It's okay to be skinny, if your friends are fat*

Songrit Maneewongvatana and David M. Mount†

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. Our primary interest is in data structures that can be stored in main memory (as opposed to database applications). Since such data sets can be quite large in practice (tens of thousands to tens of millions) we limit ourselves to consideration of data structures whose total space grows linearly with d and n. Under these stringent assumptions, it is difficult to achieve very efficient search times in higher dimensions. Arya, et al. [1] showed that $(1+\epsilon)$ approximate nearest neighbor queries can be answered from such a data structure in $O((d/\epsilon)^d \log n)$ time, assuming $O(dn)$ storage. There have been a number of approaches to reduce the exponential dependence on d. The closest to achieving the linear storage bound are the data structures by Indyk and Motwani [5], which achieves $O(d \log^{O(1)} n)$ search time using $O(1/\epsilon)^d n \log^{O(1)} n$ storage, and by Kleinberg [7] which achieves $O(n + d\log^3 n)/\epsilon^2$ search time using $nd \log^{O(1)} n/\epsilon^2$ storage.

One of the most popular linear-storage method for nearest neighbor searching are those data structures based on hierarchical decompositions These include kd-trees [4], R-trees[10], the SR-tree [6], and other numerous variants. An important question raised by the large number of different data structures is that of determining the essential properties that a data structure should possess in order to provide efficient search times for these query problems. In [1] a list of sufficient properties was presented for answering approximate nearest neighbor queries efficiently. Define the *aspect ratio* of a rectangular cell to be the ratio of the lengths of its longest and shortest sides. These properties are: a tree of $O(n)$ size and $O(\log n)$ depth, bounded aspect ratio of cells, and cell complexity depending only on dimension. They introduced the BBD-tree, which satisfies all of these properties. The BBD-tree has the unpleasant property of producing nonconvex cells. Duncan, Goodrich and Kobourov [3] introduced the BAR-tree, which satisfies these properties and also has convex cells. In [9, 8] the *sliding-midpoint* splitting rule was introduced (definitions given in the next section).

Our main result is a proof that the presence of cells of unbounded aspect ratio in the sliding-midpoint tree is not an impediment to its ability to answer queries efficiently. The efficiency of query processing depends on a result, called the *packing lemma*. It states that the number of cells of a given size s that can overlap a ball of radius r in dimension d is bounded by $O(r/s)^d$. This is trivial to prove for trees in which all the cells are *fat* (have bounded aspect ratio.) But the sliding-midpoint method does not have this property. We show that, even though the individual cells produced by the sliding-midpoint rule may not be fat, as a group they still satisfy the packing lemma. The key property that the sliding-midpoint rule has is that with every skinny cell it produces, it produces a neighboring cell that is fat. Thus, it is not possible to create a large number of parallel skinny cells, the sheets of paper in a book. It is the absence of many parallel skinny cells that is important to efficiency.

We assume that the reader is familiar with the basic facts about kd-tree. See [2] for further information. The tree defines a hierarchical partition of space into *cells*, each of which is a hyperrectangle with axis-orthogonal faces. The structure of the tree is determined by its *splitting rule*. Some examples of splitting rules include the following.

Standard split: Friedman, Bentley and Finkel [4] selected the splitting dimension to be the one for which the data points have the maximum *spread* (difference between the maximum and minimum values).

*The support of the National Science Foundation under grant CCR–9712379 is gratefully acknowledged.

†Department of Computer Science, University of Maryland, College Park, Maryland. {songrit,mount}@cs.umd.edu.

The splitting value is chosen to be the median in that dimension. This is the most well-known and widely used splitting method.

Midpoint split: The splitting hyperplane passes through the center of the cell and bisects the longest side of the cell.

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.

The sliding midpoint splitting method was first introduced in the ANN library for approximate nearest neighbor searching [9] and was subsequently analyzed empirically in [8]. This method produces no empty nodes, and hence the tree has $O(n)$ 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. Below we show that this is sufficient to satisfy the necessary packing lemma that fat subdivisions possess. This tree can be constructed in $O(dn \log n)$ time, as is true for the standard kd-tree [8]. Our main results are stated below. We refer the reader to the complete paper for the proofs.

Lemma 1. (Packing Lemma) *Consider any set C of cells of the sliding-midpoint tree with pairwise disjoint interiors, each of size at least s and each intersecting a Minkowski ball B of radius r. Then C is of size at most $d\left(1 + \lceil \frac{4r}{s} \rceil\right)^d$.*

Theorem 1. *The number of leaf cells visited by a priority search algorithm applied to a sliding-midpoint kd-tree in computing a $(1+\epsilon)$-approximate nearest neighbor is at most $d\lceil 1 + 4d/\epsilon \rceil^d$ (in any Minkowski metric).*

References

[1] 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.

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

[3] 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.

[4] 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.

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

[6] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 369–380, 1997.

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

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

[9] D. M. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching. Center for Geometric Computing 2nd Annual Fall Workshop on Computational Geometry, URL: `http://www.cs.umd.edu/~mount/ANN.`, 1997.

[10] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 71–79, 1995.