a state as a pair $(\mathbf{u}, C)$, where $\mathbf{u}$ is a robot configuration and C is a risk memory with $|C| \leq h$. Suppose edge $(\mathbf{u}, \mathbf{v})$ in the original graph enters some set C' of shadows. Then the augmented graph will have a corresponding set of edges $((\mathbf{u}, C), (\mathbf{v}, C''))$ for each C of size at most h, and for each C'' that can be obtained from C by adding (i.e. memorizing) the additional shadows in C' then removing (i.e. forgetting) some number of remembered shadows such that the resulting $|C''| \leq h$. Each of these edges would have a cost equal to the marginal risk of entering the shadows in C' conditioned on having already entered the shadows in C. This allows the cost function to only increase when the search enters a higher risk-level shadow than those recently visited.

We note that standard Dijkstra's algorithm with this graph augmentation on its own would be sufficient for our theoretical results below to hold, but the large branching factor stemming from the choice of which collision to forget causes a substantial, albeit still polynomial, increase in runtime. Instead, we introduce an optimization via an extension to the expansion rule that takes advantage of the structure imposed by shadows. In order to define the expansion rule we use the term "open" to refer to a node which is in the queue to be expanded, and "closed" to refer to nodes which have already been expanded. While the search must retain multiple paths to each robot configuration, potentially with different risks, not all paths are useful. Consider the situation where we already have a path to configuration $\mathbf{u}$ with computed risk 0.5 with a 0.2 risk shadow for obstacle one and 0.2 risk for obstacle two. The next node to be expanded reaches the same configuration with computed risk 0.5 with a 0.1 risk for obstacle one and a 0.1 risk for obstacle two. The computed risk for any path going through this new node is either worse or unchanged relative to if it had instead went through the first node, since any future marginal cost would be at least as high for the new node as for the first one. Then even though the new node technically has a different state, we don't have to expand it since it will not lead to any paths that are better than those that go through the first node.

We formally encode this idea by defining a partial ordering between risk memories such that $C \quad C'$ iff, for every obstacle o that is represented in C, the risk level for o in C is at most the risk level for o in C' (defaulting to false if o is not represented in C'). Then when deciding whether to expand a state $(\mathbf{u}, C)$, instead of just checking whether the exact state is closed, we check for every subset $M \quad C$ where $jMj \quad h$, whether any closed state at vertex $\mathbf{u}$ has a risk memory that M precedes. This works because $C \quad C'$ implies that any future marginal cost starting from $(\mathbf{u}, C)$ would be at least the future marginal cost starting from $(\mathbf{u}, C')$. This is effectively treating each state as a collection of all states for that vertex with a preceding risk memory. Pseudocode for this algorithm is provided in Algorithm 1. A step-by-step example of running a few steps of this algorithm is described in Appendix B.

Algorithm 1 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.

closed = {}
open = PriorityQueue({(0, (s, [], {}))})
while not open.empty() **do**
  // $\mathbf{u}$ is the configuration
  // τ is the trajectory that reaches $\mathbf{u}$
  // C is the risk memory
  // i.e. (obstacle, collision level) pairs
  $\mathbf{u}, \tau, C$ = open.pop()
  if $\mathbf{u} = \mathbf{t}$ **then**
    return τ // Reached goal
  // Check whether any closed state's
  // memory precedes C, or every
  // subset of C of size at most h
  // precedes the memory of some
  // closed state
  if $(@(\mathbf{w}, C'') \in \text{closed} s.t. \mathbf{w} = \mathbf{u}, C'' \leq C)$ and $(\exists M \leq C \ s.t. \ (|M| \leq h$ and $@ \ (\mathbf{w}, C'') \in \text{closed} \quad s.t. \quad \mathbf{w} = \mathbf{u}, M \leq C''))$ **then**
    closed.insert($\mathbf{u}$, C)
    // Add each outgoing edge to open
    for $\mathbf{v} \in E[\mathbf{u}]$ **do**
      $C' = \{(o, \max(C[o], f_o((\mathbf{u}, \mathbf{v})))$ for each $o \in O\}$
      open.insert($\sum_{o \in O} C'[o]$, $(\mathbf{v}, \tau + (\mathbf{u}, \mathbf{v}), C')$)

With k obstacles, L risk levels, and E edges, it can visit at most $(kL)^h$ unique states for each vertex (since there are $(kL)^h$ potential memories of size h, and each additional visited state corresponds to at least one new memory of size h that precedes the state's memory but none of the prior states' memories), so it can query at most $E(kL)^h$ edges. Since each edge can take $O\left(\binom{k}{h}(kL)^h h\right)$ to check whether it can be skipped (for each subset of size h, check against the memory for each visited state at that vertex, which requires comparing h risk levels for each check), the overall algorithm runs in time $O\left(\binom{k}{h}(kL)^{2h}h^2 E \log VkL\right)$, which is polynomial when h is fixed. We provide a more constrained runtime bound under certain conditions in Appendix E.

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 4. *Let ϵ be the associated cost of the trajectory generated by $M_h(G, O, \mathbf{s}, \mathbf{t})$ and let τ^* be any optimal trajectory, with associated cost ϵ^*. Then*

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

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

The proof for Theorem 4 is in Appendix D. And the following corollary holds when the collision horizon is fixed.

Theorem 5. *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 how much the optimal trajectory exceeds a collision horizon of h. This bound is especially helpful for domains where obstacle extents (i.e. the distribution over the obstacle boundaries) have infinite support, because while the collision horizon of these problems are large due to significant overlap of shadows, they impact the quality of the solution only by the amount of change in risk level over the areas that interact with more than h other obstacles, which is typically only the large, low-risk shadows. 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 5.

3.1 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 [9], 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.

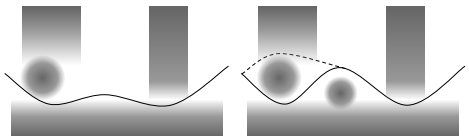

Fig. 5. An example where M_0 is optimal (left) and one where it is suboptimal (right). The shading indicates the obstacle shadows, so the probability a trajectory collides with a given obstacle is given by the darkness of the maximally shaded point it goes through. In both cases the optimal trajectory is the solid black line, which risks collision with the long obstacle at the bottom —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.

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 [9] and by Shimanuki and Axelrod [25], 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 shadow 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 [9], and M_1 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 Empirical Results

In this section we compare our algorithm to various baselines and illustrate how the collision horizon parameter infuences behavior. We consider two baselines: First, we implemented the naive and commonly used approach of setting all the shadows to be equal, often referred to as constructing buffers, and then we ran 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 [9] 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 avoiding 90% of samples does not guarantee a collision rate of less than 10%. In fact, the number of samples required for such an empirical collision rate estimate to be useful is quite large and depends on the dimension of the space. As such, we slightly modify it to construct each shadow as an individual obstacle rather than sampling actual obstacle instances. This ensures the soundness of the algorithm while also reducing the runtime due to needing fewer obstacles.

4.1 Moving Boxes Domain

Our first domain is a pick-and-place motion planning problem among uncertain obstacles. An example problem instance is depicted in Figure 6. 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 7. We also measured the effect of the collision horizon on these performance metrics, depicted in Figure 8. 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 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.

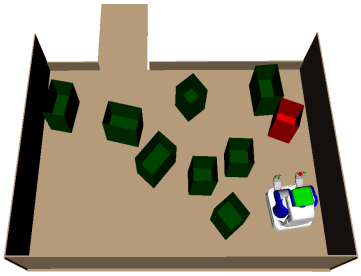

Fig. 6. The robot is tasked to pick up the red box and carry it out of the room through the hallway at the top. The green boxes (of which there are 8 to 24) have known position but Gaussian distributed extents. This is then discretized into shadows for 3 risk levels.

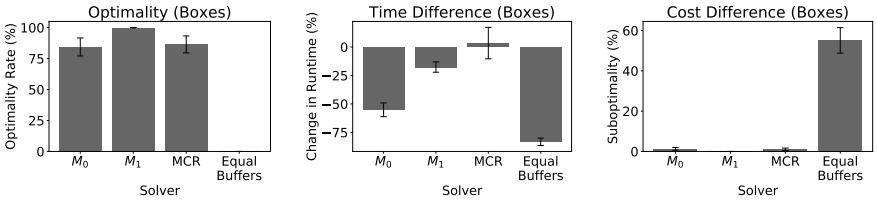

Fig. 7. The optimality rate (percent of problem instances where the algorithm returns an optimal solution), runtime, and planning risk of each method. Runtime and cost are depicted as the difference compared to the optimal planner M_{24} to control for the variance in difficulty of different problem instances. Note that the Equal Buffers algorithm, which assigns equal risk to every obstacle, was not able to find the optimal solution in any problem instance.

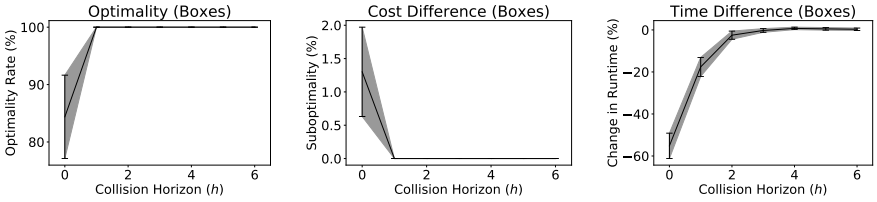

Fig. 8. The optimality, runtime, and planning risk of M_h for each collision horizon. Runtime and cost are depicted as the difference compared to the optimal planner M_{24} to control for the variance in difficulty of different problem instances. Increasing the collision horizon past 6 up to 24 shows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

The MCR algorithm runs slower, however, because it cannot take advantage of the fact that the obstacles correspond to quantile shadows. Setting the shadows to be equal is a fast solution, but is very suboptimal. Overall, M_1 appears to be the most generally attractive, as it is optimal in every instance and slightly faster than the exact search. We also notice that the runtime did not increase with h as much as we would expect. We speculate as to why this might be the case and analyze conditions under which we would expect similar behavior in Appendix E.

4.2 Additional Experiments

In Appendix A we present similar results in an autonomous driving domain and in a MCR domain similar to the above manipulation problem.

5 Conclusion and Future Work

We show that while searching a graph for minimal risk plans is NP-hard, it becomes tractable when the collision horizon is bounded. Furthermore, there is 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, modeling a pattern for improvements in other settings.

A potential direction for future work would be to further explore the relationship between the collision horizon and the hardness of the problem. In particular, our result provides a runtime with n in the base of the exponential dependency on the collision horizon, placing it in the complexity class XP instead of FPT (fixed-parameter tractability), which would require a runtime of the form $f(h)poly(n)$. Either finding more efficient algorithms that reduce 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 further our understanding of the nature of these problems.

Another limitation is that our algorithm operates on problems with discrete risk levels. While we can always discretize obstacles based on the precision required, 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 continuous notion of risk levels.

Finally, we have shown that some realistic domains tend to have small collision horizons. An interesting open question is what determines this, and whether we can define special classes of problems that have provably fixed collision horizon. Generally it will come down to either the distance between sections of task space a single obstacle might occupy or how much the solution requires the robot to backtrack in the same section of task space. For example, we expect that many movable-base navigation problems with small obstacles will also have small collision horizons since the robot would not need to revisit any previously visited areas. On the other hand, long-horizon manipulation problems may exhibit unbounded collision horizon since each motion might revisit the same area. Additionally, problems with moving obstacles may also

have high collision horizon since each obstacle can interact with the robot at multiple different points along a planned trajectory. Unfortunately, it is not clear how to determine the collision horizon of a given problem without knowing the optimal solution. Another potential direction for future work would be to show whether this is tractable or NP-hard.

Acknowledgments

We gratefully acknowledge support from NSF grants 1723381 and IIS 1908774; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; from MIT-IBM Watson Lab and from the MIT Quest for Intelligence. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

We would also like to thank Tomás Lozano-Pérez and Leslie Pack Kaelbling for their advice and guidance throughout this project.

References

[1] Brian Williams Ashkan Jasour. Risk contours map for risk bounded motion planning under perception uncertainties. In *Robotics: Science and Systems*, 2019.

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

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

[4] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 723–730. IEEE, 2011.

[5] 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.

[6] Xu Chu Ding, Alessandro Pinto, and Amit Surana. Strategic planning under uncertainties via constrained Markov decision processes. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4568–4575. IEEE, 2013.

[7] Eduard Eiben, Jonathan Gemmell, Iyad Kanj, and Andrew Youngdahl. Improved results for minimum constraint removal. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[8] Seyedshams Feyzabadi and Stefano Carpin. Multi-objective planning with multiple high level task specifications. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5483–5490. IEEE, 2016.

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

[10] 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.

[11] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 2013.

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

[13] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 2011.

[14] 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.

[15] 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.

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

[17] 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.

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

[19] 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.

[20] Jae Sung Park, Chonhyon Park, and Dinesh Manocha. Efficient Probabilistic Collision Detection for Non-Convex Shapes. *CoRR*, abs/1610.03651, 2016.

[21] 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.

[22] 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.

[23] John Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, 1979.

[24] 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.

[25] Luke Shimanuki and Brian Axelrod. Hardness of motion planning with obstacle uncertainty in two dimensions. *The International Journal of Robotics Research*, 40 (10-11):1151–1166, 2021. doi: 10.1177/0278364921992787.

[26] 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.

[27] 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.

A Additional Experiments

A.1 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 9.

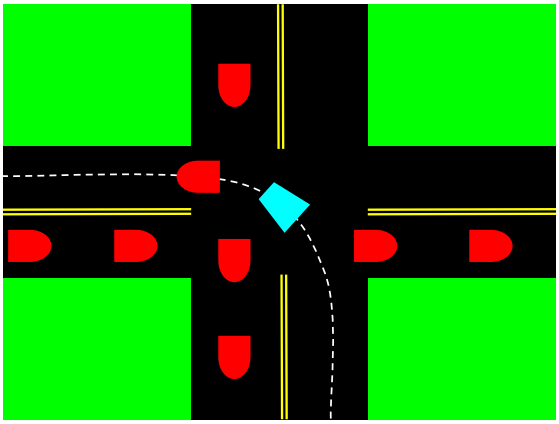

Fig. 9. The robot (blue trapezoidal vehicle) making an unprotected left turn along the dotted white curve. Each obstacle (red rounded vehicles) exists in space-time and has uncertain speed. There is cross traffic going to the right blocking the robot's path before entering the intersection, oncoming 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.

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 10. We also measured the effect of the collision horizon on these performance metrics, depicted in Figure 11. 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 significantly.

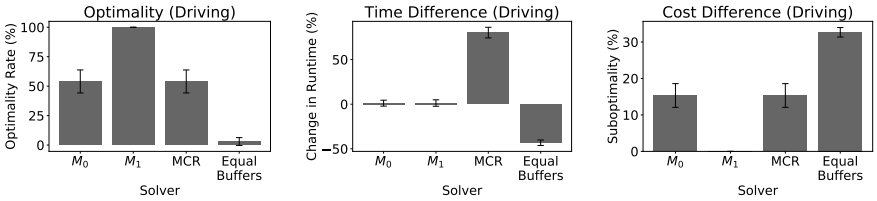

Fig. 10. The optimality rate (percent of problem instances where the algorithm returns an optimal solution), runtime, and planning risk of each method. Runtime and cost are depicted as the difference compared to the optimal planner M_{12} to control for the variance between problem instances.

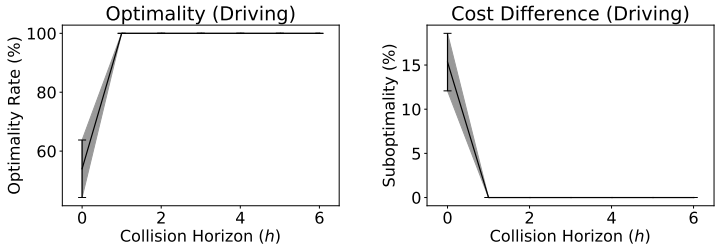

Fig. 11. The optimality and planning risk of M_h for each collision horizon. Cost is depicted as the percent difference compared to the optimal planner M_{12} to control for the variance in difficulty between problem instances. There was no significant difference in runtime across different values of 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.

A.2 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 4.1, 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 [9] 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_{1} when there are at most 24 obstacles). Our algorithm is compared for different settings of the collision horizon in Figure 12.

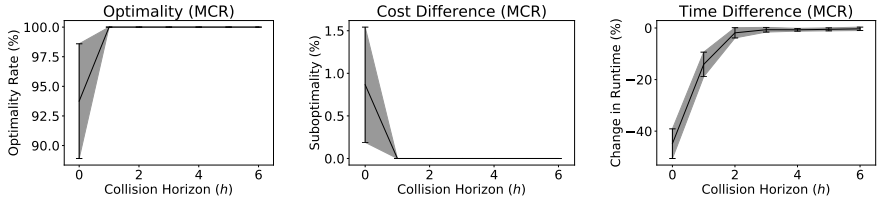

Fig. 12. The optimality, runtime, and planning risk of M_h for each collision horizon. Runtime and cost are depicted as the difference compared to the optimal planner M_{24} to control for the variance in difficulty of different problem instances. Increasing the collision horizon past 6 up to 24 shows no noticeable change in behavior, so we have cropped the graphs for clarity.

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.

B Step-by-step Example of Algorithm

B.1 Problem Setup

Here we walk through a step-by-step example of running Algorithm 1 on a small problem. Suppose we have a graph with vertices v_1, v_2, v_3, v_4 and edges

$$\begin{aligned} e_1 &= (\mathbf{v_1}, \mathbf{v_2}) \\ e_2 &= (\mathbf{v_1}, \mathbf{v_3}) \\ e_3 &= (\mathbf{v_2}, \mathbf{v_3}) \\ e_4 &= (\mathbf{v_3}, \mathbf{v_4}) \\ e_5 &= (\mathbf{v_3}, \mathbf{v_2}) \end{aligned} \tag{1}$$

Then let there be 2 obstacles o_1 and o_2, each with two risk levels of .01 and .05, where

$$\begin{aligned} &f_{o_1}(\{e_1\}) = f_{o_1}(\{e_3\}) = f_{o_1}(\{e_4\}) = f_{o_1}(\{e_5\}) = .05 \\ &f_{o_1}(\{e_2\}) = f_{o_1}(\{e_4\}) = 0 \\ &f_{o_2}(\{e_2\}) = .01 \\ &f_{o_2}(\{e_1\}) = f_{o_2}(\{e_3\}) = f_{o_2}(\{e_4\}) = 0 \end{aligned} \tag{2}$$

B.2 Algorithm Execution

We will now walk through M_1 starting at $\mathbf{s} = \mathbf{v_1}$ and ending at $\mathbf{t} = \mathbf{v_4}$. The referenced psudeocode may be found in Algorithm 1.

Iteration 1 We start with an empty closed set (line 1) and an open set containing the start vertex $\mathbf{v_1}$ (line 2). When we enter the main loop (line 3), we assign $\mathbf{u} = \mathbf{v_1}$ with empty τ and C (line 8). $\mathbf{u} \neq \mathbf{t}$, and so we do not end here (line 9).

Now we get to line 16. The closed set is empty, and so the first clause evaluates to true since there does not exist any $(\mathbf{w}, C'')$ in closed. For the second clause, we can set $M = \{\}$ C, which satisfies $|M| \leq 1$ and also there not existing any $(\mathbf{w}, C'')$ in closed. Hence, the overall predicate evaluates to true.

We then add $(\mathbf{v_1}, \{\})$ to the closed set (line 17). For outgoing edge e_1 (line 19), we set $C' = \{(o_1, .05)\}$ (line 20) and add $(\mathbf{v_2}, [e_1], C')$ to the open set with key .05 (line 21). For outgoing edge e_2 (line 19), we set $C' = \{(o_2, .01)\}$ (line 20) and add $(\mathbf{v_3}, [e_2], C')$ to the open set with key .01 (line 21).

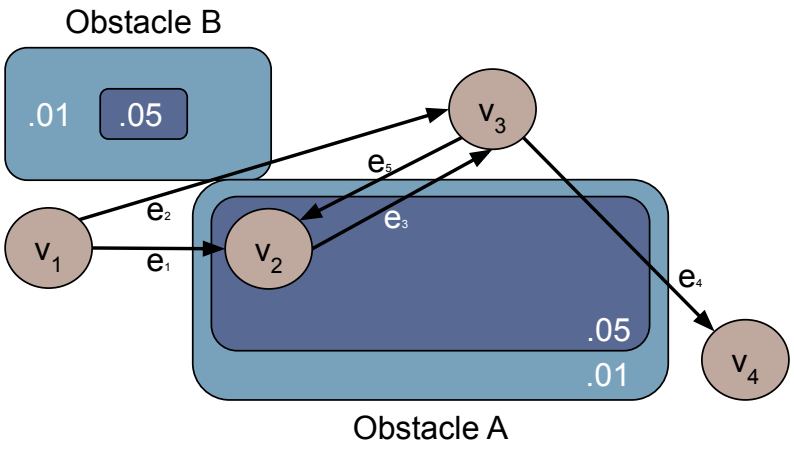

Fig. 13. End of iteration 1.

The set of closed states at the end of this iteration is depicted in Figure 13.

Iteration 2 Now we come back to the top of the main loop (line 3). We assign $\mathbf{u} = \mathbf{v_3}$ with $\tau = [e_2]$ and $C = \{(o_2, .01)\}$ (line 8). $\mathbf{u} \neq \mathbf{t}$, and so we do not end here (line 9).

There is no closed state at vertex $\mathbf{v_3}$, and so the predicate evaluates to true (line 16).

We then add $(\mathbf{v_3}, \{(o_2, .01)\})$ to the closed set (line 17). For outgoing edge e_4 (line 19), we set $C' = \{(o_1, .05), (o_2, .01)\}$ (line 20) and add $(\mathbf{v_4}, [e_2, e_4], C')$ to the open set with key .06 (line 21). For outgoing edge e_5 (line 19), we set $C' = \{(o_1, .05), (o_2, .01)\}$ (line 20) and add $(\mathbf{v_2}, [e_2, e_5], C')$ to the open set with key .06 (line 21).

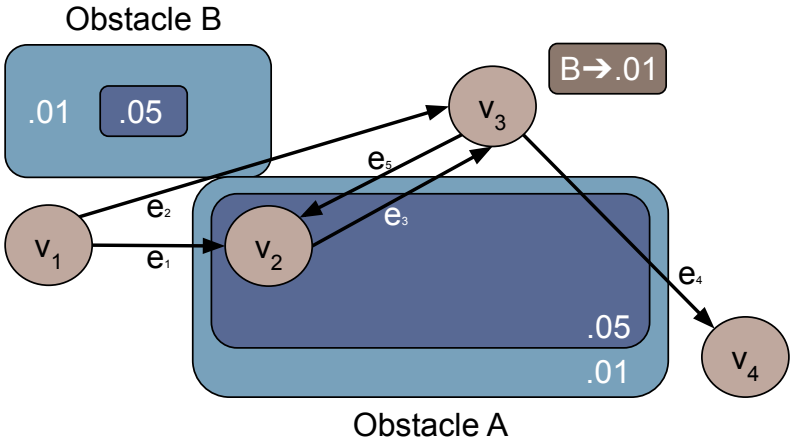

Fig. 14. End of iteration 2.

The set of closed states at the end of this iteration is depicted in Figure 14.

Iteration 3 We come back to the top of the main loop (line 3). We assign $\mathbf{u} = \mathbf{v_2}$ with $\tau = [e_1]$ and $C = f o_1, .05g$ (line 8). $\mathbf{u} \neq \mathbf{t}$, and so we do not end here (line 9).

There is no closed state at vertex $\mathbf{v_2}$, and so the predicate evaluates to true (line 16).

We then add $(\mathbf{v_2}, f(o_1, .05)g)$ to the closed set (line 17). For outgoing edge e_3 (line 19), we set $C' = f(o_1, .05)g$ (line 20) and add $(\mathbf{v_3}, [e_1, e_3], C')$ to the open set with key .05 (line 21).

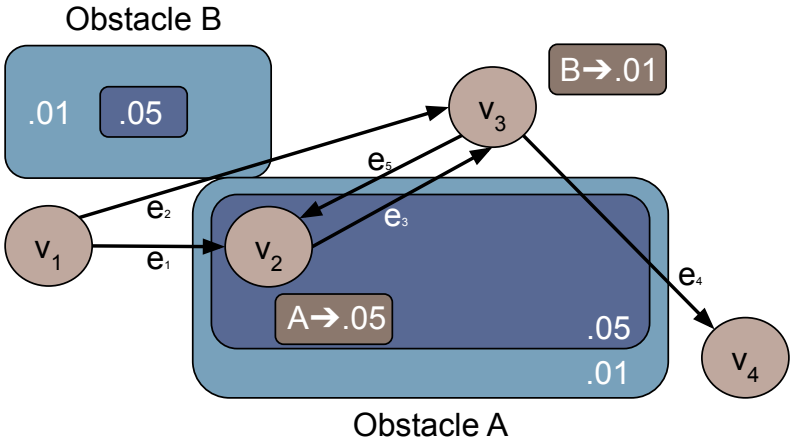

Fig. 15. End of iteration 3.

The set of closed states at the end of this iteration is depicted in Figure 15.

Iteration 4 Now we come back to the top of the main loop (line 3). We assign $\mathbf{u} = \mathbf{v_3}$ with $\tau = [e_1, e_3]$ and $C = \{o_1, .05\}$ (line 8). $\mathbf{u} \neq \mathbf{t}$, and so we do not end here (line 9).

There is only closed state $(\mathbf{v_3}, \{(o_2, .01)\})$ at vertex $\mathbf{v_3}$. $\{(o_2, .01)\} \neq C$ and we can set $M = C$ satisfying $|M| \geq 1$ and $M \neq \{(o_2, .01)\}$. Therefore, the predicate evaluates to true (line 16).

We then add $(\mathbf{v_3}, \{(o_1, .05)\})$ to the closed set (line 17). For outgoing edge e_4 (line 19), we set $C' = \{(o_1, .05)\}$ (line 20) and add $(\mathbf{v_4}, [e_1, e_3, e_4], C')$ to the open set with key .05 (line 21). For outgoing edge e_5 (line 19), we set $C' = \{(o_1, .05)\}$ (line 20) and add $(\mathbf{v_2}, [e_1, e_3, e_5], C')$ to the open set with key .05 (line 21).

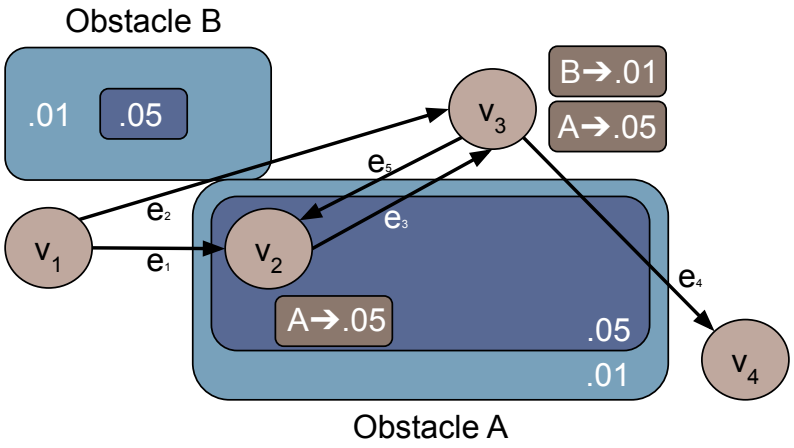

Fig. 16. End of iteration 4.

The set of closed states at the end of this iteration is depicted in Figure 16.

Iteration 5 We come back to the top of the main loop (line 3). We assign $\mathbf{u} = \mathbf{v_2}$ with $\tau = [e_1, e_3, e_5]$ and $C = \{o_1, .05\}$ (line 8). $\mathbf{u} \neq \mathbf{t}$, and so we do not end here (line 9).

There is only closed state $(\mathbf{v_2}, \{(o_1, .05)\})$ at vertex $\mathbf{v_3}$. $\{(o_1, .05)\} \in C$ and so the predicate evaluates to false (line 16).

Because we did not need to expand this state, the closed set remains unchanged, and so it is still as depicted in Figure 16.

Iteration 6 Finally, we reach the top of the main loop (line 3) with $\mathbf{u} = \mathbf{v_4}$, $\tau = [e_1, e_3, e_4]$ and $C = \{o_1, .05\}$ (line 8). $\mathbf{u} = \mathbf{t}$, and so we return the solution $[e_1, e_3, e_4]$ (line 9).

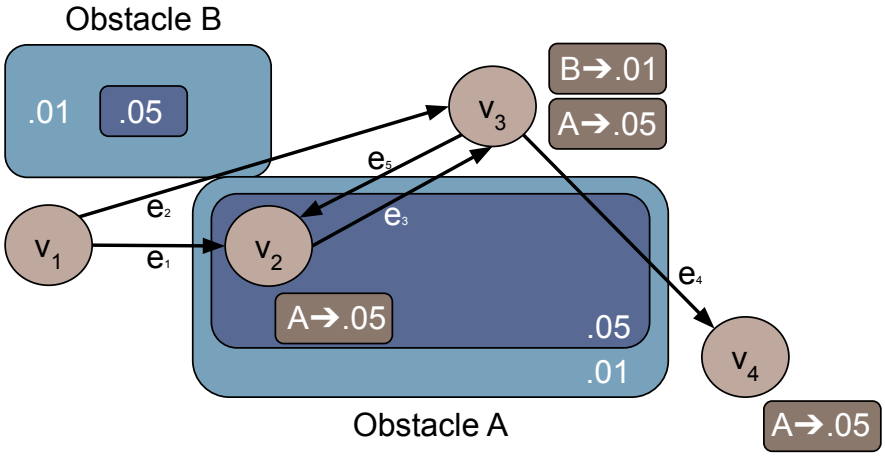

Fig. 17. End of iteration 6.

The set of closed states at the end of this iteration is depicted in Figure 17.

C Theoretical Guarantees for Motion Planning

While motion planning has been shown to be PSPACE-hard [23], the community has developed algorithms which are able to solve many practical motion planning problems. Even though we cannot guarantee that any algorithm is both efficient (polynomial time) and complete (guaranteed to find a solution), we can provide lighter guarantees. The goal of this section is to provide background on different theoretical guarantees relevant to motion planning with obstacle uncertainty.

C.1 Completeness

An algorithm is said to be *complete* if it is always able to find a solution if one exists. Unfortunately, many algorithms which depend on heuristics are not complete and can fail in certain scenarios. For example, optimization based motion planners can fail to find a solution if they are not initialized with a trajectory whose homotopy class contains a valid trajectory.

When working with sampling based motion planners, we work with a criteria known as *probabilistic completness*, a property introduced with the RRT algorithm [16]. While we cannot guarantee that any algorithm efficiently returns a valid solution if one exists, we can ensure the algorithm is probabilistically complete.

Definition 9 (Probabilistically Complete Motion Planning Algorithm). *A motion planning algorithm takes a set of obstacles, a start state* $\mathbf{s}$*, and a goal state* $\mathbf{t}$ *as input and generates a trajectory that does not intersect any obstacle as output. A planning algorithm is probabilistically complete if, with* n *samples, the probability that it finds a safe trajectory approaches* 1 *as* n *approaches* ∞.

Probabilistic completeness essentially means the algorithm will eventually find a solution if one exists.

Many sampling based algorithms, including RRTs, guarantee probabilistic completeness as long as there exists a solution that meets certain conditions. In particular, RRTs and many other sampling based motion planners are only probabilistically complete if there exists a solution trajectory in the topological interior of free configuration space. Since a similar condition is necessary for the algorithm presented in this paper, we develop this condition without explicitly relying on topology below.

This condition can be articulated as the existence of a path in the δ-interior of the free space X_{free} where X_{free} is a bounded subset of R^n.

Definition 10 (δ-interior [13]). *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}.*

A sampling based algorithm can only succeed if it always has a non-zero probability of sampling a waypoint leading to more progress. One way to ensure this is requiring the existence of a solution where every waypoint has a ball of nonzero diameter around it, guaranteeing that said ball has non-zero measure and the algorithm will eventually draw a sample in said ball.

When there exists a path in the δ-interior of free space for some $\delta > 0$, many sampling based motion planners are probabilistically complete. 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 11 (δ-inflation). *The δ-inflation of the set X is the set $Y = \bigcup_{x \in X} \{y \mid d(x, y) \leq \delta\}$, where $d(x, y)$ is any distance metric.*

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 12 (ϵ-safe δ-inflated trajectory). *A trajectory τ is an ϵ-safe δ-inflated trajectory if its δ-inflation intersects an obstacle with probability at most ϵ.*

While the standard RRT requires the existence of a trajectory in the interior of free space in order to be probabilistically complete, the algorithm in this paper requires the existence of an ϵ safe δ inflated trajectory in order to be probabilistically complete.

C.2 Graph Restriction Hardness

When solving motion planning without uncertainty, once the algorithm has identified a graph that contains a solution, the problem is essentially solved. Algorithms like Djikstra's algorithm and A* can be applied out of the box to find the minimum cost path within said graph.

We refer to the hardness of finding a solution restricted to a graph the *graph restriction complexity* of a problem. Since we can apply Djikstra's algorithm to solve motion

planning without uncertainty, the graph restriction complexity is P. This is crucial to enabling the fast solving of motion planning in practice. Sampling based planners tackle the problem in two [sometimes alternating] phases. The first phase involves sampling a graph. The second involves checking if the graph contains a solution, and finding the lowest cost one if it does. Motion planning problems that are "easy" for sampling based planners are ones where the first phase is easy. The hardness of the second phase is exactly the graph restriction complexity.

One could hope that with the right approximations, a similar pattern could work for motion planning with obstacle uncertainty. In [3], the authors develop a notion of confidence intervals around obstacles with the aim of using them as an approximation enabling efficient planning. Unfortunately, the graph restriction complexity of planning with obstacle uncertainty with shadows is still NP-hard, even in two dimensions [25]. In other words, even if you are able to efficiently identify a graph containing the solution, it is not easy to find the solution in this graph unless P=NP.

This paper presents an algorithm that shows that the graph-restriction complexity of MRMP is P when the collision horizon is constant. Under these conditions, the same paradigm as for standard motion planning applies. One can sample a graph in configuration space just like with a standard sampling based motion planner and then use the presented graph search algorithm in order to solve MRMP. This allows us to adapt the standard sampling based motion planners to be able to solve MRMP.

D Proofs

Proof of Theorem 4:

Proof. Let ϵ be the associated cost of the trajectory generated by $M_h(G, O, \mathbf{s}, \mathbf{t})$ and let τ be any optimal trajectory, 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 an obstacle level with edge (u, v), the planning tree generated by M_h must contain a state $\mathfrak{s}_u$ 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 closed 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 $(\mathfrak{s}_u, \mathfrak{s}_v)$, where $\mathfrak{s}_v$ 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

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

Because M_h greedily expands nodes in order of computed cost, it would only select a different trajectory if its computed cost $\hat{\epsilon}$ $\widetilde{\epsilon}$. 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:

$$\epsilon \quad \hat{\epsilon} \quad \widetilde{\epsilon} \quad \epsilon + \sum_{o \in O} (|S_o^{(\tau)}| - 1) f_o(\tau)$$

E Additional Runtime Analysis

Although the runtime of the exact algorithm (i.e. M_k) has poor asymptotic complexity according to our analysis, in our experiments we observe that in practice it seems to not be that much slower than the approximate versions. We would like to investigate why that would be the case, since it is a surprising result.

We define an extension of the irreducible constraint removal (ICR) defined by Hauser [9] for MCR domains.

Definition 13 (minimal reachable memory (MRM)). *A memory C is a minimal reachable memory for a configuration $\mathbf{u}$ if there exists a trajectory from the initial configuration $\mathbf{s}$ to $\mathbf{u}$ that does not collide with any shadow not in C, and there is no lower memory $C' < C$ such that there exists such a trajectory that does not collide with any shadow not in C'.*

Hauser [9] observe that the pruning of non-ICR states eliminates a large number of states, leading to practical efficiency for many MCR problem instances. In our algorithm, we prune non-MRM states (note that when using our algorithm as an MCR planner, this is equivalent to pruning of non-ICR states), so we would like to quantify how much of an effect the pruning has on the overall runtime.

Let U denote the set of risk memories associated with all states for configuration $\mathbf{u}$ that are expanded by M_k. Suppose $C_1, C_2 \in U$, and M_k expanded C_1 before C_2. If $C_1 \not\subset C_2$, then C_2 would not have been expanded since $(\mathbf{u}, C_2)$ is a strictly worse state than $(\mathbf{u}, C_1)$, hence a contradiction. But if $C_2 \not\subset C_1$, then C_2 would have been expanded before C_1, which is also a contradiction. Therefore, C_1 and C_2 are incomparable, and so U is an antichain.

For the purpose of simplifying the following math, we will continue only for the case where $L = 1$ (i.e. equivalent to MCR), but we believe the general orders of magnitude of the effect to be similar for larger L. In this case, the maximum antichain is the set of subsets of size $\frac{k}{2}$, which has size $\binom{k}{\frac{k}{2}}$. However, the actual antichain the algorithm produces is usually much smaller.

One reason is that it selects ICR sets rather than those in the maximum antichain. If most ICR sets have size approximately λ, then the size of U would be reduced to

approximately $\binom{k}{\lambda}$. This is the extent of the effect of pruning on the runtime, and it does not appear to sufficiently explain the low runtime we see in practice (in the boxes MCR domain, we see typical ICR sizes of on the order of around 8, which would still suggest a hundreds-of-thousands-fold runtime multiplier over the greedy algorithm).

We speculate that the topology of the obstacle placements induce additional constraints on which memories are reachable. For example, if the obstacles are arranged into a 2D grid, you would not be able to reach a shadow in the corner without also reaching some set of shadows in-between. Computational experiments suggest that this would reduce the size of U to a manageable number for the size of problems we have shown here, Exact counts for an MCR domain where $\mathbf{s}$ and $\mathbf{t}$ are δ cells apart and the algorithm is limited to reaching Δ obstacles (e.g. due to the algorithm terminating upon reaching the goal by passing through Δ obstacles) shown in the below table. Beyond the boundaries of the table, we expect that the size of U has asymptotic complexity at least $o\left(\delta^{\text{poly}(\Delta \quad \delta)}\right)$.

		Δ				
		8	9	10	11	12
δ	6	31	31	203	203	823
	7	1	43	43	375	375
	8	1	1	57	57	647
	9	0	1	1	73	73
	10	0	0	1	1	91

We expect the boxes domain to exhibit similar properties due to the dense placement of the obstacles (we estimate that in the boxes MCR domain the goal is reached with cost around 10 and the average obstacle is reached with cost around 8), although likely in an inexact manner. A potential line of future work could be to analytically quantify the impact of the obstacle topology on the runtime of the algorithm, and determine whether that sufficiently explains the empirically low runtime we have observed. Additionally, it would be helpful to evaluate the runtime of our algorithm on problems where δ and $\Delta \quad \delta$ are much larger to see if it continues to be efficient. This would also provide further intuition as to what is driving the hardness of the general problem, and which special cases can be solved efficiently when an exact solution is required and the collision horizon is unknown.