On the Efficiency of Nearest Neighbor Searching with Data Clustered in Lower Dimensions

Songrit Maneewongvatana and David M. Mount
{songrit,mount}@cs.umd.edu

Department of Computer Science
University of Maryland
College Park, Maryland

1 Introduction

Nearest neighbor searching is an important and fundamental problem in the field of geometric data structures. Given a set S of n *data points* in real d-dimensional space, $\mathbf{R}^d$, we wish to preprocess these points so that, given any *query point* $q \in \mathbf{R}^d$, the data point nearest to q can be reported quickly. We assume that distances are measured using any Minkowski distance metric, including the Euclidean, Manhattan, and max metrics. Nearest neighbor searching has numerous applications in diverse areas of science.

In spite a recent theoretical progress on this problem, the most popular linear-space data structures for nearest neighbor searching are those based on hierarchical decompositions of space. Although these algorithms do not achieve the best asymptotic performance, they are easy to implement, and can achieve fairly good performance in moderately high dimensions. Friedman, Bentley, and Finkel [FBF77] showed that kd-trees achieve $O(\log n)$ expected-case search time and $O(n)$ space, for fixed d, assuming data distributions of bounded density. Arya, et al. [AMN+98] showed that $(1+\epsilon)$ approximate nearest neighbor queries can be answered $O((d/\epsilon)^d \log n)$ time, assuming $O(dn)$ storage. There have been many approaches to reduce the exponential dependence on d [IM98,Kle97].

The unpleasant exponential factors of d in the worst-case analyses of some data structures would lead one to believe that they would be unacceptably slow, even for moderate dimensional nearest neighbor searching ($d < 20$). Nonetheless, practical experience shows that, if carefully implemented, they can applied successfully to problems in these and higher dimensions [AMN+98].

The purpose of this paper is to attempt to provide some theoretical explanation for a possible source for this unexpectedly good performance, and to comment on the limitations of this performance. Conventional wisdom holds that because of dependencies between the dimensions, high dimensional data sets often consist of many low-dimensional clusters. A great deal of work in multivariate data analysis deals with the problems of dimension reduction and determining the *intrinsic dimensionality* of a data set [CP96]. For example, this may be done through the use of techniques such as the Karhunen-Loeve transform [Fuk90].

This suggests the question of how well do data structures take advantage of the presence of low-dimensional clustering in the data set to improve the search?

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2073, pp. 842–851, 2001.

Traditional worst-case analysis does not model the behavior of data structures in the presence of simplifying structure in the data. In fact, it focuses on worst-case situations, which may be rare in practice. Even expected-case analyses based on the assumption of uniformly distributed data [FBF77,Cle79] are not dealing with "easy" instances since the curse of dimensionality is felt in its full force.

We consider the following very simple scenario. Assuming that the data points and query points are sampled uniformly from a k-dimensional hyperplane (or k-flat), where $k < d$, what is the expected-case search time for kd-trees as a function of n, k and d? In [FBF77] it is shown that when $k = d$ and if boundary effects (explained in [AMN96]) are ignored, the expected number of leaf cells in the tree to be visited is at most $(G(d)^{1/d} + 1)^d$, where $G(d)$ is the ratio of the volumes of a d-dimensional hypercube and a maximal enclosed ball for the metric inside the hypercube. These results rely on the fact that when data points are uniformly distributed, the cells of the kd-tree can be approximated by d-dimensional hypercubes. However this is not the case when data points lie on a lower dimensional hyperplane.

It is natural to conjecture that if $k \ll d$, then search times grow exponentially in k but not in d. Indeed, we show that this is the case, for a suitable variant of the kd-tree. We introduce a new splitting method, called the *canonical sliding-midpoint splitting method*. This is a variant of a simpler splitting method called *sliding-midpoint*, which is implemented in the ANN approximate nearest neighbor library [MA97]. (Definitions are given in the next section.)

Our main result is that canonical sliding-midpoint kd-trees can achieve query times depending exponentially on the intrinsic dimension of data, and not on the dimension of the space. We show that if the data points are uniformly distributed on a k-flat, then the expected number of leaf cells that intersect a nearest neighbor ball is $O(d^{k+2})$. Further, we show that if the points are clustered along a k-flat that is aligned with the coordinate axes, even better performance is possible. The expected number of leaf cells intersecting the nearest neighbor ball decreases to $O((d-k+1)c^k)$, where c is the quantity $(G(k)^{1/k}+1)$. The restrictions of using the canonical sliding-midpoint splitting method and having points lie on a flat do not seem to be easy to eliminate. It is not hard to show that if points are perturbed away from the flat, or if some other splitting method is used, there exist point configurations for which 2^d cells will be visited.

The problem of how hierarchical decomposition methods perform when given data with low intrinsic dimensionality has been studied before. Faloutsos and Kamel [FK94] have shown that under certain assumptions, the query time of range queries in an R-tree depends on the fractal dimension of the data set. Their results do not apply to nearest neighbor queries, because their analysis holds in the limit for a fixed query range as the data size tends to infinity.

We also present empirical results that support our results. Furthermore, we consider its robustness to violations in our assumptions. We consider the cases where there is more than just a single cluster of points, but a number of clusters of points lying on different hyperplanes, and where the points do not lie exactly on the hyperplane, but are subject to small perturbations. These empirical results

bear out the fact that the query times are much more strongly dependent on k than on d.

2 Background

First we recall the basic facts about kd-trees [Ben75]. Consider a set S of n data points in $\mathbf{R}^d$. A kd-tree is a binary tree that represents a hierarchical subdivision of space, using splitting planes that are orthogonal to the coordinate axes. Each node of the kd-tree is associated with a closed rectangular region, called a *cell*. The root's cell is associated with a bounding hypercube that contains all the points of S. Information about splitting dimension and splitting value is associated with each cell. These define an axis-orthogonal *splitting hyperplane*. The points of the cell are partitioned to one side or the other of this hyperplane. The resulting subcells are the children of the original cell. This process continues until the number of points is at most one. There are a number of ways of selecting the splitting hyperplane, which we outline below.

Standard split: Proposed in [FBF77], it selected the splitting dimension to be the one for which point set has the maximum *spread* (difference between the maximum and minimum values). The splitting value is chosen to be the median in that dimension. This method is well-known and widely used.

Midpoint split: The splitting hyperplane passes through the center of the cell and bisects the longest side of the cell. If there are many sides of equal length, any may be chosen first, say, the one with the lowest coordinate index. This is just a binary version of the well-known quadtree and octree decompositions.

Observe that the standard splitting rule produces balanced kd-trees with $O(\log n)$ depth. The midpoint tree has the feature that for all cells, the ratio of the longest to shortest side (the *aspect ratio*) is at most 2. (We will sometimes use the term *box* to mean a cell of bounded aspect ratio.) This is not necessarily true for the standard splitting method. As shown in [AMN+98], bounded aspect ratio is important to the efficiency of approximate nearest neighbor searching. Unfortunately, if the data are clustered, it is possible to have many empty cells that contain no data points. This is not uncommon in practice, and may result in trees that have many more than $O(n)$ nodes.

Note that the set of possible splitting planes in midpoint split is determined by the position of the initial bounding hypercube. For example, suppose that the initial bounding box is affinely mapped to a unit hypercube $[0, 1]^d$. The splitting values are all of the form $k/2^i$, for some odd integer k, $1 \leq k < 2^i$. We call any cell which could result from the application of this method a *midpoint box*. The concept of such a *canonical set* of splitting planes will be considered later.

Unfortunately, there does not seem to be a single simple splitting rule that provides us with all the properties one might wish for (linear size, logarithmic depth, bounded aspect ratio, convexity, constant cell complexity). In [AMN+98] the BBD-tree was introduced. This tree uses a combination of two operations,

splitting and shrinking to provide for all of these properties (except for convexity). The BAR-tree [DGK99] provides all of these properties, by using nonorthogonal splitting planes, but the cells may have as many as 2^d bounding faces.

We now discuss two other splitting methods, the sliding-midpoint and the canonical sliding-midpoint methods. The sliding-midpoint method was first introduced in [MA97] and was subsequently analyzed empirically in [MM99a]. This method produces no empty nodes. Although cells may not have bounded aspect ratio, observe that every skinny cell that is produced by sliding is adjacent to a fat leaf cell. In [MM99b] we show that this is sufficient to satisfy the necessary packing constraint that fat subdivisions possess. The canonical sliding-midpoint method is introduced primarily for technical reasons. The proof of the main theorem of Section 3 relies on the presence of having a canonical set of splitting planes, while retaining the property that no empty cells are produced.

Sliding-midpoint: It first attempts to perform a midpoint split, by considering a hyperplane passing through the center of the cell and bisecting the cell's longest side. If the data points lie on both sides of the splitting plane then the splitting plane remains here. However, if a *trivial split* were to result (in which all the data points lie to one side of the splitting plane), then it "slides" the splitting plane towards the data points until it encounters the first such point. One child is a leaf cell containing this single point, and the algorithm recurses on the remaining points.

Canonical sliding-midpoint: Define the *enclosure* for a cell to be the smallest midpoint box that encloses the cell. During the construction phase, each node of the tree is associated both with its cell and the cell's enclosure. We first try to split the cell using a hyperplane that bisects the longest side of this enclosure (rather than the cell itself). Again, if this results in a trivial split, then it slides the splitting plane towards the data points until it encounters the first such point.

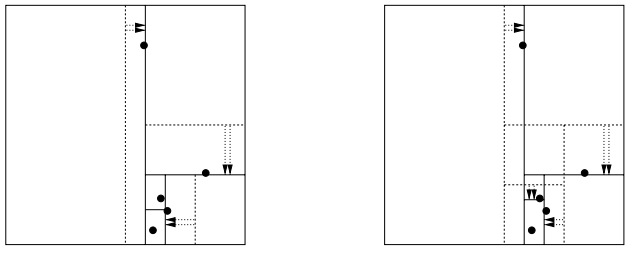

Fig. 1. Sliding-midpoint and canonical sliding-midpoint.

The differences between these two splitting methods is illustrated in Fig. 1. Notice that in the sliding-midpoint method the slides originate from a line that

bisects the cell (shown in dashed lines), whereas in the canonical sliding-midpoint method, the slides originate from the midpoint cuts of the enclosing midpoint cell (shown in dashed lines).

Because of prior sliding operations, the initial split used in the canonical sliding-midpoint method may not pass through the midpoint of the cell. After splitting, the enclosures for the two child cells must also be computed. This can be done in $O(d)$ time [BET93]. Thus, this tree can be constructed in $O(dn \log n)$ time, and has $O(n)$ nodes. just like the sliding-midpoint split kd-tree.

3 Points Clustered on Arbitrarily Oriented Flats

Let F be an arbitrary k-dimensional hyperplane (or k-*flat*, for short) in $\mathbf{R}^d$. We assume that F is in general position, and in particular that F is not parallel to any of the coordinate axes. Let S denote a set of data points sampled from a closed convex, *sampling region* of F according to some probability distribution function. We assume that the distribution function satisfies the following *bounded density assumption* [BWY80]. There exist constants $0 < c_1 \leq c_2$, such that for any convex open subregion of the sampling region with k-dimensional volume V, the probability that a given sampled point lies within this region is in the interval $[c_1 V, c_2 V]$. (This is just a generalization of a uniform distribution but allows some variation in the probability density.)

To avoid having to deal with boundary effects, we will assume that there are sufficiently many data points sampled, and that the query points are chosen from a sufficiently central region, such that with high probability the nearest neighbor ball for any query point lies entirely within the sampling region. More formally, fix any compact convex region on F, called the *query region*, from which query points will be sampled. Let w denote the diameter of this region. Now, take the data points to be sampled from a hypercube of side length $w' > w$ centered around this region, such that the local density of the distribution is independent of w'. Our results hold in the limit as w' tends to infinity. In [AMN96], it is shown that consideration of boundary effects for kd-trees with uniformly distributed points only tends to decrease the number of cells of the tree visited.

Let $B(r)$ denote a ball of radius r. Let $V_F(q, r)$ denote the k-dimensional volume of intersection of F and ball $B(r)$ centered at point q. If we restrict q to lying on F, then $V_F(q, r)$ is a constant for all q, which we denote as $V_F(r)$. Following the approach taken in [AMN96], let us first scale space so that the lower density bound becomes $c_1 = 1/V_k(1)$. After this scaling, a ball of unit radius is expected to contain at least one point of the sample. As observed in [AMN96], as k increases, a ball of unit radius is a very good approximation to the expected nearest neighbor ball. The reason is that $V_F(r)$ is growing as r^k, and so for large k, the probability that a data point lies in $B((1-\delta)r)$ drops rapidly with δ, and the probability that there is at least one point in $B((1+\delta)r)$ increases rapidly with δ.

Consider a kd-tree built for such a distribution, assuming the canonical sliding-midpoint splitting method. Our analysis will focus on the number of leaf

cells of the kd-tree that are visited in the search. The running time of nearest neighbor search (assuming priority search [AMN+98]) is more aptly bounded by the product of the depth of the tree and the time to access these nodes. This access time can be assumed to be $O(\log n)$ either because the tree is balanced, or auxiliary data structures are used. We focus just on the number of leaf cells primarily because in higher dimensions this seems to be the more important factor influencing the running time.

The main result of this section is that the expected number of cells of a canonical sliding-midpoint kd-tree that intersect a unit ball centered on F is exponential in k, but not in d. To see that the proof is nontrivial, suppose that we had stored the points in a regular grid instead. If the nearest neighbor ball contained even a single vertex of the grid, then it would overlap at least 2^d cells. The proof shows that in the canonical midpoint-split kd-tree, it is not possible to generate a vertex that is incident to such a large number of cells when the points lie on a lower dimensional flat. This feature seems to be an important reason that these trees adapt well to the intrinsic dimensionality of the point set. Although it is not clear how to establish this property for other splitting methods in the worst case, we believe that something analogous to this holds in the expected case.

Theorem 1. *Let S be a set of points from $\mathbf{R}^d$ sampled independently from a k-flat F by a distribution satisfying the bounded density assumptions and scaled as described above. Let T be a kd-tree built for S using the canonical sliding-midpoint splitting method. Then, the expected number of leaf cells of T that intersect a unit ball centered on F is $O(d^{k+2})$.*

For the complete proof, see [MM01]. Using Theorem 1 and the observation made earlier that a ball of unit radius is good approximation to (or larger than) the nearest neighbor ball, we have the following bound.

Corollary 1. *The expected number of leaf cells of T encountered in nearest neighbor searching is $O(d^{k+2})$.*

4 Points Clustered on Axis-Aligned Flats

We now consider the case where the set S of data points in $\mathbf{R}^d$ sampled independently from a distribution of bounded density along an axis-aligned k-flat. If in the kd-tree construction we split orthogonal to any of the $d - k$ coordinate axes that are orthogonally to the flat, the points will all lie to one side of this splitting hyperplane. The splitting hyperplane will slide until it lies on the flat. After any sequence of $2(d - k)$ such slides, the flat will be tightly enclosed within a cell. Splits along other axes will be orthogonal to the flat, and so will behave essentially the same a sliding-midpoint decomposition in k-space. The main complication is that the algorithm does not know the location of the flat, and hence these two types of splits may occur in an unpredictable order.

Let $G(k)$ denote the dimension dependent ratio of the volumes of a k-dimensional hypercube and a maximal enclosed k-ball for the metric inside the hypercube. Let $c(k) = (G(k)^{1/k} + 1)$. For example, for the L_∞ (max) metric the metric ball is a hypercube, and $c(k) = 2$. For the L_2 (Euclidean) metric $G(k) = k\Gamma(k/2)/(2^{k+1}\pi^{k/2})$. The proof is presented in [MM01].

Theorem 2. *Let S be a set of points from $\mathbf{R}^d$ sampled independently from an axis-aligned k-flat F by a distribution satisfying the bounded density assumptions described in Section 3. Let T be a kd-tree built for S using the canonical sliding-midpoint splitting method. Then, the expected number of leaf cells of T that intersect a unit ball centered on F is $O((d-k+1)c(k)^k)$.*

5 Empirical Results

We conducted experiments on the query performance of the kd-tree for data sets lying on a lower dimensional flat. We used the ANN library [MA97] to implement the kd-tree. We used priority search to answer queries. We present the total number of nodes, and the number of leaf nodes in our grades, because these parameters are machine-independent and closely correlated with CPU time.

5.1 Distributions Tested

Before discussing what we did in the experiments, we briefly describe the distributions used.

Uniform-on-orthogonal-flat: The dimension of the flat, k, is provided, and k dimensions are chosen at random. Among these dimensions, the points are distributed uniformly over $[-1, 1]$. For the other $(d-k)$ dimensions, we generate a uniform random coordinate that is common to all the points.

Uniform-on-rotated-flat: This distribution is the result of applying r random rotation transformations to the points in uniform-on-orthogonal-flat distribution. In the experiments, r is fixed at $d^2/2$. The flat is therefore rotated in a random direction. Each rotation is through a uniformly distributed angle in the range $[-\pi/2, \pi/2]$ with respect to two randomly chosen dimensions.

Our theoretical results for arbitrary flats apply only to the canonical sliding-midpoint method. This was largely for technical reasons. A natural question is how much this method differs from the more natural sliding-midpoint method. We tested both splitting methods for some other distributions, and discovered that their performances were quite similar. These results as well as additional experiments will be presented in the full version of the paper [MM01].

5.2 Points on a k-Flat

To support our theoretical bounds on number of leaf nodes visited when the point set is on a k-flat, we set up an experiment with both k and d varying,

while fixing the other parameters. This allows us to observe the dependency of the query performance (in terms of the number of nodes visited) relative to d and k. The Uniform-on-orthogonal-flat and Uniform-on-rotated-flat distributions were used in the experiments. We fixed d at 4, 8, 12, 16, 20, 24, 32, 40 (note that the scale is nonlinear), and k ranged from 4 to $\min(d, 16)$. The number of points, n ranged from 40 to 163,840. The queries were sampled from the same distribution. The number of query points was set to $\min(n, 2560)$.

Normally, ANN places a tight bounding box around the points. Such a bounding box would tightly wrap itself around the flat reducing the problem to a purely k-dimensional subdivision. In order to observe the behavior of the scenario considered in Theorem 2, we modified the library so that the initial bounding box is the hypercube $[-1, 1]^d$. The results of this modification are showed in Fig. 2. Note that we plotted the logarithm base 10 of the number of nodes visited. As predicted, the running shows a strong dependence on k, and very little dependence on d. However, it does not grow as fast as predicted by Theorem 2. This suggests that the average case is much better than our theoretical bounds.

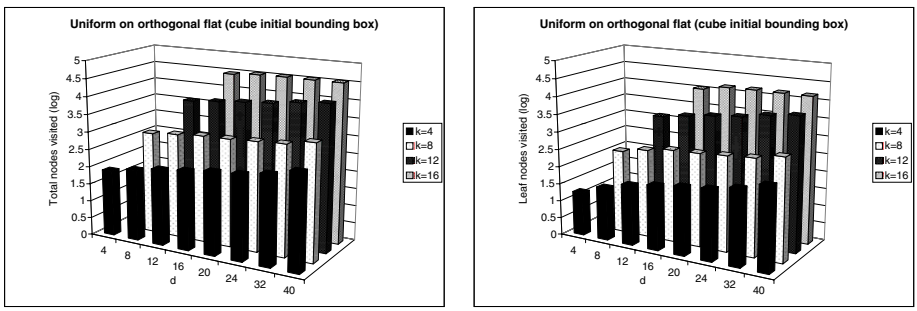

Fig. 2. Number of total and leaf nodes visited, $n = 163,840$, Uniform-on-orthogonal-flat distribution with cube initial bounding box

The Uniform-on-rotated-flat distribution is also used in the experiment to see the effect assuming that data is uniform on an arbitrarily oriented flat. For this distribution, the canonical sliding-midpoint is a little slower (typically, the difference is less than 5%) than the sliding-midpoint in few cases. In general, the number of nodes visited still shows a greater dependence on k than on d, but the dependence on d has increased, as predicted by Theorem 1. Yet, the growth rate is still less than what the theorem predicts. We tested the sensitivity of our result to the presence of multiple clusters. We also ran experiments on the standard kd-tree. Although we could not prove bounds on the expected query time, the empirical performance was quite similar to these other methods. This supports the rule-of-thumb that the standard-split kd-tree tends to perform well when data and query points are chosen from a common distribution.

5.3 Comparison with Theoretical Results

In this section, we take a closer look on whether our theoretical bounds can predict the actual query performance in terms of the number of leaf nodes visited. From Corollary 1, the expected number of leaf nodes of a kd-tree encountered in the search is $O(d^{k+2})$. We model this bound as $L = c_1(c_2 d)^{c_3 k}$, where L is the number of leaf nodes visited and c_1, c_2, c_3 are constants. We set up the experiment such that the data and query distributions are uniform-on-rotated-flat. The parameters are slightly different from the previous experiments. The number of random rotations is d^2, and there is no gaussian noise. The number of data points, n, remains at 163,840. We gathered results for $k = 1$ to 12 and $d = 10, 20, 40, 80$. The results are plotted in Fig 3.

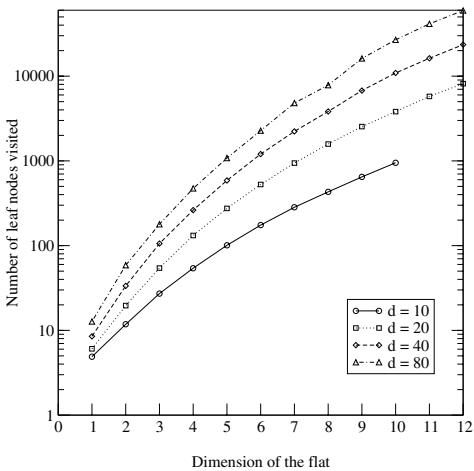

Fig. 3. Number of leaf nodes visited, $n = 163,840$, Uniform-on-rotated-flat distribution

The model suggests that the curves in Fig 3 should be linear. However, the empirical results show that it is not the case. We conjecture that this is due to boundary effects, which would presumably diminish as n increases. These boundary effects are more pronounced for larger values of k [AMN96]. Because of memory limitation, we cannot scale n exponentially with the value of k.

We observed that for smaller values of k (e.g. $k = 1, 2, 3$), the number of leaf nodes visited, L, is almost unchanged when n is increased. It indicates the boundary effects are minimum. Therefore we use the results from $k = 1, 2$ to find values of c_1, c_2, c_3 of our model equation. This yields the following equation,

$$L = 2.054(1.674 \cdot d)^{(0.312 \cdot k)}.$$

References

[AMN96] S. Arya, D. M. Mount, and O. Narayan. Accounting for boundary effects in nearest neighbor searching. *Discrete Comput. Geom.*, 16(2):155–176, 1996.

[AMN+98] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.

[Ben75] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[BET93] M. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes in Computer Science*, pages 188–199. Springer-Verlag, 1993.

[BWY80] J. L. Bentley, B. W. Weide, and A. C. Yao. Optimal expected-time algorithms for closest-point problems. *ACM Trans. Math. Software*, 6(4):563–580, 1980.

[Cle79] J. G. Cleary. Analysis of an algorithm for finding nearest neighbors in euclidean space. *ACM Trans. Math. Software*, 5(2):183–192, 1979.

[CP96] M. Carreira-Perpiñán. A review of dimension reduction techniques. Technical Report CS–96–09, Dept. of Computer Science, University of Sheffield, UK, 1996.

[DGK99] C. Duncan, M. Goodrich, and S. Kobourov. Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees. In *Proc. 10th ACM-SIAM Sympos. Discrete Algorithms*, pages 300–309, 1999.

[FBF77] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Software*, 3(3):209–226, 1977.

[FK94] Christos Faloutsos and Ibrahim Kamel. Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension. In *Proc. Annu. ACM Sympos. Principles Database Syst.*, pages 4–13, 1994.

[Fuk90] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 2nd edition, 1990.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 604–613, 1998.

[Kle97] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimension. In *Proc. 29th Annu. ACM Sympos. Theory Comput.*, pages 599–608, 1997.

[MA97] D. M. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching. Center for Geometric Computing 2nd Annual Workshop on Computational Geometry, 1997.

[MM99a] S. Maneewongvatana and D. Mount. Analysis of approximate nearest neighbor searching with clustered point sets. In *ALENEX*, 1999.

[MM99b] S. Maneewongvatana and D. Mount. It's okay to be skinny, if your friends are fat. Center for Geometric Computing 4th Annual Workshop on Computational Geometry, 1999.

[MM01] S. Maneewongvatana and D. Mount. On the efficiency of nearest neighbor searching with data clustered in lower dimensions. Technical Report CS-TR-4209, Dept. Computer Science, Univ. Maryland, 2001.