Faster Sample-based Motion Planning using Instance-based Learning

Jia Pan and Sachin Chitta and Dinesh Manocha

Abstract We present a novel approach to improve the performance of sample-based motion planners by learning from prior instances. Our formulation stores the results of prior collision and local planning queries. This information is used to accelerate the performance of planners based on probabilistic collision checking, select new local paths in free space, and compute an efficient order to perform queries along a search path in a graph. We present fast and novel algorithms to perform k-NN (k-nearest neighbor) queries in high dimensional configuration spaces based on locality-sensitive hashing and derive tight bounds on their accuracy. The k-NN queries are used to perform instance-based learning and have a sub-linear time complexity. Our approach is general, makes no assumption about the sampling scheme, and can be used with various sample-based motion planners, including PRM, Lazy-PRM, RRT and RRT*, by making small changes to these planners. We observe up to 100% improvement in the performance of various planners on rigid and articulated robots.

1 Introduction

Motion planning is an important problem in robotics, virtual prototyping and related areas. Most of the practical methods for motion planning of high-DOF (degrees-of-freedom) robots are based on random sampling in configuration spaces, including PRM [14] and RRT [15]. The resulting algorithms avoid explicit computation of obstacle boundaries in the configuration space ($\mathscr{C}$-space) and use sampling techniques to compute paths in the free space ($\mathscr{C}_{\text{free}}$). The main computations include probing the configuration space for collision-free samples, joining nearby collision-free samples by local paths, and checking whether the local paths lie in the free space.

Jia Pan and Dinesh Manocha are with the Department of Computer Science, the University of North Carolina, Chapel Hill {panj,dm}@cs.unc.edu. Sachin Chitta is with Willow Garage Inc., Menlo Park, CA 94025, USA sachinc@willowgarage.com

There is extensive work on different sampling strategies, faster collision checking, or biasing the samples based on local information to handlle narrow passages.

The collision detection module is used as an oracle to collect information about the free space and approximate its topology. This module is used to classify a given configuration or a local path as either collision-free (i.e. in $\mathscr{C}_{\text{free}}$) or in-collision (i.e. overlaps with $\mathscr{C}_{\text{obs}}$). Most motion planning algorithms tend to store only the collision-free samples and local paths, and use them to compute a global path from the initial configuration to the goal configuration. However, the in-collision configurations or local paths are typically discarded.

One of our goals is to exploit all prior or historical information related to collision queries and improve the performance of the sample-based planner. Some planners tend to utilize the in-collision configurations or the samples near the boundary of the configuration obstacles ($\mathscr{C}_{\text{obs}}$) to bias the sample generation or improve the performance of planners in narrow passages [4, 9, 19, 24]. However, it can be expensive to perform geometric reasoning based on the outcome of a large number of collision queries in high-dimensional spaces. As a result, most prior planners only use partial or local information about configuration spaces, and can't provide any guarantees in terms of improving the overall performance.

Main Results: We present a novel approach to improve the performance of sample-based planners by learning from prior instances of collision checking, including all in-collision samples. Our formulation uses the historical information generated using collision queries to compute an approximate representation of $\mathscr{C}$-space as a hash table. Given a new probe or collision query in $\mathscr{C}$-space, we first perform instance-based learning on the approximate $\mathscr{C}$-space and compute a collision probability. This probability is used as a similarity result or a prediction of the exact collision query and can improve the efficiency of a planner in the following ways:

- The probability is used to design a collision filter for a local planning query in high-dimensional configuration spaces.
- Explore the $\mathscr{C}$-space around a given configuration and select a new sample (e.g. for RRT planners [15]).
- Compute an efficient order to perform local planning queries along a path in the graph (e.g. for lazyPRM planners [14]).

The underlying instance-based learning is performed on approximate $\mathscr{C}$-space using k-NN (k-nearest neighbor) queries. All the prior configurations used by the planning algorithm and their collision outcomes are stored incrementally in a hash table. Given a new configuration or a local path, our algorithm computes the nearest neighbors in the hash table. We use locality-sensitive hashing (LSH) algorithms to perform approximate k-NN computations in high-dimensional configuration spaces. Specifically, we present a line-point k-NN algorithm that can compute the nearest neighbors of a line. We derive bounds on the accuracy and time complexity of k-NN algorithms and show that the collision probability computed using LSH-based k-NN algorithm converges to exact collision detection as the size of dataset increases.

We present improved versions of PRM, lazyPRM, RRT planning algorithms based on instance-based learning. Our approach is general and can be combined with any sampling scheme. Furthermore, it is quite efficient for high-dimensional configuration spaces. We have applied these planners to rigid and articulated robots, and observe up to 100% speedups based on instance-based learning. The additional overheads are in terms of storing the prior instances in a hash table and performing k-NN queries, which take a small fraction of the overall planning time.

The rest of the paper is organized as follows. We survey related work in Section 2. Section 3 gives an overview of sample-based planners, instance-based learning and our approach. We present the new k-NN and learning algorithms and analyze their accuracy and complexity in Section 4. We show the integration of instance-based learning with different motion planning algorithms in Section 5 and highlight the performance of modified planners on various benchmarks in Section 6.

2 Related Work

In this section, we give a brief overview of prior work on the use of machine learning techniques in motion planning and performing efficient collision checking to accelerate sample-based motion planning.

2.1 Machine Learning in Motion Planning

Many techniques have been proposed to improve the performance of sample-based motion planning based on machine learning. Marco et al. [18] combine a set of basic PRM motion planners into a powerful 'super' planner by assigning different basic planners to different regions in $\mathscr{C}$-space, based on offline supervised learning. Some unsupervised learning approaches construct an explicit or implicit representation of $\mathscr{C}_{\text{free}}$ and perform adaptive sampling based on this model. Burns and Brock [5] use entropy to measure each sample's utility to improve the coverage of PRM roadmap. Hsu et al. [12] adaptively combine multiple sampling strategies to improve the roadmap's connectivity. Some variants of RRT, which use workspace or taskspace bias (e.g., [10]), can be extended by changing the bias parameters adaptively. Scholz and Stilman [22] combine RRT with reinforcement learning. Given a sufficient number of collision-free samples in narrow passage, learning techniques have been used to estimate a zero-measure subspace to bias the sampling in narrow passages [7]. Our approach is complimentary to all these techniques.

2.2 Collision Detection Oracle in Motion Planning

Some of the previous approaches tend to exploit the knowledge about $\mathscr{C}$-space gathered using collision checking. Boor et al. [4] use pairs of collision-free and in-collision samples to collect information about $\mathscr{C}_{\text{obs}}$ and perform dense sampling near $\mathscr{C}_{\text{obs}}$. The same idea is used in many variants of RRT, such as retraction-based planners [19]. Sun et al. [24] bias sampling near narrow passages by using two in-collision samples and one collision-free sample to identify narrow passages in $\mathscr{C}$-space. Kavraki et al. [14] use in-collision samples to estimate the visibility of each sample and perform heavier sampling in regions with small visibility. Denny and Amato [9] present a variation of PRM that memorizes in-collision samples and constructs roadmaps in both $\mathscr{C}_{\text{free}}$ and $\mathscr{C}_{\text{obs}}$, in order to generate more samples in narrow passages. All of these methods utilize in-collision samples to provide better sampling strategies for the planners, in the form of different heuristics. Our approach neither makes assumptions about the underlying sampling scheme nor biases the samples. As a result, our algorithm can be combined with these methods.

2.3 *k*-Nearest Neighbor (*k*-NN) Search

The problem of finding the *k*-nearest neighbor within a database of high-dimensional points is well-studied in various areas, including databases, computer vision, and machine learning. Samet's book [21] provides a good survey of various techniques used to perform *k*-NN search. In order to handle large and high-dimensional spaces, most practical algorithms are based on approximate *k*-NN queries [6]. In these formulations, the algorithm is allowed to return a point whose distance from the query point is at most $1+\varepsilon$ times the distance from the query to its *k*-nearest points; $\varepsilon > 1$ is called the *approximation factor*.

3 Overview

In this section, we give an overview of the sample-based planner and provide a brief background on instance-based learning.

3.1 Notations and Symbols

We denote the configuration space as $\mathscr{C}$-space, and each point within the space as a configuration $\mathbf{x}$. $\mathscr{C}$-space is composed of two parts: the collision-free points ($\mathscr{C}_{\text{free}}$) and the in-collision points ($\mathscr{C}_{\text{obs}}$). $\mathscr{C}$-space may be non-Euclidean, but it is possible to approximately embed a non-Euclidean space into a higher-dimensional Euclidean space (e.g., using the Linial-London-Robinovich embed [17]) to perform *k*-nearest neighbor queries. A *local path* in $\mathscr{C}$-space is a continuous curve that connects two configurations. It is difficult to compute $\mathscr{C}_{\text{obs}}$ or $\mathscr{C}_{\text{free}}$ explicitly, therefore sample-based planners use collision checking between the robot and obstacles to probe the

$\mathscr{C}$-space implicitly. These planners perform two kinds of queries: the *point query* and the *local path query*. We use the symbol Q to denote either of these queries.

3.2 Enhance Motion Planner with Instance-based Learning

The goal of a motion planner is to compute a collision-free continuous path between the initial and goal configurations in $\mathscr{C}$-space. The resulting path should completely lie in $\mathscr{C}_{\text{free}}$ and should not intersect with $\mathscr{C}_{\text{obs}}$. As shown in Figure 1(a), sample-based planners learn about the connectivity of $\mathscr{C}$-space implicitly based on collision queries. The query results can also be used to bias the sampling scheme of the planner via different heuristics (e.g., retraction rules).

Instance-based learning is a well known family of algorithms in machine learning that learn properties of new problem instances by comparing them with the instances observed earlier that have been stored in memory [20]. In our case, we store all the results of prior collision queries, including collision-free as well as in-collision queries. Our goal is to sufficiently exploit such prior information and accelerate the planner computation. The problem instance in our context is the collision query being performed on a given configuration or a local path in $\mathscr{C}$-space. In particular, performing exact collision queries for local planning can be expensive. Collision checking for a discrete configuration is relatively cheap, but still can be time consuming if the environment or robot's geometric representation has a high complexity. We utilize the earlier instances or the stored information by performing k-NN queries and geometric reasoning on query results.

Our new approach to exploit prior information for motion planning is shown in Figure 1(b). When the collision checking routine finishes probing the $\mathscr{C}$-space for a given query, it stores all the obtained information in a dataset corresponding to historical collision query results. If the query is a point within $\mathscr{C}$-space, the stored information is its binary collision status. If the query is a local path, the stored information includes the collision status of different configuration points along the path. The resulting dataset of historical collision results constitutes the complete knowledge we have about $\mathscr{C}$-space, coming from collision checking routines. Therefore, we use it as an approximate description of the underlying $\mathscr{C}$-space: the in-collision samples are an approximation of $\mathscr{C}_{\text{obs}}$, while the collision-free samples are used to encode $\mathscr{C}_{\text{free}}$. These samples are used by instance-based learning algorithms to estimate the collision status of new queries.

Given a new query Q, either a point or a local path, we first perform k-NN search on the dataset to find its neighbor set S, which provides information about the local $\mathscr{C}$-space around the query. If S contains sufficient information to infer the collision status of the query, we compute a collision probability for the new query based on S; otherwise we perform exact collision checking for this query. The calculated collision probability provides prior information about the collision status of the given query and is useful in many ways. First, it can be used as a culling filter to avoid the exact (and expensive) collision checking for queries that correspond to the configu-

Algorithm 1: learning-based-collision-query(Q)

```
begin
    if $Q$ is point query then
        $S \leftarrow$ point-point-$k$-NN($Q$)
        if $S$ provides sufficient information for reasoning then
            approximate-collision-query($S$, $Q$)
        else exact-collision-query($S$, $Q$)
    if $Q$ is line query then
        $S \leftarrow$ line-point-$k$-NN($Q$)
        if $S$ provides sufficient information for reasoning then
            approximate-continuous-collision-query($S$, $Q$)
        else exact-continuous-collision-query($S$, $Q$)
end
```

rations or local paths deep inside $\mathscr{C}_{\text{free}}$ or $\mathscr{C}_{\text{obs}}$. Secondly, it can be used to decide an efficient order to perform exact collision checking for a set of queries. For example, many planners like RRT need to select a local path that can best improve the local exploration in $\mathscr{C}_{\text{free}}$, i.e., a local path with a long length but is also collision-free. The collision probability computation can be used to compute an efficient sorting strategy and thereby reduces the number of exact collision tests.

Whether we have sufficient information about S (in Algorithm 1) is related to how much confidence we have in terms of performing inferencing from S. For example, if there exists an in-collision sample very close to the new query, then there is a high probability that the new query is also an in-collision query. The probability decreases when the distance between the sample and the query increases. A description of our prior instance based collision framework is given in Algorithm 1, which is used as an inexpensive routine to perform probabilistic collision detection. More details about this routine and its applications are given in Section 4.

3.3 LSH-based Approximate k-NN Query

A key issue in terms of the instance-based learning framework is its computational efficiency. As we generate hypotheses directly from training instances, the complexity of this computation can grow with the size of historical data. If we use exact k-NN computation as the learning method, its complexity can be a linear function of the size of the dataset, especially for high-dimensional spaces. To improve the efficiency of the instance-based learning, we use approximate k-NN algorithms.

Given a dataset $\mathscr{D} = \{\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_N\}$ of N points in $\mathscr{R}^d$, we consider two types of retrieval queries. One is to retrieve points from $\mathscr{D}$ that are closest to a given point query, i.e. the well-known k-NN query, and we call it the *point-point k-NN* query. The second query tries to find the points from $\mathscr{D}$ that are closest to a given line in $\mathscr{R}^d$, whose direction is $\mathbf{v}$ and passes through a point $\mathbf{a}$, where $\mathbf{v}, \mathbf{a} \in \mathscr{R}^d$. We call

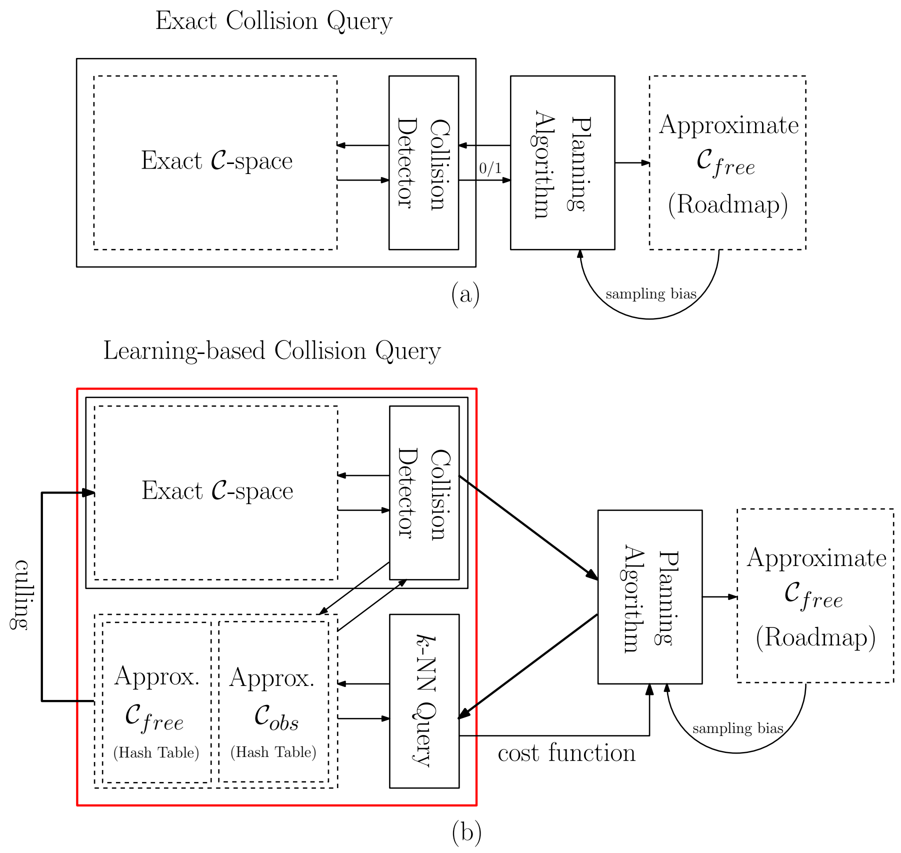
Exact Collision Query
Exact $\mathcal{C}$-space
Collision Detector
0/1
Planning Algorithm
Approximate $\mathcal{C}_{free}$ (Roadmap)
sampling bias
(a)
Learning-based Collision Query
Exact $\mathcal{C}$-space
Collision Detector
culling
Approx. $\mathcal{C}_{free}$ (Hash Table)
Approx. $\mathcal{C}_{obs}$ (Hash Table)
k-NN Query
cost function
Planning Algorithm
Approximate $\mathcal{C}_{free}$ (Roadmap)
sampling bias
(b)

Fig. 1: Comparison between how collision checking is used in prior approaches (a) and our method (b). (a) The collision detection routine is the oracle used by the planner to gather information about $\mathscr{C}_{\text{free}}$ and $\mathscr{C}_{\text{obs}}$. The planner performs binary collision queries, either on point configurations or 1-dimensional local paths, and estimates the connectivity of $\mathscr{C}_{\text{free}}$ (shown as Approximate $\mathscr{C}_{\text{free}}$). Moreover, some planners utilize the in-collision results to bias sample generation by using different heuristics. (b) Our method also performs collision queries. However, we store all in-collision results (as Approximate $\mathscr{C}_{\text{obs}}$) in addition to collision-free results (as Approximate $\mathscr{C}_{\text{free}}$). Before performing an exact collision query, our algorithm performs a k-NN query on the given configuration or local path to compute a collision probability for each query. The collision probability can be used as a cost function to compute an efficient strategy to perform exact collision queries during motion planning. We use novel LSH-based algorithms to perform k-NN queries efficienty and speed up the overall planner.

the second query the *line-point k-NN* query. The two types of k-NN queries are illustrated in Figure 2.

In order to develop an efficient instance-based learning framework, we use locality-senstive hashing (LSH) as an approximate method for k-NN queries, which is mainly designed for point-point queries [1]. However, it can be extended to line queries [2] and hyperplane queries [13]. Basri [3] et al. further extend it to perform point/subspace queries.

LSH requires randomized hash functions, which can guarantee that the probability of two points being mapped into the same hash bucket is inversely proportional to the distance between them. In these cases, the distance metric is defined based on the specific task or application. Since two similar points are likely to fall into nearby hash buckets, we only need to perform local search within the bucket that contains the given query.

Formally, let $\mathrm{dist}(\cdot,\cdot)$ be a distance metric over the items from $\mathscr{D}$, and for any item $\mathbf{x} \in \mathscr{D}$, let $B(\mathbf{x}, r)$ denote the set of elements from $\mathscr{D}$ within radius r from $\mathbf{x}$.

Definition 1. [1] Let $h_{\mathscr{H}}$ denote a random choice of hash functions from the function family $\mathscr{H}$ and $B(\mathbf{x}, r)$ is a radius-r ball centered at $\mathbf{x}$. $\mathscr{H}$ is called $(r, r(1+\varepsilon), p_1, p_2)$-sensitive for $\mathrm{dist}(\cdot,\cdot)$ when for any $\mathbf{x}$, $\mathbf{y} \in \mathscr{D}$,

- if $\mathbf{y} \in B(\mathbf{x}, r)$, then $\mathbb{P}[h_{\mathscr{H}}(\mathbf{x}) = h_{\mathscr{H}}(\mathbf{y})] \geq p_1$,
- if $\mathbf{y} \notin B(\mathbf{x}, r(1+\varepsilon))$, then $\mathbb{P}[h_{\mathscr{H}}(\mathbf{x}) = h_{\mathscr{H}}(\mathbf{y})] \leq p_2$,

For a family of functions to be useful, we require $p_1 > p_2$. A dimension-M LSH function computes a hash 'key' by concatenating the results returned by a random sampling of $\mathscr{H}$: $g(\mathbf{x}) = [h^{(1)}_{\mathscr{H}}(\mathbf{x}), h^{(2)}_{\mathscr{H}}(\mathbf{x}), ..., h^{(M)}_{\mathscr{H}}(\mathbf{x})]$ and therefore the hashing collision probability for two close points is at least p_1^M, while for dissimilar points it is at most p_2^M. Each item in the dataset $\mathscr{D}$ is mapped to a series of L hash tables indexed using independently constructed $g_1, ..., g_L$, where each g_i is a dimension-M function. Next, given a point query $\mathbf{y}$, an exhaustive search is carried out only on the items in the union of the L buckets. These candidates constitute the (r, ε)-nearest neighbor for $\mathbf{y}$, meaning that if $\mathbf{y}$ has a neighbor within radius r, then with high probability some item within radius $r(1+\varepsilon)$ would be found. When $\mathrm{dist}(\cdot,\cdot)$ corresponds to the l_2 metric, we have following result about point-point k-NN query:

Theorem 1. *(Point-point k-NN query) [8] Let $\mathscr{H}$ be a family of $(r, r(1+\varepsilon), p_1, p_2)$-sensitive hash functions, with $p_1 > p_2$. Given a dataset of size N, we set $M = \log_{1/p_2} N$ and $L = N^{\rho}$, where $\rho = \frac{\log p_1}{\log p_2}$. Using $\mathscr{H}$ along with L-hash tables over M-dimensions, given a point query $\mathbf{y}$, with probability at least $\frac{1}{2} - \frac{1}{e}$, the LSH algorithm solves the (r, ε)-neighbor problem. In other words, if there exists a point $\mathbf{x}$ that $\mathbf{x} \in B(\mathbf{y}, r(1+\varepsilon))$, then the algorithm will return the point with probability $\geq \frac{1}{2} - \frac{1}{e}$. The retrieval time is bounded by $\mathscr{O}(N^{\rho})$.*

If we choose $\mathscr{H}$ to be the hamming hash or p-stable hash:

$$\{h^{\mathbf{u}} : h^{\mathbf{u}}(\mathbf{x}) = \mathrm{sgn}(\mathbf{u}^T \mathbf{x})\} \text{ or } \{h^{\mathbf{a},b} \quad : h^{\mathbf{a},b}(\mathbf{x}) = \lfloor \frac{\mathbf{a}^T \mathbf{x} + b}{W} \rfloor\}, \tag{1}$$

where $\mathbf{u}$ and $\mathbf{a} \sim \mathscr{N}(\mathbf{0}, \mathbf{I})$, $b \sim \mathscr{U}[0, W]$ and W is a fixed constant, we have $\rho \leq \frac{1}{1+\varepsilon}$ and the algorithm has sublinear complexity, i.e., the results can be retrieved in time $\mathscr{O}(N^{\frac{1}{1+\varepsilon}})$.

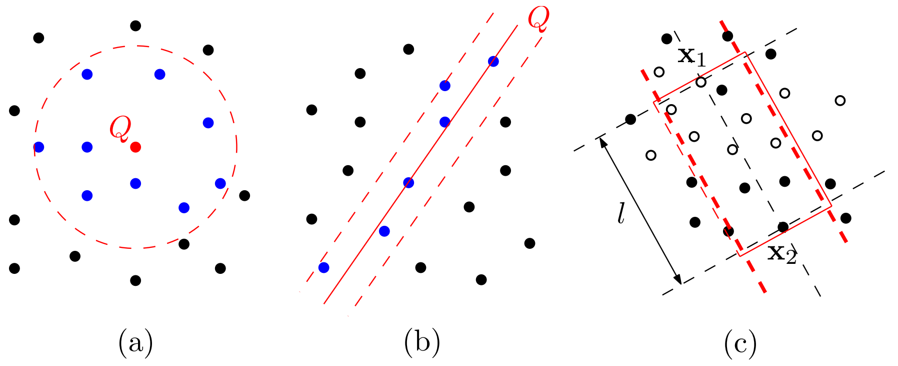

Fig. 2: Two types of k-NN queries used in our method: (a) point-point k-NN; (b) line-point k-NN. Q is the query item and the results of different queries are shown as blue points in each figure. We present novel LSH-based algorithms for fast computation of these queries. (c) Use line-point k-NN query to compute prior instances that can influence the collision status of a local path, which connects $\mathbf{x}_1$ and $\mathbf{x}_2$ in $\mathscr{C}$-space. The query line is the line segment between $\mathbf{x}_1$ and $\mathbf{x}_2$. The white points are prior collision-free samples in the dataset and the black points are prior in-collision samples.

We build on these prior results for point-point k-NN queries and present a new LSH-based algorithm for line-point k-NN queries by defining the Euclidean embedding for line queries and point datasets. Later in Section 4.1, we discuss issues in designing appropriate hash functions for line-point k-NN queries and derive LSH bounds for line-point k-NN, which are indeed similar to Theorem 1. We also address many issues in extending our formulation to non-Euclidean metrics (e.g., in handling articulated models) and reducing the dimension of embedded space.

4 Probabilistic Collision Detection based on Instance-based Learning

In this section, we first present our LSH-based line-point k-NN query algorithm and analyze its properties. Next, we use this line-point k-NN query to estimate the collision probability for a given query.

4.1 LSH-based Line-Point k-NN Query

One of the contributions of this paper is to extend the LSH formulation to the line-point k-NN by defining an Euclidean embedding for line queries and point datasets. In comparison with previous methods [2, 3], our line-point k-NN results in a more compact representation and we also derive LSH bounds similar to the point-point

k-NN case in Theorem 1. Moreover, we address several issues that arise in terms of using this algorithm for sample-based motion planning, such as handling non-Euclidean metrics and reducing the dimension of the embedded space.

The main issue in terms of using LSH to perform line-point k-NN query is to compute an Euclidean embedding for the line query and the point dataset, and then perform point-point k-NN query in the embedded space. First, we present a technique to perform line-point embedding. Next, we design hash functions for the embedding and prove that these hash functions satisfy the locality-sensitive property for the original data (i.e., $\mathscr{D}$). Finally, we derive the error bound and time bound for approximate line-point k-NN query, similar to what is derived in Theorem 1.

4.1.1 Line-point Embedding

A line l in $\mathscr{R}^d$ is described as $l = \{\mathbf{a} + \lambda \cdot \mathbf{v}\}$, where $\mathbf{a}$ is a point in $\mathscr{R}^d$ and $\mathbf{v}$ is a unit vector in $\mathbf{R}^d$. The Euclidean distance of a point $\mathbf{x} \in \mathscr{R}^d$ to the line l is given as:

$$\mathrm{dist}(\mathbf{x}, l) = (\mathbf{x} - \mathbf{a}) \cdot (\mathbf{x} - \mathbf{a}) - ((\mathbf{x} - \mathbf{a}) \cdot \mathbf{v})^2. \tag{2}$$

Given a database $\mathscr{D} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ of N points in $\mathscr{R}^d$, the goal of line-point k-NN query is to retrieve points from $\mathscr{D}$ that are closest to l.

We first assume that l passes through the origin (i.e. $\mathbf{a} = \mathbf{0}$) and therefore $\mathrm{dist}(\mathbf{x}, l) = \mathbf{x} \cdot \mathbf{x} - (\mathbf{x} \cdot \mathbf{v})^2$. This distance can be reformalized as the inner product of two $(d+1)^2$-dimensional vectors:

$$\begin{aligned}
&\mathrm{dist}^2(\mathbf{x}, l) \\
&= \mathbf{x} \cdot \mathbf{x} - (\mathbf{x} \cdot \mathbf{v})^2 = \mathbf{x}^T \mathbf{I} \mathbf{x} - \mathbf{x}^T \mathbf{v} \mathbf{v}^T \mathbf{x} \\
&= \mathbf{x}^T (\mathbf{I} - \mathbf{v}\mathbf{v}^T) \mathbf{x} = \mathrm{Tr}(\mathbf{x}^T (\mathbf{I} - \mathbf{v}\mathbf{v}^T) \mathbf{x}) \\
&= \mathrm{Tr}(\begin{pmatrix} \mathbf{x} \\ t \end{pmatrix}^T \big(\mathbf{I}\ \mathbf{0}\big)^T (\mathbf{I} - \mathbf{v}\mathbf{v}^T) \big(\mathbf{I}\ \mathbf{0}\big) \begin{pmatrix} \mathbf{x} \\ t \end{pmatrix}) \\
&= \mathrm{Tr}(\big(\mathbf{I}\ \mathbf{0}\big)^T (\mathbf{I} - \mathbf{v}\mathbf{v}^T) \big(\mathbf{I}\ \mathbf{0}\big) \begin{pmatrix} \mathbf{x} \\ t \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ t \end{pmatrix}^T) \\
&= \mathrm{vec}(\big(\mathbf{I}\ \mathbf{0}\big)^T (\mathbf{I} - \mathbf{v}\mathbf{v}^T) \big(\mathbf{I}\ \mathbf{0}\big)) \cdot \mathrm{vec}(\begin{pmatrix} \mathbf{x} \\ t \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ t \end{pmatrix}^T) \\
&= V(\mathbf{v}) \cdot V(\mathbf{x}),
\end{aligned} \tag{3}$$

where $\mathbf{I}$ is $d \times d$ identity matrix, $\mathrm{Tr}(\cdot)$ is the trace of a given square matrix and t can be any real value. $\mathrm{vec}(\cdot)$ is the vectorization of a given matrix. Specifically, the vectorization of an $m \times n$ matrix $\mathbf{A}$, denoted by $\mathrm{vec}(\mathbf{A})$, is the $mn \times 1$ column vector obtain by stacking the columns of the matrix $\mathbf{A}$ on top of one another:

$$\mathrm{vec}(\mathbf{A}) = [a_{1,1}, ..., a_{m,1}, a_{1,2}, ..., a_{m,2}, ..., a_{1,n}, ..., a_{m,n}]^T, \tag{4}$$

where $a_{i,j}$ represents the (i,j)-th element of matrix $\mathbf{A}$. $V(\cdot)$ is an embedding which yields a $(d+1)^2$-dimensional vector from d-dimensional vector:

$$V(\mathbf{z}) = \begin{cases} \mathrm{vec}(\begin{pmatrix}\mathbf{z}\\ t\end{pmatrix}\begin{pmatrix}\mathbf{z}\\ t\end{pmatrix}^T), & \text{if } \mathbf{z} \text{ is a database point,} \\ \mathrm{vec}(\begin{pmatrix}\mathbf{I}-\mathbf{z}\mathbf{z}^T & \mathbf{0}\\ \mathbf{0}^T & 0\end{pmatrix}), & \text{if } \mathbf{z} \text{ is a query line vector.} \end{cases} \tag{5}$$

Moreover, we notice that the Euclidean distance between the embeddings $V(\mathbf{x})$ and $-V(\mathbf{v})$ is given by

$$\begin{aligned} \|V(\mathbf{x}) - (-V(\mathbf{v}))\|^2 &= d - 1 + 2(V(\mathbf{x})\cdot V(\mathbf{v})) + \|V(\mathbf{x})\|^2 \\ &= d - 1 + 2\,\mathrm{dist}^2(\mathbf{x}, l) + (\|\mathbf{x}\|^2 + t^2)^2. \end{aligned} \tag{6}$$

Since t can be chosen arbitrarily, we choose t as a function of $\mathbf{x}$ so that $V(\mathbf{x})$ a vector of constant length: if we choose $t(\mathbf{x}) = \sqrt{c - \|x\|^2}$, where $c > \max_{\mathbf{x}\in\mathscr{D}} \|\mathbf{x}\|^2$ is a constant real value, then $\|V(\mathbf{x})\|^2 = c^2$ and Equation 6 reduces to $\|V(\mathbf{x}) - (-V(\mathbf{v}))\|^2 = 2\,\mathrm{dist}^2(\mathbf{x}, l) + \text{constant}$. This implies that we can reduce the line-point k-NN query in a d-dimensional database $\mathscr{D}$ to a point k-NN query in a $(d+1)^2$-dimensional embedded database $V(\mathscr{D}) = \{V(\mathbf{x}_1), ..., V(\mathbf{x}_N)\}$, where the query item corresponds to $-V(\mathbf{v})$.

Now we consider the case of any arbitrary affine line, i.e. $\mathbf{a} \neq \mathbf{0}$. Similar to Equation 3, we obtain

$$\begin{aligned} &\mathrm{dist}^2(\mathbf{x}, l) \\ &= (\mathbf{x}-\mathbf{a})\cdot(\mathbf{x}-\mathbf{a}) - ((\mathbf{x}-\mathbf{a})\cdot\mathbf{v})^2 \\ &= \mathrm{Tr}((\mathbf{x}-\mathbf{a})^T(\mathbf{I}-\mathbf{v}\mathbf{v}^T)(\mathbf{x}-\mathbf{a})) \\ &= \mathrm{Tr}(\begin{pmatrix}\mathbf{x}\\ 1\\ t\end{pmatrix}^T \underbrace{\begin{pmatrix}\mathbf{I} & -\mathbf{a} & \mathbf{0}\end{pmatrix}^T(\mathbf{I}-\mathbf{v}\mathbf{v}^T)\begin{pmatrix}\mathbf{I} & -\mathbf{a} & \mathbf{0}\end{pmatrix}}_{\mathbf{B}} \begin{pmatrix}\mathbf{x}\\ 1\\ t\end{pmatrix}) \\ &= \mathrm{vec}(\begin{pmatrix}\mathbf{x}\\ 1\\ t\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ 1\\ t\end{pmatrix}^T)\cdot\mathrm{vec}(\mathbf{B}) \\ &= \hat{V}(\mathbf{x})\cdot\hat{V}(\mathbf{v},\mathbf{a}), \end{aligned} \tag{7}$$

where $\hat{V}(\mathbf{x})$ and $\hat{V}(\mathbf{v},\mathbf{a})$ are $(d+2)^2$-dimensional embeddings for a point and line in $\mathscr{R}^d$, respectively. Similar to Equation 6, if we choose $t(\mathbf{x}) = \sqrt{c - \mathbf{x}^2 - 1}$, where $c > \max_{\mathbf{x}\in\mathscr{D}} \|\mathbf{x}\|^2 + 1$ is a constant (i.e., set $\|\hat{V}(\mathbf{x})\|^2 = c^2$), then $\mathrm{dist}^2(\mathbf{x}, l)$ also linearly depends on the Euclidean distance between the embedded database and the query item: $\|\hat{V}(\mathbf{x}) - \hat{V}(\mathbf{v},\mathbf{a})\|^2 = c^2 + d - 2 + (\mathrm{dist}^2(\mathbf{0}, l) + 1)^2 + 2\,\mathrm{dist}^2(\mathbf{x}, l)$. As a result, we can perform affine line-point k-NN query based on a point k-NN query in a

$(d+2)^2$-dimensional database $\hat{V}(\mathscr{D}) = \{\hat{V}(\mathbf{x}_1), ..., \hat{V}(\mathbf{x}_N)\}$ and the corresponding query item is $-\hat{V}(\mathbf{v}, \mathbf{a})$.

The dimension of the embedded space (i.e., $(d+1)^2$ or $(d+2)^2$) is much higher than the original space (i.e., d), and would therefore slow down the LSH computation. We present two techiques to reduce the dimension of this query.

First, notice that the matrices used within $\mathrm{vec}(\cdot)$ tend to be symmetrical matrices. For a $d \times d$ matrix $\mathbf{A}$, we can define a $d(d+1)/2$-dimensional embedding $\widehat{\mathrm{vec}}(\mathbf{A})$ as follows

$$\widehat{\mathrm{vec}}(\mathbf{A}) = [\frac{a_{1,1}}{\sqrt{2}}, a_{1,2}, ..., a_{1,d}, \frac{a_{2,2}}{\sqrt{2}}, a_{2,3}, ..., \frac{a_{d,d}}{\sqrt{2}}]^T, \tag{8}$$

It is easy to see that $\|\mathrm{vec}(\mathbf{A}) - \mathrm{vec}(\mathbf{B})\|^2 = 2\|\widehat{\mathrm{vec}}(\mathbf{A}) - \widehat{\mathrm{vec}}(\mathbf{B})\|^2$ and hence this dimension-reduction will not influence the accuracy of the line-point k-NN algorithm introduced above.

Secondly, we can use the Johnson-Lindenstrauss lemma to reduce the dimension of the embedded data by randomly projecting onto a lower dimensional space [16]. Compared to the first approach, this method can generate an embedding with lower dimensions, but it may reduce the accuracy of the line-point k-NN algorithm.

4.1.2 Locality-Sensitive Hash Functions for Line-Point Query

We design the hash function $\hat{h}$ for the line-point query as follows:

$$\begin{cases} \hat{h}(\mathbf{x}) = h(\hat{V}(\mathbf{x})), & \mathbf{x} \text{ is a database point} \\ \hat{h}(l) = h(-\hat{V}(\mathbf{v}, \mathbf{a})), & l \text{ is the line } \{\mathbf{a} + \lambda \cdot \mathbf{v}\}, \end{cases} \tag{9}$$

where h is the hamming hash functions or a p-stable hash functions defined in Equation 1. The new hash functions are locality-sensitive for line-point query, as shown by the following two theorems:

Theorem 2. *The hash function family $\hat{h}$ is $(r, r(1+\varepsilon), p_1, p_2)$-sensitive if h is the hamming hash, (i.e. $h = h^{\mathbf{u}}$), where $p_1 = \frac{1}{\pi}\cos^{-1}(\frac{r^2}{C})$, $p_2 = \frac{1}{\pi}\cos^{-1}(\frac{r^2(1+\varepsilon)^2}{C})$ and C is a value independent of database point, but is related to the query. Moreover, $\frac{1}{(1+\varepsilon)^2} \leq \rho = \frac{\log p_1}{\log p_2} \leq 1$.*

Theorem 3. *The hash function family $\hat{h}$ is $(r, r(1+\varepsilon), p_1, p_2)$-sensitive if h is the p-stable hash, (i.e. $h = h^{\mathbf{a},b}$), where $p_1 = f(\frac{W}{\sqrt{2r^2+C}})$ and $p_2 = f(\frac{W}{\sqrt{2r^2(1+\varepsilon)^2+C}})$ and C is a value independent of database point, but is related to the query. The function f is defined as $f(x) = \frac{1}{2}(1 - 2\,\mathrm{cdf}(-x)) + \frac{1}{\sqrt{2\pi}x}(e^{-\frac{1}{2}x^2} - 1)$, where* $\mathrm{cdf}(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2} dt$. *Moreover, $\frac{1}{1+\varepsilon} \leq \rho = \frac{\log p_1}{\log p_2} \leq 1$.*

The proofs are given in Appendix 1 and 2.

Similar to point-point k-NN query, we can compute the error bound and time complexity for line-point k-NN query as follows:

Theorem 4. *(Line-point k-NN query) Let $\mathscr{H}$ be a family of $(r, r(1+\varepsilon), p_1, p_2)$-sensitive hash functions, with $p_1 > p_2$. Given a dataset of size N, we set $M = \log_{1/p_2} N$ and $L = N^{\rho}$, where $\rho = \frac{\log p_1}{\log p_2}$. Using $\mathscr{H}$ along with L-hash tables over M-dimensions, given a line query l, with probability at least $\frac{1}{2} - \frac{1}{e}$, our LSH algorithm solves the (r, ε)-neighbor problem, i.e., if there exists a point $\mathbf{x}$ that $\mathrm{dist}(x, l) \leq r(1+\varepsilon)$, then the algorithm will return the point with probability $\geq \frac{1}{2} - \frac{1}{e}$. The retrieval time is bounded by $\mathscr{O}(N^{\rho})$.*

The proof is given in Appendix 3.

Theorem 4 along with Theorem 1 guarantees sublinear time complexity of performing instance-based learning on the historical collision results, if hamming or p-stable hashing functions are applied.

4.2 $\mathscr{C}$-space Reasoning based on k-NN Queries

Our approach stores the outcome of prior instances of exact collision queries, including point queries and local path queries, within a database (shown as Approximate $\mathscr{C}_{\text{free}}$ and Approximate $\mathscr{C}_{\text{obs}}$ in Figure 1(b)). In this section, we describe our approach to use those stored instances to perform probabilistic collision queries.

The first case is the query point, i.e., the task is to compute the collision status for a sample $\mathbf{x}$ in $\mathscr{C}$-space. We first perform point-point k-NN query to compute the prior collision instances closest to $\mathbf{x}$. Next, we use a simple k-NN classifier to estimate $\mathbf{x}$'s *collision probability* as

$$\mathbb{P}[\mathbf{x} \text{ in collision}] = \frac{\sum_{\mathbf{x}_i \in S \backslash \mathscr{C}_{\text{free}}} w_i}{\sum_{\mathbf{x}_i \in S} w_i}, \tag{10}$$

where $w_i = e^{-\lambda \, \mathrm{dist}(\mathbf{x}_i, \mathbf{x})}$ is the distance-tuned weight for each k-NN neighbor and S is the neighborhood set computed using point-point k-NN query. In this case, the parameter λ controls the magnitude of the weights. λ corresponds to the *obstacle density* of $\mathscr{C}_{\text{obs}}$, if we model $\mathscr{C}_{\text{obs}}$ as a point set generated using a Poission point process in the $\mathscr{C}$-space [23].

The second case is the line query, i.e., the goal is to estimate the collision status of a local path in $\mathscr{C}$-space. We require the local path to lie within the neighborhood of the line segment l connecting its two endpoints, i.e., the local path should not deviate too much from l. The first step is to perform a line-point k-NN query to find the prior collision query results closest to the infinite line that l lies on. Next, we need to filter out the points in S whose projections are outside the truncated segment of l, as shown in Figure 2(c). Finally, we apply our learning method on the filtered results, denoted as S, to estimate the collision probability of the local path.

We compute the line's collision probability via an optimization scheme. The line l is divided into I segments and we assign each segment, say l_i, a label c_i to indicate its collision status: we assign label $c_i = 0$ if l_i is collision-free and assign label $c_i = 1$ if it is in-collision. Given line l's neighborhood set S computed using the line-point

k-NN query, we now compute the label assignments for these segments. First, we compute the conditional collision probability of one point $\mathbf{x}_j \in S$, given the collision status of one line segment l_i:

$$\mathbb{P}[\mathbf{x}_j \text{ in collision} \mid c_i] = \begin{cases} 1 - e^{-\lambda\, \mathrm{dist}(l_i, \mathbf{x}_j)}, & c_i = 0, \\ e^{-\lambda\, \mathrm{dist}(l_i, \mathbf{x}_j)}, & c_i = 1; \end{cases} \tag{11}$$

$$\mathbb{P}[\mathbf{x}_j \text{ collision free} \mid c_i] = \begin{cases} e^{-\lambda\, \mathrm{dist}(l_i, \mathbf{x}_j)}, & c_i = 0, \\ 1 - e^{-\lambda\, \mathrm{dist}(l_i, \mathbf{x}_j)}, & c_i = 1, \end{cases} \tag{12}$$

where $\mathrm{dist}(l_i, \mathbf{x}_j)$ is the distance between $\mathbf{x}_j$ and l_i's midpoint. Given this formalization, we can compute l_i's posterior collision probability. given l's neighborhood set S:

$$\mathbb{P}[c_i \mid S] \propto \mathbb{P}[S \mid c_i] \cdot \mathbb{P}[c_i] = \prod_{\mathbf{x}_j \in S} \mathbb{P}[\mathbf{x}_j\text{'s collision status} \mid c_i] \cdot \mathbb{P}[c_i]. \tag{13}$$

The error in this label assignment, i.e., the probability that the computed l_i's label is not the same as the outcome of the exact collision query, is

$$\mathbb{P}_{\text{error}}[c_i \mid S] = c_i \cdot \mathbb{P}[c_i = 0 \mid S] + (1 - c_i) \cdot \mathbb{P}[c_i = 1 \mid S]. \tag{14}$$

The label assignment algorithm needs to minimize this error probability for each segment l_i. Moreover, the assignment should be coherent, i.e., there is a high probability that adjacent line segments should have the same label. As a result, we can compute a suitable label assignment $\{c_i^*\}_{i=1}^I$ using a binary integer programming:

$$\{c_i^*\} = \mathrm{argmin}_{\{c_i\} \in \{0,1\}^I} \sum_{i=1}^{I} \mathbb{P}_{\text{error}}[c_i \mid S] + \kappa \sum_{i=1}^{I-1} (c_i - c_{i+1})^2, \tag{15}$$

where κ is a weight parameter. The binary integer programming problem can be solved efficiently using dynamic programming. After that, we can estimate the collision probability for the line as

$$\mathbb{P}[l \text{ in collision}] = \max_{i:\ c_i^* = 1} \mathbb{P}[c_i = 1 | S]. \tag{16}$$

As a byproduct, the approximate first time of contact can be given as $\min_{i:\ c_i^*=1} i/I$.

A natural way to use the collision probability formulated as above is to use a specific threshold to justify whether a given query is in-collision or not: if the query's collision probability is larger than the given threshold, we return in-collision; otherwise we return collision-free. We can prove that, for any threshold $0 < t < 1$, the collision status returned by the instance-based learning will converge to the exact collision detection results, when the size of the dataset increases (asymptotically):

Theorem 5. *The collision query performed using LSH-based k-NN will converge to the exact collision detection as the size of the dataset increases, for any threshold between 0 and 1.*

Proof. We only need to prove the probability of false positive (i.e., returns in-collision status when there is in fact no collision) and false negative (i.e., returns collision-free when there is in fact a collision) both converging to zero, as the size of the dataset increases.

Given a query, we denote its r-neighbor as B_r, where r is the distance between the query and its k-th nearest neighbor. For a point query, B_r is a r-ball around it. For a line query, B_r is the set of all points with distance r to the line (i.e., a line swept-sphere volume). Let $P_1 = (|B_{r(1+\varepsilon)} \cap \mathscr{C}_{\text{obs}}|)/|\mathscr{C}\text{-space}|$ and $P_2 = (|B_{r(1+\varepsilon)} \cap \mathscr{C}_{\text{free}}|)/|\mathscr{C}\text{-space}|$, which are the probabilities that a uniform sample in $\mathscr{C}$-space is in-collision or collision-free and within query's $r(1+\varepsilon)$-neighborhood. Let N be the size of the dataset corresponding to prior instances.

A false negative occurs if and only if there are no in-collision points within $B_{r(1+\varepsilon)}$. If we can find some in-collision points within $B_{r(1+\varepsilon)}$, then $\mathbb{P}[\mathbf{x} \text{ in collision}]$ and $\mathbb{P}[l \text{ in collision}]$ will converge to 1, as r converges to zero, if k is fixed and N increases. Hence $\mathbb{P}[\mathbf{x} \text{ in collision}] > t$ or $\mathbb{P}[l \text{ in collision}] > t$ always holds no matter what $t \in (0,1)$ are chosen and we can conclude that the query is in collision. The event that there are no in-collision points within $B_{r(1+\varepsilon)}$ happens either because no dataset point lies within $B_{r(1+\varepsilon)}$ or there exist some points within that ball, but they are missed due to the approximate nature of LSH-based k-NN query. As a result, we have

$$\begin{aligned}
&\mathbb{P}[\text{false negative}] \\
&= \sum_{i=0}^{N} \binom{N}{i} (1-P_1)^{N-i} P_1^i (1-(1/2-1/e))^i \\
&= (1-P_1(1/2-1/e))^N \to 0 \ (\text{as } N \to \infty).
\end{aligned}$$

Similarly, a false negative occurs if there are no collision-free points within $B_{r(1+\varepsilon)}$, and the probability of this can be given as $\mathbb{P}[\text{false positive}] \le (1-P_2(1/2-1/e))^N \to 0$ (as $N \to \infty$). $\square$

Remark 1. Note that the convergence of collision query using LSH-based k-NN query is slower than that using the exact k-NN based method ($\mathbb{P}[\text{false negative}] = (1-P_1)^N$ and $\mathbb{P}[\text{false positive}] \le (1-P_2)^N$).

5 Accelerating Sample-based Planners

In this section, we first discuss techniques to accelerate various sample-based planners based on instance-based learning. Next, we analyze the factors that can influence the performance of the learning-based planners. Finally, we prove the completeness and optimality for the instance-learning enhanced planners.

The basic approach to benefit from the learning framework is highlighted in Algorithm 1, i.e., use the computed collision probability as a filter to reduce the number of exact collision queries. If a given configuration or local path query is close to in-

```
$\mathrm{sample}(\mathscr{D}^{\mathrm{out}}, n)$
$V \leftarrow \mathscr{D} \cap \mathscr{C}_{\mathrm{free}}, E \leftarrow \emptyset$
foreach $v \in V$ do
  $U \leftarrow \mathrm{near}(G^{V,E}, v, \mathscr{D}^{\mathrm{in}})$
  foreach $u \in U$ do
    if $\mathrm{icollide}(v, u, \mathscr{D}^{\mathrm{in}}) < t$
      if $\neg\mathrm{collide}(v, u, \mathscr{D}^{\mathrm{out}})$
        $E \leftarrow E \cup (v, u)$
```

near: nearest neighbor search.
icollide: probabilistic collision checking based on k-NN.
collide: exact local path collision checking.
$\mathscr{D}^{\mathrm{in/out}}$: prior instances as input/output.

(a) I-PRM

```
$\mathrm{sample}(\mathscr{D}^{\mathrm{out}}, n)$
$V \leftarrow \mathscr{D} \cap \mathscr{C}_{\mathrm{free}}, E \leftarrow \emptyset$
foreach $v \in V$ do
  $U \leftarrow \mathrm{near}(G^{V,E}, v, \mathscr{D}^{\mathrm{in}})\{v\}$
  foreach $u \in U$ do
    $w \leftarrow \mathrm{icollide}(v, u, \mathscr{D}^{\mathrm{in}})$
    $l \leftarrow \|(v, u)\|$
    $E \leftarrow E \cup (v, u)^{w,l}$
  do
    search path $p$ on $G(V, E)$ which minimizes
      $\sum_e l(e) + \lambda \min_e w(e)$.
    foreach $e \in p$, $\mathrm{collide}(e, \mathscr{D}^{\mathrm{out}})$
  while $p$ not valid
```

(b) I-lazyPRM

```
$V, \mathscr{D} \leftarrow x_{\mathrm{init}}, E \leftarrow \emptyset$
while $x_{\mathrm{goal}}$ not reach
  $x_{\mathrm{rnd}} \leftarrow \mathrm{sample\text{-}free}(\mathscr{D}^{\mathrm{out}}, 1)$
  $x_{\mathrm{nst}} \leftarrow \mathrm{inearst}(G^{V,E}, x_{\mathrm{rnd}}, \mathscr{D}^{\mathrm{in}})$
  $x_{\mathrm{new}} \leftarrow \mathrm{isteer}(x_{\mathrm{nst}}, x_{\mathrm{rnd}}, \mathscr{D}^{\mathrm{in,out}})$
  if $\mathrm{icollide}(x_{\mathrm{nst}}, x_{\mathrm{new}}) < t$
    if $\neg\mathrm{collide}(x_{\mathrm{nst}}, x_{\mathrm{new}})$
      $V \leftarrow V \cup x_{\mathrm{new}}, E \leftarrow E \cup (x_{\mathrm{new}}, x_{\mathrm{nst}})$
```

inearest: find the nearest tree node that is long and has high collision-free probability.
isteer: steer from a tree node to a new node, using icollide for validity checking.
rewire: RRT* routine used to update the tree topology for optimality guarantee.

(c) I-RRT

```
$V, \mathscr{D} \leftarrow x_{\mathrm{init}}, E \leftarrow \emptyset$
while $x_{\mathrm{goal}}$ not reach
  $x_{\mathrm{rnd}} \leftarrow \mathrm{sample\text{-}free}(\mathscr{D}^{\mathrm{out}}, 1)$
  $x_{\mathrm{nst}} \leftarrow \mathrm{inearst}(G^{V,E}, x_{\mathrm{rnd}}, \mathscr{D}^{\mathrm{in}})$
  $x_{\mathrm{new}} \leftarrow \mathrm{isteer}(x_{\mathrm{nst}}, x_{\mathrm{rnd}}, \mathscr{D}^{\mathrm{in,out}})$
  if $\mathrm{icollide}(x_{\mathrm{nst}}, x_{\mathrm{new}}) < t$
    if $\neg\mathrm{collide}(x_{\mathrm{nst}}, x_{\mathrm{new}})$
      $V \leftarrow V \cup x_{\mathrm{new}}$
      $U \leftarrow \mathrm{near}(G^{V,E}, x_{new})$
      foreach $x \in U$, compute weight $c(x) =$
        $\lambda \|(x, x_{\mathrm{new}})\| + \mathrm{icollide}(x, x_{\mathrm{new}}, \mathscr{D}^{\mathrm{in}})$
      sort $U$ according to weight $c$.
      Let $x_{\mathrm{min}}$ be the first $x \in U$ with
$\neg\mathrm{collide}(x, x_{\mathrm{new}})$
      $E \leftarrow E \cup (x_{\mathrm{min}}, x_{\mathrm{new}})$
      foreach $x \in U$, $\mathrm{rewire}(x)$
```

(d) I-RRT*

Fig. 3: Instance-based learning framework can be used to improve different motion planners. Here we present four modified planners.

collision instances, then it has a high probability of being in-collision. Similarly, if a query has many collision-free instances around it, it is likely to be collision-free. In our implementation, we only cull away queries with high collision probability. For queries with high collision-free probability, we still perform exact collision tests on them in order to guarantee that the overall collision detection algorithm is conservative. In Figure 3(a), we show how our probabilistic culling strategy can be integrated with PRM algorithm by only performing exact collision checking (collide) for queries with collision probability (icollide) larger than a given threshold t. Note that the neighborhood search routine (near) can use LSH-based point-point k-NN query.

In Figure 3(b), we show how to use the collision probability as a cost function with the lazyPRM algorithm [14]. In the basic version of lazyPRM algorithm, the expensive local path collision checking is delayed till the search phase. The basic idea is that the algorithm repeatedly searches the roadmap to compute the shortest path between the initial and the goal node, performs collision checking along the edges, and removes the in-collision edges from the roadmap. However, the shortest path usually does not correspond to a collision-free path, especially in complex environments. We improve the lazyPRM planning using instance-based learning. We compute the collision probability for each roadmap edge during roadmap construction, based on Equation 16. The probability (w) as well as the length of the edge (l) are stored as the costs of the edge. During the search step, we try to compute a shortest path with minimum collision probability, i.e., a path that minimizes the cost $\sum_e l(e) + \lambda \min_e w(e)$, where λ is a parameter that controls the relative weightage of path length and collision probability. As the prior knowledge about the obstacles is considered based on collision probability, the resulting path is more likely to be collision-free.

Finally, the collision probability can be used by motion planner to explore $\mathscr{C}_{\text{free}}$ in an efficient manner. We use RRT to illustrate this benefit (Figure 3(c)). Given a random sample x_{rnd}, RRT computes a node x_{nst} among prior collision-free configurations that are closest to x_{rnd} and expands from x_{nst} towards x_{rnd}. If there is no obstacle in $\mathscr{C}$-space, this exploration technique is based on Voronoi heuristic that biases planner in the unexplored regions. However, the existence of obtacles affects its performance: the planner may run into $\mathscr{C}_{\text{obs}}$ shortly after expansion and the resulting exploration is limited. Based on instance-based learning, we can first estimate the collision probability for local paths connecting x_{rnd} with each of its neighbors and choose x_{nst} to be the one with a long edge length, but a small collision probability, i.e., $x_{\text{nst}} = \text{argmax}(l(e) - \lambda \cdot w(e))$, where λ is a parameter used to control the relative weight of these two terms. A similar strategy can also be used for RRT*, as shown in Figure 3(d).

The learning-based planners are faster, mainly because we replace part of the expensive, exact collision queries with relatively cheap k-NN queries. Let the timing cost for a single exact collision query be T_C and for a single k-NN query be T_K, where $T_K < T_C$. Suppose the original planner performs C_1 collision queries and the instance-based learning enhanced planners performs C_2 collision queries and $C_1 - C_2$ k-NN queries, where $C_2 < C_1$. We also assume that the two planners spend the same time A on other computations within a planner, such as sample generation, maintaining the roadmap, etc. Then the speedup ratio obtained by instance-based learning is:

$$R = \frac{T_C \cdot C_1 + A}{T_C \cdot C_2 + T_K \cdot (C_2 - C_1) + A}. \tag{17}$$

Therefore, if $T_C \gg T_K$ and $T_C \cdot C_1 \gg A$, we have $R \approx C_1/C_2$, i.e., if a higher number of exact collision queries are culled, we can obtain a higher speedup. The extreme speedup ratio C_1/C_2 may not be reached, because 1) $T_C \cdot C_1 \gg A$ may not hold, such as when the planner is in narrow passages (A is large) or in open spaces ($T_C \cdot C_1$ is small); 2) $T_C \gg T_K$ may not hold, such as when the environment and robot have low geometric complexity (i.e., T_C is small) or the instance dataset is large and the cost of the resulting k-NN query is high (i.e., T_K is large).

Note that R is only an approximation of the actual acceleration ratio. It may overestimate the speedup because a collision-free local path may have collision probability higher than a given threshold and our method will filter it out. If such a collision-free local path is critical for the connectivity of the roadmap, such false positives due to instance-based learning will cause the resulting planner to perform more exploration and thereby increases the planning time. As a result, we need to choose an appropriate threshold that can provide a balance: we need a large threshold to filter out more collision queries and increase R; at the same time, we need to use a small threshold to reduce the number of false positives. However, the threshold choice is not important in asymptotic sense, because according to Theorem 5, the false positive error converges to 0 when the dataset size increases.

R may also underestimate the actual speedup. The reason is that the timing cost for different collision queries can be different. For configurations near the boundary of $\mathscr{C}_{\text{obs}}$, the collision queries are more expensive. Therefore, the timing cost of checking the collision status for an in-collision local path is usually larger than that of a collision-free local path, because the former always has one configuration on the boundary of $\mathscr{C}_{\text{obs}}$. As a result, it is possible to obtain a speedup larger than C_1/C_2.

Finally, as a natural consequence of Theorem 5, we can prove the completeness and optimality of the new planners:

Theorem 6. *I-PRM, I-lazyPRM, I-RRT are probabilistic complete. I-RRT* is probabilistic complete and asymptotic optimal.*

Proof. A motion planner MP is probabilistic complete if its failure probability, i.e., the probability that it cannot find a solution when a collision-free path exists, converges to 0 when the number of samples N increases: $\lim_{N\to\infty}\mathbb{P}[\text{MP fails}]=0$. Suppose we replace MP's exact collision detection query by the learning-based query and denote the new planner as I-MP. I-MP can fail in two cases: 1) MP fails; 2) MP computes a solution but some edges on the collision-free path are classified as in-collision by our learning algorithm (i.e., false positives). Let L be the number of edges in the solution path and let E_i denote the event that the i-th edge is incorrectly classified as in-collision. As a result, we have

$$\begin{aligned}&\mathbb{P}[\text{I-MP fails}]\\&=\mathbb{P}[\text{MP fails}]+(1-\mathbb{P}[\text{MP fails}])\cdot\mathbb{P}[\bigcup_{i=1}^{L}E_i]\\&\leq\mathbb{P}[\text{MP fails}]+\sum_{i=1}^{L}\mathbb{P}[E_i]\end{aligned}\tag{18}$$

According to Theorem 5, $\lim_{N\to\infty}\mathbb{P}[E_i]=0$ and L is a finite number, then we have $\lim_{N\to\infty}\mathbb{P}[\text{I-MP fails}]=0$, i.e., I-MP is probabilistic complete. Therefore, as PRM, lazyPRM, RRT and RRT* are all probabilistic complete, we can prove that I-PRM, I-lazyPRM, I-RRT and I-RRT* are all probabilistic complete.

Similarly, if MP is asymptotic optimal, then I-MP may not converge to the optimal path only when some of the path edges are classified as in-collision by the learning algorithm and this probability converges to zero. As a result, I-RRT* is asymptotic optimal. □

6 Results

In this section, we highlight the performance of our new planners. Figure 4 and Figure 5 show the articulated PR2 and rigid body benchmarks we used to evaluate the performance. We evaluated each planner on different benchmarks, and for each combination of planner and benchmark we ran 50 instances of the planner and computed the average planning time as an estimate of the planner's performance on this benchmark.

The comparison results are shown in Table 4 and Table 5, corresponding to PR2 benchmarks and rigid body benchmarks, respectively. Based on these benchmarks, we observe:

- The learning-based planners provide more speedup on articulated models. The reason is that exact collision checking on articulated models is more expensive than exact collision checking on rigid models. This makes T_C larger and results in larger speedups.
- The speedup of I-PRM over PRM is relatively large, as exact collision checking takes a significant fraction of overall time within PRM algorithm. I-lazyPRM also provides good speedup as the candidate path nearly collision-free and can greatly reduce the number of exact collision queries in lazy planners. The speedups of I-RRT and I-RRT* are limited or can even be slower than the original planners, especially on simple rigid body benchmarks. The reason is that the original planners are already quite efficient on the simple benchmarks and instance-based learning can result in some overhead.
- On benchmarks with narrow passages, our approach does not increase the probability of finding a solution. However, probabilistic collision checking is useful in culling some of the colliding local paths.

We need to point out that the acceleration results shown in Table 4 and Table 5 show only part of the speedups that can be obtained using learning-based planners. As more collision queries are performed and results stored in the dataset, the resulting planner has more information about $\mathscr{C}_{\text{obs}}$ and $\mathscr{C}_{\text{free}}$, and becomes effective in terms of culling. Ideally, we can filter out all in-collision queries and obtain a high speedup. In practice, we don't achieve ideal speedups due to two reasons: 1) we only have a limited number of samples in the dataset; 2) the overhead of k-NN query increases as the dataset size increases. As a reult, when we perform the global planning query repeatedly, the planning time will decrease to a minimum, and then increases. This is shown in Figure 6. To further improve the performance, we need to adaptively change the LSH parameters to perform k-NN queries efficiently for datasets of varying sizes.

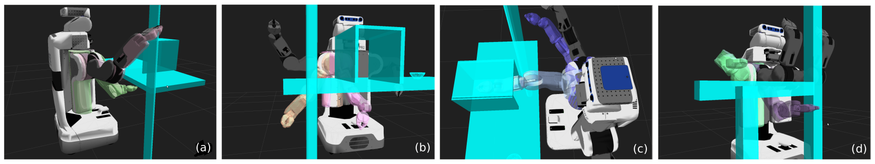

Fig. 4: PR2 planning benchmarks: robot arms with different colors show the initial and goal configurations. The first three benchmarks are of the same environment, but the robot's arm is in different places: (a) moves arm from under desk to above desk; (b) moves arm from under desk to another position under desk and (c) moves arm from inside the box to outside the box. In the final benchmark, the robot tries to move arm from under a shelf to above it. The difficulty order of the four benchmarks is (c) $>$ (d) $>$ (b) $>$ (a).

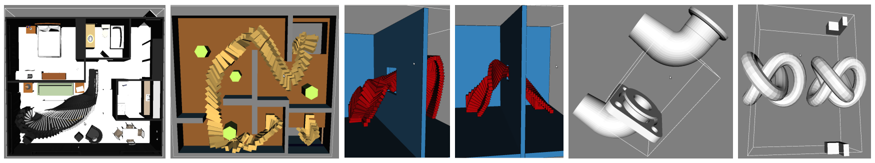

Fig. 5: Rigid body planning benchmarks: from left to right, apartment, cubicles, easy, twistycool, flange and torus. Apartment tries to move the piano to the hallway near the door entrance; cubicles moves the robot in a simple office-like environment and the robot needs to fly through the basement; easy and twistycool are of a similar environment, but twistycool contains a narrow passage. Both flange and torus contain a narrow passage.

	PRM	I-PRM	lazyPRM	I-lazyPRM	RRT	I-RRT	RRT*	I-RRT*
(a)	12.78	9.61 (32%)	1.2	0.87 (37%)	0.96	0.75 (28%)	1.12	1.01 (11%)
(b)	23.7	12.1 (96%)	1.7	0.90 (88%)	1.36	0.89 (52%)	2.08	1.55 (34%)
(c)	fail	fail	fail	fail	4.15	2.77 (40%)	3.45	2.87 (20%)
(d)	18.5	13.6 (36%)	2.52	1.06 (37%)	7.72	5.33 (44%)	7.39	5.42 (36%)

Table 1: Performance comparison on different combinations of planners and PR2 benchmarks (in milliseconds). 'fail' means all the queries cannot find a collision-free path within $1,000$ seconds. The percentage in the brackets shows the speedup obtained using instance-based learning.

7 Conclusion and Future Work

In this paper, we use instance-based learning to improve the performance of sample-based motion planners. The basic idea is to store the prior collision results as an approximate representation of $\mathscr{C}_{\text{obs}}$ and $\mathscr{C}_{\text{free}}$ and replace the expensive exact collision detection query by a relatively cheap probabilistic collision query. We integrate approximate collision routine with various sample-based motion planners and observe $30-100\%$ speedup on rigid and articulated robots.

There are many avenues for future work. First, we need to find methods to adjust LSH parameters adaptively so that the k-NN query becomes more efficient for varying dataset sizes. One possible way is to change L, the number of hash tables,

	PRM	I-PRM	lazyPRM	I-lazyPRM	RRT	I-RRT	RRT*	I-RRT*
apartment	5.25	2.54 (106%)	2.8	1.9 (47%)	0.09	0.10 (-10%)	0.22	0.23 (5%)
cubicles	3.92	2.44 (60%)	1.62	1.37 (19%)	0.89	0.87(2%)	1.95	1.83 (7%)
easy	7.90	5.19 (52%)	3.03	2.01 (50%)	0.13	0.15(-13%)	0.26	0.27 (-4%)
flange	fail	fail	fail	fail	48.47	25.6 (88%)	46.07	26.9 (73%)
torus	31.52	23.3 (39%)	4.16	2.75 (51%)	3.95	2.7 (46%)	6.01	4.23 (42%)
twistycool	1/50	3/50	2/50	3/50	4/50	3/50	2/50	3/50

Table 2: Performance comparison on different combinations of planners and rigid body benchmarks (in milliseconds). The percentage in the brackets shows the speedup based on instance-based learning. For twistycool, which has a narrow passage, we record the number of successful queries among 50 queries (for a 1000 second budget).

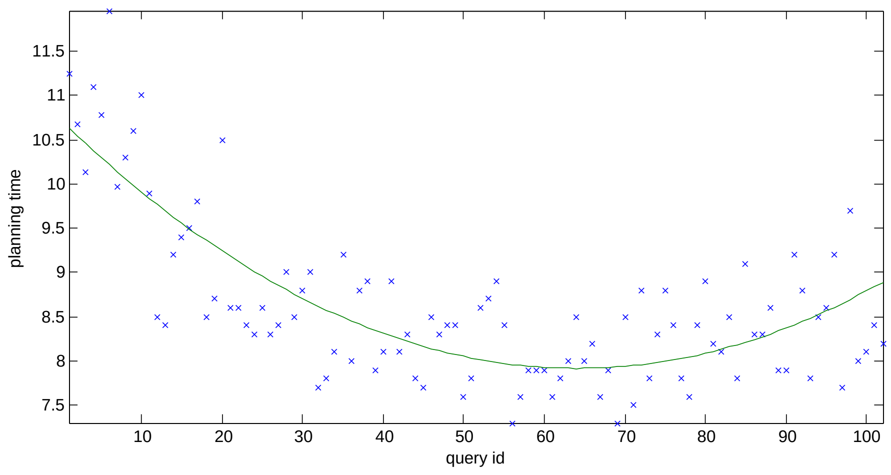

Fig. 6: The time taken by I-PRM when it runs more than 100 times on the benchmark shown in Figure 4 (b). The planning time of a single query first decreases and then increases. The red slash line shows the ideal planning time evolution. The best acceleration acquired is $12.78/7.5 = 70\%$, larger than the 32% in Table 4.

because small L may provide sufficient k-NN candidates for a large dataset. Secondly, for samples in regions that are well explored, we should avoid inserting collision results into the dataset in order to limit the dataset size. Finally, as the prior collision results are stored in hash tables, we can efficiently update the data without high overhead. Therefore, we can extend the instance-based learning framework to improve the performance of planning algorithms in dynamic environments.

Appendix

1 Proof of Theorem 2

Using the result of random projections, for any point $\mathbf{x}$ and any line $l(\mathbf{v},\mathbf{a})$, we have

$$\begin{aligned}&\mathbb{P}[h^{\mathbf{u}}(-\hat{V}(\mathbf{v},\mathbf{a})) = h^{\mathbf{u}}(\hat{V}(\mathbf{x}))]\\ &= 1 - \frac{1}{\pi}\cos^{-1}(\frac{-\hat{V}(\mathbf{v},\mathbf{a})^T\hat{V}(\mathbf{x})}{\|\hat{V}(\mathbf{v},\mathbf{a})\|\|\hat{V}(\mathbf{x})\|}),\end{aligned}$$

where $\mathbf{u} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$. According to our embedding design in Section IV-A1, we have $\hat{V}(\mathbf{v},\mathbf{a})^T\hat{V}(\mathbf{x}) = -\operatorname{dist}^2(\mathbf{x},l)$, $\|\hat{V}(\mathbf{x})\| = c$ and $\|\hat{V}(\mathbf{v},\mathbf{a})\| = \sqrt{d-2+(\operatorname{dist}^2(\mathbf{0},l)+1)^2}$. Let $C = \|\hat{V}(\mathbf{x})\|\|\hat{V}(\mathbf{v},\mathbf{a})\|$, which is a constant independent with database points and only depends on the query, we have $\mathbb{P}[\operatorname{sgn}(\mathbf{u}^T(-\hat{V}(\mathbf{v},\mathbf{a}))) = \operatorname{sgn}(\mathbf{u}^T\hat{V}(\mathbf{x}))] = 1 - \frac{1}{\pi}\cos^{-1}(\frac{-\operatorname{dist}^2(\mathbf{x},l)}{C}) = \frac{1}{\pi}\cos^{-1}(\frac{\operatorname{dist}^2(\mathbf{x},l)}{C})$.

Hence, when $\operatorname{dist}(\mathbf{x},l) \leq r$, we have

$$\mathbb{P}[h^{\mathbf{u}}(-\hat{V}(\mathbf{v},\mathbf{a})) = h^{\mathbf{u}}(\hat{V}(\mathbf{x}))] \geq \frac{1}{\pi}\cos^{-1}(\frac{r^2}{C}) = p_1,$$

and when $\operatorname{dist}(\mathbf{x},l) \geq r(1+\varepsilon)$, we have

$$\mathbb{P}[h^{\mathbf{u}}(-\hat{V}(\mathbf{v},\mathbf{a})) = h^{\mathbf{u}}(\hat{V}(\mathbf{x}))] \leq \frac{1}{\pi}\cos^{-1}(\frac{r^2(1+\varepsilon)^2}{C}) = p_2.$$

Then we can estimate the bound for $\rho = \frac{\log p_1}{\log p_2}$ as follows:

$$\begin{aligned}\rho &= \frac{\log(\frac{1}{\pi}\cos^{-1}(\frac{r^2}{C}))}{\log(\frac{1}{\pi}\cos^{-1}(\frac{r^2(1+\varepsilon)^2}{C}))} \approx \frac{\log(\frac{1}{2}-\frac{r^2}{\pi C})}{\log(\frac{1}{2}-\frac{r^2(1+\varepsilon)^2}{\pi C})}\\ &= \frac{-\log 2+\log(1-\frac{2r^2}{\pi C})}{-\log 2+\log(1-\frac{2r^2(1+\varepsilon)^2}{\pi C})} \approx \frac{-\log 2-\frac{2r^2}{\pi C}}{-\log 2-\frac{2r^2(1+\varepsilon)^2}{\pi C}},\end{aligned}$$

where the first and second approximations are due to the Taylor series of $\cos^{-1}(x)$ and $\log(1-x)$. As a result, we have $\frac{1}{(1+\varepsilon)^2} \leq \rho \leq 1$ and $\rho \approx \frac{1}{(1+\varepsilon)^2}$ if $\frac{r^2}{C}$ is large enough. $\square$

Remark 2. As the computational complexity of LSH is $\mathcal{O}(N^\rho)$ for a given approximation level ε, according to the proof above, the line-point k-NN has sublinear complexity only if $\frac{r^2}{C}$ is large enough. As $C = c\cdot\sqrt{d-2+(\operatorname{dist}^2(\mathbf{0},l)+1)^2}$, this requires $\operatorname{dist}^2(\mathbf{0},l)$ to be small. As a result, the computational complexity is query-dependent, which is caused by the affine property of line. In fact, such undesired property will disappear if we constrain all lines passing through origin, because

$\text{dist}(\mathbf{0},l) = 0$ for all l. To avoid the performance reduction for lines far from the origin, we uniformly sample several points within the space as the origins of different coordinate systems. Next, we perform line-point hashing for dataset points relative to each coordinate system and store them in different hash tables. Given a line query, we also perform similar hashing process. As the line is likely to have small distance to the origin in one of the coordinate systems, the computational complexity can nearly be $\mathscr{O}(N^{\frac{1}{(1+\varepsilon)^2}})$.

2 Proof of Theorem 3

Using the result in [8], for any point $\mathbf{x}$ and any line $l(\mathbf{v},\mathbf{a})$, we have

$$\begin{aligned}&\mathbb{P}[h^{\mathbf{a},b}(-\hat{V}(\mathbf{v},\mathbf{a})) = h^{\mathbf{a},b}(\hat{V}(\mathbf{x}))]\\ &= f(\frac{W}{\|\hat{V}(\mathbf{v},\mathbf{a})+\hat{V}(\mathbf{x})\|}).\end{aligned} \tag{19}$$

According to our embedding design in Section IV-A1, $\|\hat{V}(\mathbf{v},\mathbf{a})+\hat{V}(\mathbf{x})\|^2 = 2\,\text{dist}(\mathbf{x},l) + c^2+d-2+(\text{dist}^2(\mathbf{0},l)+1)^2$ and we let $C = c^2+d-2+(\text{dist}^2(\mathbf{0},l)+1)^2$. Then, if $\text{dist}(\mathbf{x},l) \le r$, there is $\|\hat{V}(\mathbf{v},\mathbf{a})+\hat{V}(\mathbf{x})\| \le \sqrt{2r^2+C}$; if $\text{dist}(\mathbf{x},l) \ge r(1+\varepsilon)$, there is $\|\hat{V}(\mathbf{v},\mathbf{a})+\hat{V}(\mathbf{x})\| \ge \sqrt{2r^2(1+\varepsilon)^2+C}$. Put these bounds into Equation 19, we can obtain p_1 and p_2. Using the result in [8], we have $\frac{1}{1+\varepsilon} \le \rho = \sqrt{\frac{2r^2+C}{2r^2(1+\varepsilon)^2+C}} \le 1$.
$\square$

Remark 3. Similar to the case of hamming hashing in Theorem 2, for p-stable hashing, we also require small $\text{dist}(\mathbf{0},l)$ to obtain sublinear complexity, which can be realized by using multiple coordinate systems with origins uniformly sampled in the problem space.

3 Proof of Theorem 4

The proof is in fact only a simple adaption of the proof of Theorem 1 for point-point k-NN presented in [11].

Let $\mathbf{x}^*$ be a point such that $\text{dist}(\mathbf{x}^*,l) \le r$, then for any j, $\mathbb{P}[g_j(\mathbf{x}^*) = g_j(l)] \ge p_1^M = p_1^{\log_{1/p_2} N} = N^{-\rho}$. Hence, $\mathbb{P}[\forall j, g_j(\mathbf{x}^*) \ne g_j(l)] \le (1-N^{\rho})^L = (1-N^{\rho})^{N^{\rho}} \le 1/e$. Thus, $g_j(\mathbf{x}^*) = g_j(l)$ holds for some $1 \le j \le L$ with probability at least $1-1/e$. We denote this property as P_1.

Next, consider the set of points whose distance from l is at least $r(1+\varepsilon)$ and have the same hash code with query l under hash function g_j. We denote the set as $S_j = \{\mathbf{x} \mid \text{dist}(\mathbf{x},l) > r(1+\varepsilon) \text{ and } g_j(\mathbf{x}) = g_j(l)\}$ and let $S = \cup S_j$. Note that the probability for one point in the dataset belonging to S_j is $\mathbb{P}[\text{dist}(\mathbf{x},l) >$

$r(1+\varepsilon)$ and $g_j(\mathbf{x}) = g_j(l)] = \mathbb{P}[\text{dist}(\mathbf{x}, l) > r(1+\varepsilon)]\mathbb{P}[g_j(\mathbf{x}) = g_j(l) \mid \text{dist}(\mathbf{x}, l) > r(1+\varepsilon)] \leq 1 \cdot p_2^M = p_2^{\log_{1/p_2} N} = 1/N$. As a result $\mathbb{E}(|S_j|) \leq 1/N \cdot N = 1$ and $\mathbb{E}(|S|) \leq \sum_{j=1}^{L} \mathbb{E}(|S_j|) \leq L$. By Markov's inequality, $\mathbb{P}[|S| > cL] \leq \frac{\mathbb{E}(|S|)}{cL} \leq \frac{1}{c}$. As a result $|S| < 2l$ with probability at least $1/2$. As S is the data items that the algorithm needs to search but are not valid k-NN results, this property means the search time is bounded by $|S| = \mathscr{O}(L = N^\rho)$. We denote this property as P_2.

Combine the above two properties, the probability that the algorithm can find at least one point $\mathbf{x}^*$ with $\text{dist}(\mathbf{x}^*, l) \leq r$ in bounded retrieval time $\mathscr{O}(N^\rho)$ (i.e., $|S| = 2L$) is $1-((1-\mathbb{P}[P_1]) + (1-\mathbb{P}[P_2])) \geq 1/2 - 1/e$. $\square$

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of the ACM **51**(1), 117–122 (2008)
2. Andoni, A., Indyk, P., Krauthgamer, R., Nguyen, H.L.: Approximate line nearest neighbor in high dimensions. In: Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 293–301 (2009)
3. Basri, R., Hassner, T., Zelnik-Manor, L.: Approximate nearest subspace search. Pattern Analysis and Machine Intelligence, IEEE Transactions on **33**(2), 266–278 (2011)
4. Boor, V., Overmars, M., van der Stappen, A.: The gaussian sampling strategy for probabilistic roadmap planners. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 1018–1023 (1999)
5. Burns, B., Brock, O.: Toward optimal configuration space sampling. In: Proceedings of Robotics: Science and Systems (2005)
6. Chakrabarti, A., Regev, O.: An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In: Proceedings of IEEE Symposium on Foundations of Computer Science, pp. 473–482 (2004)
7. Dalibard, S., Laumond, J.P.: Linear dimensionality reduction in random motion planning. International Journal of Robotics Research **30**(12), 1461–1476 (2011)
8. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of Symposium on Computational Geometry, pp. 253–262 (2004)
9. Denny, J., Amato, N.M.: The toggle local planner for probabilistic motion planning. In: Proceedings of IEEE International Conference on Robotics and Automation (2012)
10. Diankov, R., Ratliff, N., Ferguson, D., Srinivasa, S., Kuffner, J.: Bispace planning: Concurrent multi-space exploration. In: Proceedings of Robotics: Science and Systems (2008)
11. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of International Conference on Very Large Data Bases, pp. 518–529 (1999)
12. Hsu, D., Sanchez-Ante, G., Sun, Z.: Hybrid PRM sampling with a cost-sensitive adaptive strategy. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 3874–3880 (2005)
13. Jain, P., Vijayanarasimhan, S., Grauman, K.: Hashing hyperplane queries to near points with applications to large-scale active learning. In: Neural Information Processing Systems (2010)
14. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation **12**(4), 566–580 (1996)
15. Kuffner, J., LaValle, S.: RRT-connect: An efficient approach to single-query path planning. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 995–1001 (2000)

16. Li, P., Hastie, T.J., Church, K.W.: Very sparse random projections. In: Proceedings of International Conference on Knowledge Discovery and Data Mining, pp. 287–296 (2006)
17. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. Combinatorica **15**(2), 215–245 (1995)
18. Morales, M., Tapia, L., Pearce, R., Rodriguez, S., Amato, N.M.: A machine learning approach for feature-sensitive motion planning. In: Proceedings of International Workshop on Algorithmic Foundation of Robotics, pp. 361–376 (2004)
19. Rodriguez, S., Tang, X., Lien, J.M., Amato, N.: An obstacle-based rapidly-exploring random tree. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 895–900 (2006)
20. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall (2003)
21. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc. (2005)
22. Scholz, J., Stilman, M.: Combining motion planning and optimization for flexible robot manipulation. In: Proceedings of IEEE-RAS International Conference on Humanoid Robots, pp. 80–85 (2010)
23. Stoyan, D., Kendall, W.S., Mecke, J.: Stochastic Geometry and Its Applications. John Wiley & Sons (1996)
24. Sun, Z., Hsu, D., Jiang, T., Kurniawati, H., Reif, J.H.: Narrow passage sampling for probabilistic roadmap planners. IEEE Transactions on Robotics **21**(6), 1105–1115 (2005)