Efficient Nearest Neighbor Searching for Motion Planning

Anna Atramentov
Dept. of Computer Science
Iowa State University
Ames, IA 50011 USA
anjuta@cs.iastate.edu

Steven M. LaValle
Dept. of Computer Science
University of Illinois
Urbana, IL 61801 USA
lavalle@cs.uiuc.edu

Abstract

We present and implement an efficient algorithm for performing nearest-neighbor queries in topological spaces that usually arise in the context of motion planning. Our approach extends the Kd tree-based ANN algorithm, which was developed by Arya and Mount for Euclidean spaces. We argue the correctness of the algorithm and illustrate its efficiency through computed examples. We have applied the algorithm to both probabilistic roadmaps (PRMs) and Rapidly-exploring Random Trees (RRTs). Substantial performance improvements are shown for motion planning examples.

1 Introduction

Nearest neighbor searching is a fundamental problem in many applications, such as pattern recognition, statistics, and machine learning. It is also an important component in several path planning algorithms. Probabilistic roadmap (PRM) approaches [1, 10], build a graph of collision-free paths that attempts to capture the connectivity of the configuration space. The vertices represent configurations that are generated using random sampling, and attempts are made to connect each vertex to nearby vertices. Some roadmaps contain thousands of vertices, which can lead to substantial computation time for determining nearby vertices in some applications. Approaches based on Rapidly-exploring Random Trees (RRTs) [12, 13, 14] rely even more heavily on nearest neighbors. An RRT is a tree of paths that is grown incrementally. In each iteration, a random configuration is chosen, and the RRT vertex that is closest (with respect to a predefined metric) is selected for expansion. An attempt is made to connect the RRT vertex to the randomly-chosen state.

An approach that can efficiently find nearest neighbors can dramatically improve the performance of these path planners. Although packages exist, such as ANN ([16], U. of Maryland) and Ranger (SUNY Stony Brook), they are designed for the common case of generating nearest neighbors in $\mathbb{R}^n$. This cannot be applied to path planning algorithms because of the complicated topologies of configuration spaces. The topologies that usually arise are Cartesian products of $\mathbb{R}$, S^1, and projective spaces. In these cases, metric information must be appropriately processed by any data structure designed to perform nearest neighbor computations. The problem is further complicated by the high dimensionality of configuration spaces.

In this paper, we extend the nearest neighbor algorithm and part of the ANN package of Arya and Mount [16] to handle general topologies with very little computational overhead. Related literature is discussed in Section 3. Our approach is based on hierarchically searching nodes in a Kd tree while handling the appropriate constraints induced by the metric and topology. We have proven the correctness of our approach, and we have demonstrated the efficiency of the algorithm experimentally. We first present experiments that demonstrate the performance improvement over using linear-time naive nearest neighbor computations. The improvement is a few orders of magnitude in some cases. We also present experiments that show substantial performance improvement in the PRM and RRT applied to difficult path planning examples.

2 Problem Formulation

The configuration space, $\mathcal{C}$, which arises in motion planning problems, is usually a non-Euclidean manifold. In the case of a 2D rigid body in the plane, the $\mathcal{C} = \mathbb{R}^2 \times S^1$, in which S^1 represents a circle. In the case of a 3D rigid body in space, $\mathcal{C} = \mathbb{R}^3 \times P^3$, in which P^3 denotes a three-dimensional projective space. In the case of multiple bodies, the resulting configuration space is obtained by taking Cartesian products of copies of $\mathbb{R}$, S^1, and P^3.

These desired n-dimensional configuration spaces, and many others, can be represented by defining a rectangular subset of $\mathbb{R}^n$, and identifying appropriate pairs of boundary points to obtain the desired topology. For example, several two-dimensional manifolds can be obtained by identifying points on the unit square in the plane, as shown in Figure 1. Arrows on a pair of opposite edges indicate identification of the opposite points on these edges.

If arrows are drawn in opposite directions, then there is a twist in the identification.

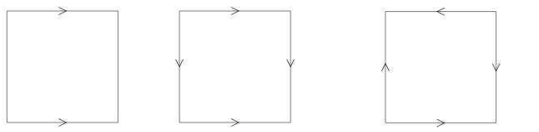


Figure 1: Some 2D manifolds obtained by identifications of the rectangle boundary points.

The manifolds of interest in this paper are constructed as the Cartesian products of any of:

- Euclidean one-space, represented by $(0, 1) \subset \mathbb{R}$.
- Circle, represented by $[0, 1]$, in which $0 \sim 1$ by identification.
- P^3, represented by a three-dimensional unit cube with antipodal points identified.

To define a nearest neighbor problem, it will be essential to define a metric on these manifolds.

Definition 2.1 *The distance between two points $p, q \in \mathbb{R}^1$ is defined as follows:*

$$dist_{\mathbb{R}}(q,p) = |q-p|.$$

Definition 2.2 *The distance between two points $p, q \in S^1$ is defined as follows:*

$$dist_{S^1}(q,p) = \min(|q-p|, 1-|q-p|).$$

Definition 2.3 *The distance between two points $q, p \in P^3$, in which $q = (q_1, q_2, q_3)$ and $p = (p_1, p_2, p_3)$, is defined as follows:*

$$\begin{aligned}
&dist_{P^3}(q,p) = \min_{i=1..8}(d_i).\\
&d_1 = dist_{\mathbb{R}^3}(q,p);\\
&d_2 = \sqrt{\left(\sum_{j=1}^{3}(1-|q_j-p_j|)\right)^2};\\
&d_{3,4} = dist_{\mathbb{R}^3}((-q_1,-q_2,q_3 \pm 1),p);\\
&d_{5,6} = dist_{\mathbb{R}^3}((-q_1,q_2 \pm 1,-q_3),p);\\
&d_{7,8} = dist_{\mathbb{R}^3}((q_1 \pm 1,-q_2,-q_3),p).
\end{aligned}$$

Definition 2.4 *For manifolds, T_1 and T_2, if $q, p \in T_1 \times T_2$ then the distance between these two points is defined as follows:*

$$dist_{T_1 \times T_2}(q,p) = \sqrt{dist^2_{T_1}(q,p) + dist^2_{T_2}(q,p)}.$$

Other metric definitions could be used in our approach; however, we preclude their consideration in this presentation.

The problem is: given a d-dimensional manifold, T, and a set of data points in T, preprocess these points so that, for any query point $q \in T$, the nearest data point to q can be found quickly.

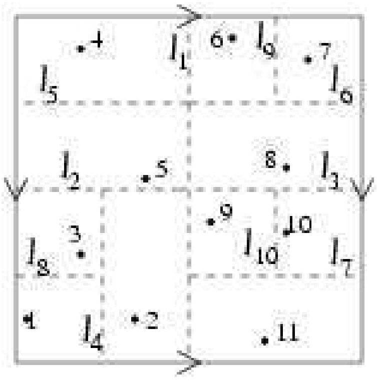
3 Algorithm Presentation

To solve the problem we use existing techniques for searching for the nearest neighbors in Euclidean spaces. We make modifications to these techniques to handle topology of the space. First we precompute the data structure for storing points, and then search this data structure when a query is given.

Kd trees There has been a significant interest in nearest neighbors and related problems over the last couple of decades. The Kd tree is a powerful data structure that based on recursively subdividing a set of points based on alternating axis-aligned hyperplanes. The classical Kd tree uses $O(d \lg n)$ precomputation time, and answers orthogonal range queries in $O(n^{1-\frac{1}{d}})$. One of the first appearances of the Kd tree is in [8]. A more recent introduction appears in [6]. Improvements to the data structure and its construction algorithm in the context of nearest neighbor searching are described in [18]. In [4] it is shown that using Kd trees for finding approximate nearest neighbors allows significant improvement in running time with a very small loss in performance for higher dimensions.

Suppose that the data points lie in an d-dimensional box, which has some points identified on its boundary. We build the data structure inside this box, and define it recursively as follows. The set of data points is split into two parts by splitting the box containing them into two child boxes by a plane, according to some splitting rule specified by the algorithm; one subset contains the points in one child box, and another subset contains the rest of the points. The information about the splitting plane and the boundary values of the initial box are stored in the root node, and the two subsets are stored recursively in the two subtrees. When the number of the data points contained in some box falls below a given threshold, then we call a node associated with this box a leaf node, and we store a list of coordinates for these data points in this node.

We use dividing rules suggested by [16], which divides the current cell through its midpoint orthogonal to its longest side. If there are ties, it selects the dimension with longest point spread. However, in the case in which points are all on one side of the splitting plane, then the algorithm slides the plane toward the first encountered



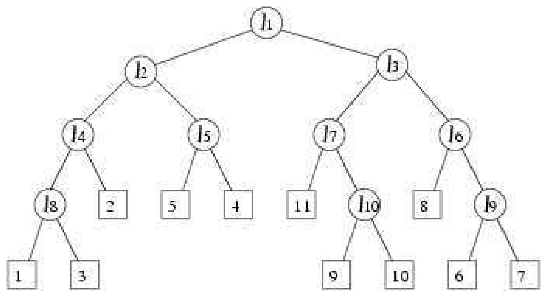


Figure 2: A Kd tree: a) how a torus is subdivided, b) the corresponding binary tree

data point. According to [16] these rules perform very well with typical data sets.

Figure 2 illustrates how the splitting is done and how the corresponding binary tree looks for the data points on a torus.

Construction phase A Kd tree is constructed using the recursive procedure, which returns the root of the Kd tree. This construction is essentially identical to the case of constructing a Kd tree in $\mathbb{R}^d$ [16]. The identifications are ignored.

Query phase The query phase must be handled differently in comparison to a standard Kd tree. The topology must be carefully considered when traversing the tree. When a query is made, a point, $q \in T$, is given, and the closest point to q is found. At first the query algorithm descends to a leaf node which contains the query point, finds all distances from the data points in this leaf to the query point, and picks up the closest one. Then, it recursively visits those surrounding leaf nodes which are further from the query point than the closest point found so far. Figure 3 illustrates the query algorithm.

We borrowed some efficient techniques from [3] to further improve the computations. Using squared distances prevents calculating costly square roots. We also modified method of *incremental distance calculation* for speeding up the calculations of a distance between the query point and rectangle. This method can be described as follows. Let the manifold T be a Cartesian product of some manifolds $T = T_1 \times T_2 \times ... \times T_m$ and some coordinate axis k corresponds to T_K. Suppose a query point, q, and a rectangle, $R \subseteq T$ are given. Divide R with a plane orthogonal to coordinate axis k into two child rectangles R_1 and R_2. If we know that $dist^2(q, R) = d$, then the squared distance from one of the rectangles (let it be R_1) to q will be d as well. To calculate $dist^2(q, R_2)$, note that R_2 has the same projections as R on every T_i except for T_K. Therefore, if $dist^2_{T_K}(q, R_1) = dist^2_{T_K}(q, R)$, then

$$dist^2_T(q, R_2) = d - dist^2_{T_K}(q, R_1) + dist^2_{T_K}(q, R_2).$$

Therefore, calculating distances from point to rectangle nodes in d-dimensional space, T, will take $O(d)$ time only for the root node, for any other node the time will be proportional to calculating distance from point to rectangle

```
KDTree::SEARCH($q$)
*Output* the closest to $q$ point $p$ stored in Kd tree
1   Calculate squared distance $d$ from the box
    associated with the *root* node to $q$
2   $p = NULL$
3   *root*$-$ >SEARCH($d$, $\infty$, $p$)
4   **return** $p$

Node::SEARCH($d$, &$d_{best}$, &$p$)
*Input* squared distances $d$, from $q$ to the box containing
current Node and $d_{best}$, from $q$ to the closest to it point
$p$ seen so far, $d_{best}$ and $p$ are to be updated
1   **if** $d < d_{best}$
2       Split $b_K$ (the projection of the current
        node onto the topological space $T_K$,
        stored in this node) into two subboxes,
        $b_{K_1}$ and $b_{K_2}$, by the splitting line $l$,
        corresponding to $v_1$ and $v_2$ respectively.
3       $d_1 = dist^2_{T_K}(q, b_{K_1})$
4       $d_2 = dist^2_{T_K}(q, b_{K_2})$
5       **if** $d_1 < d_2$
6           **then** $v_1-$ >SEARCH($d$, $d_{best}$, $p$)
7               $v_2-$ >SEARCH($d - d_1 + d_2$, $d_{best}$, $p$)
8           **else** $v_2-$ >SEARCH($d$, $d_{best}$, $p$)
9               $v_1-$ >SEARCH($d - d_2 + d_1$, $d_{best}$, $p$)

Leaf::SEARCH($d$, &$d_{best}$, &$p$)
*Input* squared distances $d$, from $q$ to the box containing
current Leaf and $d_{best}$, from $q$ to the closest to it point $p$
seen so far, $d_{best}$ and $p$ are to be updated
1   Calculate squared distances from $q$ to all points
    in a leaf node, and update $p$ and $d_{best}$ if needed
```

Figure 3: The algorithm portions for searching KD tree on the root level and internal and leaf nodes levels

in T_K, the subspace of T.

Incremental computations In some algorithms, such as the RRT, the number of points grows incrementally while nearest-neighbor queries are performed at each iteration. In this case, it is inefficient to rebuild the Kd tree at every iteration. One approach to incrementalize the algorithm is to use the point insertion operation with tree re-balancing [17]. It is costly, however, to ensure that the trees are balanced. Another approach, which we used in our implementation, is to construct a vector of trees. For n points, there is a tree that contains 2^i points for each "1" in the i^{th} place of the binary representation of n. As bits are cleared in the representation due to increasing n, the trees are deleted, and the points are included in a tree that corresponds to the higher-order bit which changed to "1". This general scheme incurs logarithmic-time overhead, regardless of dimension. It is also straightforward to implement, and leads to satisfactory experimental performance.

4 Analysis

Proposition 1 *The algorithm presented in Section 3 correctly returns the nearest neighbor.*

Proof: We argue that the points of our Kd tree not visited by an algorithm will always be further from the query point then some point already visited. At first the algorithm descends to the leaf node to which a query point belongs. Therefore, the closest point to the query point from this leaf will be the first candidate to be the nearest neighbor. After searching this leaf node the algorithm will skip only those nodes that are further from the query point then the candidate to the nearest neighbor seen so far. Any point inside a node that was skipped cannot be the nearest neighbor, since the point inside rectangle is further from the query point than the rectangle itself by the definition of the Hausdorff distance. ∎

Proposition 2 *For n points in dimension d, the construction time is $O(d \lg n)$, the space is $O(dn)$, and the query time is logarithmic in n, but exponential in d.*

Proof: This follows directly from the well-known complexity of the basic Kd tree [8]. Our approach performs correct handling of the topology without any additional asymptotic complexity. The metric evaluations are more costly due to identifications in the manifold definition; however, this results only in a larger constant in the asymptotic analysis. ∎

By following several performance enhancements recommended in [16], the effects of high dimensionality on the query time are minimized, yielding good performance for nearest neighbor searching in for several dozen dimensions in both ANN and our algorithm. Performance can be further improved by returning approximate nearest neighbors, if suitable for motion planning algorithms.

dim	topology	new alg.	naive
3	$S^1 \times S^1 \times S^1$	5.6s+0.1s	29.2s
6	$R^3 \times P^3$	7.1s+3.2s	68.9s
13	$R^3 \times (S^1)^4 \times (P^3)^2$	8.8s+9.2s	153.1s

Figure 4: Nearest neighbor computations are evaluated in isolation for several manifolds. For each space, 50000 data points were generated at random. For the new algorithm, both the Kd tree construction time and the time required to perform 100 queries are shown. For the naive, brute-force algorithm, the time to perform 100 queries is shown.

5 Experiments

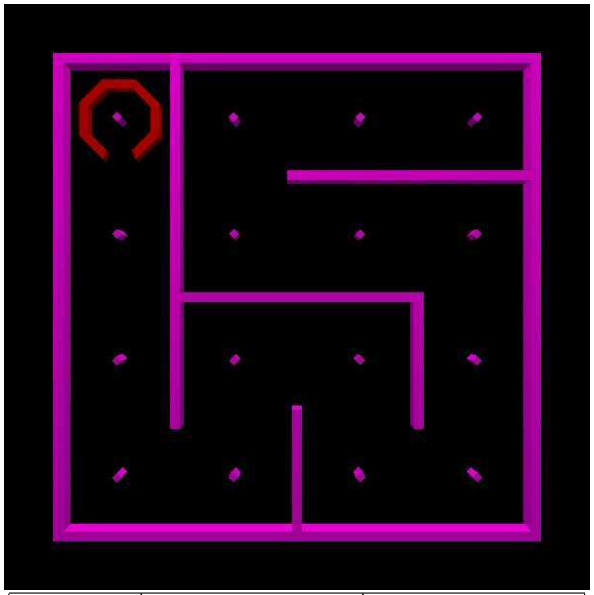
We have implemented our nearest neighbor algorithm in C++ by directly building upon the ANN library. The nearest neighbor algorithm was then applied to implementations of RRT-based and PRM-based planners in the Motion Strategy Library. The experiments reported here were performed on a 500 Mhz Pentium III running Linux and compiled under GNU C++.

Figure 4 indicates substantial performance improvements in the time to perform 100 queries in various topological spaces. Figures 5 and 6 show performance improvements in bidirectional RRT-based planners for 3-dof and 48-dof problems, respectively. Finally, performance improvement is shown for a basic PRM applied to a 6-dof example in Figure 7.

Based on the experiments, we have observed improvements of up to an order of magnitude in the performance of RRT-based and PRM-based planning algorithms. It is important to note, however, that nearest neighbor searching does not represent the only bottleneck in motion planning. Sampling strategies and collision detection issues are also critical. For the experiments, we focused on examples that lead to a large number of nodes so that the nearest-neighbor searching would dominate. In general, the development of the most efficient algorithms should involve consideration of all of these issues.

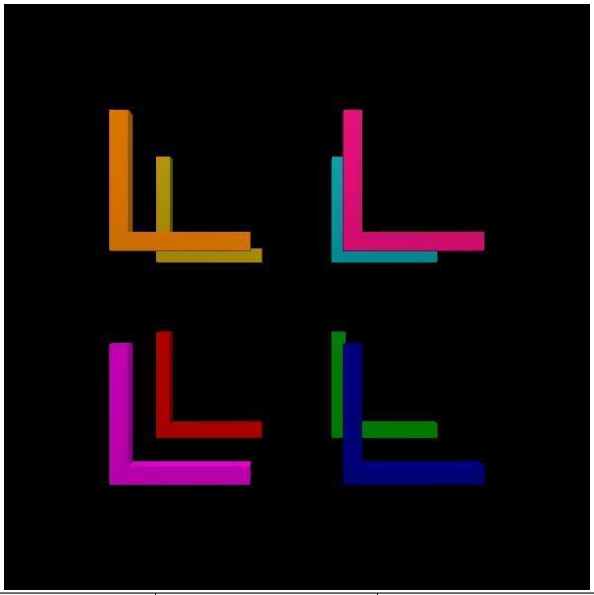
6 Conclusions

We have presented and implemented a practical algorithm for performing efficient nearest neighbor searches for the topological spaces that commonly arise in motion planning. We have illustrated the importance of performing efficient nearest neighbor computations in the context



topology	new algorithm		brute-force	
	nodes	time (s)	nodes	time (s)
$R^2 \times S^1$	37177	584.9	39610	7501.4

Figure 5: This example involves moving the C-shaped object to the end of the maze. There are many narrow corridors in the configuration space, $R^2 \times S^1$. The problem was solved using a bidirectional RRT (RRTConCon, described in [15]).



topology	new algorithm		brute-force	
	nodes	time (s)	nodes	time (s)
$(R^3 \times P^3)^8$	17271	4631.9	18029	8461.6

Figure 6: A 48-dimensional problem that involves exchanging positions of 8 L-shaped objects contained in a rectangular box. It was solved using a bidirectional RRT (RRTConCon, described in [15]).

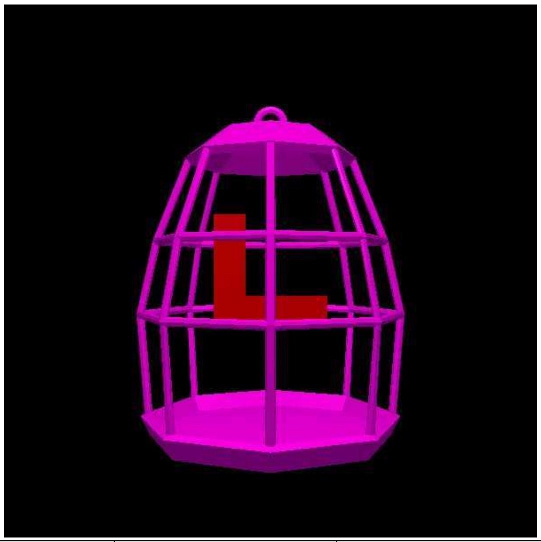
of path planning. Our method extends previous techniques designed for Euclidean spaces by building Kd trees that respect topological identifications and the resulting distance metric.

Our method has been implemented, and is observed to be orders of magnitude faster than naive nearest neighbor searching. It is substantially faster for high-dimensional spaces, which are of great importance in motion planning. We evaluated the implemented algorithm as a means to accelerate performance in both PRM and RRT algorithms. Substantial improvement was observed in both cases; however, it is important to note that the benefits are substantial only if the nearest neighbor computations dominate the total running time. Collision detection is a competing bottleneck in path planning algorithms; therefore, strong performance benefits can be expected in cases in which the number of PRM or RRT nodes is large in comparison to the number of primitives in the geometric models used for collision detection.

Even though the set of 3D rotations forms a projective space, there are many ways to parameterize it and define a metric. In this paper, we make a single choice, but many others are possible. The effect of metric choices in the PRM context are discussed in [2]. It is important to note, however, that our method can be used for other representations, such as yaw-pitch-roll.

Several directions are possible for future work. The extension to different topological spaces can also be applied to other extensions of the Kd tree that have been used for nearest neighbor searching, such as the relative neighbor graph [3] and balanced box-decomposition tree [5]. It has been shown recently that it is possible to remove exponential dependencies in dimension from the nearest neighbor problem [9, 11]. Powerful new techniques are based on approximate distance-preserving embeddings of the points into lower-dimensional spaces. It remains to be seen whether these theoretical ideas will lead to practical algorithms, and whether they will yield superior performance for the dimension and number of points that are common in path planning problems. In path planning problems that involve differential constraints (nonholonomic and kinodynamic planning), it might be preferable to use complicated distance functions [7]. Such functions should be well-suited for a particular nonlinear system, and they might not even be symmetric. In these difficult cases, it remains to determine practical, efficient nearest neighbor algorithms.

Acknowledgments Funding was provided in part by NSF CAREER Award IRI-9875304 (LaValle).



topology	new algorithm		brute-force	
	nodes	time (s)	nodes	time (s)
$R^3 \times P^3$	3837	21.81	3889	31.63
	11684	96.83	11698	247.29
	19292	124.38	19545	506.08
	27076	147.58	27259	740.42
Success	34835	176.92	35063	999.96
	42630	173.43	42714	1229.70
	50392	213.67	50514	1507.15
	58117	215.16	58308	1796.17
	65858	281.66	66156	2020.86
	73613	224.73	73978	2271.60

Figure 7: This example is solved using the PRM approach. The goal is to move the object out of the cage.

References

[1] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *IEEE Int. Conf. Robot. & Autom.*, pages 113–120, 1996.

[2] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *IEEE Int. Conf. Robot. & Autom.*, pages 630–637, 1998.

[3] S. Arya and D. M. Mount. Algorithm for fast vector quantization. In *IEEE Data Compression Conference*, pages 381–390, March 1993.

[4] S. Arya and D. M. Mount. Approximate nearest neihgbor queries in fixed dimensions. In *ACM-SIAM Sympos. Discrete Algorithms*, pages 271–280, 1993.

[5] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions (revised version). In *Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, January 1994.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer, Berlin, 1997.

[7] E. Frazzoli, M. A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicles motion planning. Technical Report LIDS-P-2468, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1999.

[8] J. H. Friedman, J. L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[9] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[10] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. & Autom.*, 12(4):566–580, June 1996.

[11] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *ACM Symposium on Theory of Computing*, pages 599–608, May 1997.

[12] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 995–1001, 2000.

[13] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University, Oct. 1998.

[14] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 473–479, 1999.

[15] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics*, 2000.

[16] D. M. Mount. ANN programming manual. Technical report, Dept. of Computer Science, U. of Maryland, 1998.

[17] M. H. Overmars and J. van Leeuwen. Dynamic multidimensional data structures based on Quad- and K-D trees. *Acta Informatica*, 17:267–285, 1982.

[18] R. L Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.