Appeared in WADS 2001, pp. 276-286.

The Analysis of a Probabilistic Approach to Nearest Neighbor Searching[⋆]

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

Department of Computer Science
University of Maryland
College Park, Maryland

1 Introduction

Given a set S of n data points in some metric space. Given a query point q in this space, a *nearest neighbor query* asks for the nearest point of S to q. Throughout we will assume that the space is real d-dimensional space $\Re^d$, and the metric is Euclidean distance. The goal is to preprocess S into a data structure so that such queries can be answered efficiently. Nearest neighbor searching has applications in many areas, including data mining [7], pattern classification [5], data compression [10].

Because many applications involve large data sets, we are interested in data structures that use linear storage space. Naively, nearest neighbor queries can be answered in $O(dn)$ time through brute-force search. Although nearest neighbor searching can be performed efficiently in low-dimension spaces, for all known exact linear-space data structures, search times grow exponentially as a function of dimension. Thus for reasonably large dimensions, brute-force search is often the most efficient in practice. One approach to reducing the search time is through *approximate nearest neighbor search*. A number of data structures for approximate nearest neighbor searching have been proposed [1, 3, 11]. The phenomenon of concentration of distance would suggest that approximate nearest neighbor searching is meaningless. Fortunately, the distributions that arise in applications tend to be clustered in lower dimensional subspaces [6]. Good search algorithms take advantage of this low-dimensional clustering.

The fundamental problem that motivates this work is the lack of *predictability* in existing practical approaches to nearest neighbor searching. In high dimensions, exact search is no better than brute-force, and approximate search algorithms are acceptably fast only when the allowed

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

approximation factors are set to unreasonably high values [1]. The goal of this work is to address this shortcoming by providing a practical data structure for nearest neighbor searching that is both *efficient* and guarantees *predictable performance*.

We measure performance in an aggregate sense, rather than for individual queries. We assume that queries are drawn from some known probability distribution. The user provides a desired *failure probability*, p_f, and the resulting search fails to return the true nearest neighbor with probability at most p_f. The query distribution is presented by a set of *training queries* as part of the preprocessing stage. The training data permits us to tailor the data structure to the underlying structure of the point distribution.

The idea of allowing failures in nearest neighbor searching was proposed by Ciaccia and Patella [2], but no performance guarantees were provided. We present a data structure called an *overlapped split tree*, or *os-tree* for short. The tree, which we first introduced in [12], is a generalization of the well known BSP-tree, in which each node of the tree is associated with a convex region of space. However, unlike the BSP-tree in which the child regions partition the parent's region, here we allow these regions to overlap one another. The degree of overlap is determined by the failure probability and the query distribution as represented by the training points.

Based on empirical experiments on both synthetic and real data sets, we have shown that compared to the popular kd-tree data structure, it is possible to achieve significantly better predictability of failure probabilities while achieving running times that are competitive, and sometimes superior to the kd-tree [12]. However that paper did not address the efficiency of the data structure except for experiments on synthetic data sets. In this paper we present a theoretical analysis of the os-tree's efficiency and performance and provide experimental evidence of its efficiency on real application data sets.

2 Overlapped Split Tree

The os-tree was introduced by the authors in an earlier paper [12]. We summarize the main elements of the data structure here. It is a generalization of the well-known *binary space partition (BSP) tree* (see, e.g., [4]). Consider a set S of points in d-dimensional space. A BSP-tree for this set of points is based on a hierarchical subdivision of space into convex polyhedral *cells*. Each node in the BSP-tree is associated with such a cell and the subset of points lying within this cell. The root node is associated with the entire space and the entire point set. A cell is split into two disjoint cells by a *separating hyperplane*, and these two cells are then associated with the children of the current node. Data points are then distributed among the children according to which side of the separating hyperplane they lie. The process is repeated until the number of points associated with a node falls below a given threshold, which we assume to be one throughout.

The os-tree generalizes the BSP-tree by associating each node with an additional convex polyhedral region called a *cover*. (In [12] the term *approximate cover* was used.) Consider a node δ in the tree, associated with the point subset S_δ. The cover C_δ is constructed to contain a significant fraction of the points of $\Re^d$ whose nearest neighbor is in S_δ, that is, C_δ covers a significant fraction of the union of the Voronoi cells [4] of the points of S_δ. Computing these Voronoi cells would be too expensive in high dimensional spaces, and so the covers are computed with the aid of a large set of *training points* T, which is assumed to be sampled from the same distribution at the data points, and where $|T| \gg |S|$. For each point in T we compute its nearest neighbor in S. The cover C_δ contains all the points of T whose nearest neighbor is in S_δ. See Fig. 1. As the size of T increases, the probability that a point lies outside of C_δ but has its nearest neighbor in S_δ decreases.

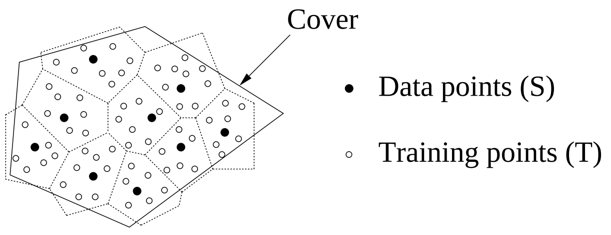

Fig. 1. The cover for a set of data points (filled circles) contains all the training points (shown as white circles) whose nearest neighbor is in this subset.

As with the BSP tree, covers are not stored explicitly, but rather are defined by a set of *boundary hyperplanes* stored at the nodes of each of the ancestors. In typical BSP fashion, the parent cell is split by a hyperplane, which partitions the point set and cell. We introduce two halfspaces, supported by parallel hyperplanes. One covers the Voronoi cells associated with the left side of the split and the other covers the

Voronoi cells associated with the right side. These hyperplanes are stored with the parent cell. Thus, as we descend the tree, the intersection of the associated halfspaces implicitly defines the cover.

The construction algorithm works recursively, starting with the root. The initial point set consists of all the data points, and the initial cell and cover are $\Re^d$. Consider a node δ containing two or more data points. First, we compute a separating hyperplane H for the current point set (see Fig. 2). This hyperplane is chosen to be orthogonal to the largest eigenvector of the covariance matrix of the points of S_δ. This is the direction of maximum variance for the points of S_δ [9]. The position of the hyperplane is selected so that it bisects the points of S_δ. Let S_l and S_r denote the resulting partition, and define T_l and T_r analogously. (This partition is indicated using circles and squares in Fig. 2.)

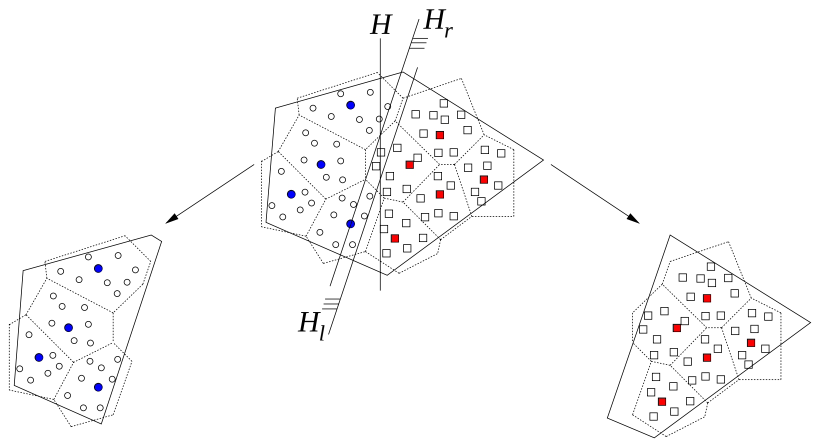

Fig. 2. The construction of the boundary hyperplanes (H_l and H_r) and the resulting partition of the data and training points.

To determine the orientation of the boundary hyperplanes, we invoke support-vector machines (SVM) to the subsets S_l and S_r. SVM was developed in learning theory [13] to find the hyperplane that best separates two classes of points in multidimensional space. SVM finds a splitting hyperplane with the highest margin, which is defined to be the sum of the nearest distance from the hyperplane to the points in S_l and the nearest distance to the points in S_r. After SVM determines orientation of the splitting hyperplane, the two boundary hyperplanes H_l and H_r are chosen to be parallel to this hyperplane, such that H_l bounds all the points of T_l and H_r bounds all the points of T_r. The hyperplanes H_l and H_r are stored in node δ, and the process continues recursively on each of the two subsets.

To answer the nearest neighbor query q, the search starts at the root. For any leaf, the distance to the point in this node is returned. At any internal node δ, we first determine which cover(s) the query lies. This is done in $O(d)$ time by comparing q against each of the boundary hyperplanes. If q lies in only one cover, then we recursively search the corresponding subtree and return the result. Otherwise, the subtree closest to q (say the left) is searched first, and the distance d_l to the nearest neighbor in this subtree is returned. If the distance from q to the other (right) boundary hyperplane is less than d_l, then this subtree is also searched resulting in a distance d_r. The minimum of d_l and d_r is returned.

It is easy to see that this search will fail only if for some node δ, the query point q lies in the Voronoi cell of some point $p \in S_\delta$, but lies outside the associated cover. It is not hard to show that if T and S are independent and identically distributed, then the probability of such a failure is proportional to $|S|/|T|$. (The proof is omitted from this version.) By making the training set sufficiently large, we can reduce the failure probability below any desired threshold.

3 Theoretical Analysis

In this section we explore the expected search time of the os-tree. As the search visits each nonleaf node of the os-tree, there are three possibilities: visit the left child only, visit the right child only, or visit both children. Suppose that the probability that we visit both children at any given node is bounded above by b, $0 < b \leq 1$. Let $T(n)$ denote the expected running time of the search algorithm given a subtree with n data points. Then because we do $O(d)$ computations at each node and split the data points evenly at each step, it follows that up to constant factors, $T(n)$ is bounded by the following recurrence

$$T(n) \;\leq\; 2bT(n/2) + (1-b)T(n/2) + d \;=\; (1+b)T(n/2) + d.$$

Solving the recurrence yields

$$T(n) \in O\left(dn^{\lg(1+b)}\right).$$

Thus the analysis of the expected search reduces to bounding the value of b.

In general b is a function of the dimension d and the point distribution. It is not difficult to contrive distributions in which, on any given node, virtually all query points visit both subtrees. In such cases the above analysis suggests that the running time is no better than brute-force search. Our analysis is based on some assumptions on the data distribution, which we believe are reasonable models of typical real data sets.

An important aspect of real data sets in high dimensional spaces is that many of the coordinates are correlated and some exhibit very small variances. Hence, points tend to be clustered (at least locally) around low dimensional spaces. A common way of modeling this is through *principal component analysis*. Consider a fixed node δ in the tree and the subset S_δ of data points associated with this node, and let n_δ be the cardinality of this set. Let C_δ denote the corresponding cover. Because δ will be fixed for the rest of the analysis, we will drop these subscripts to simplify the notation. We assume that the data points are sampled from a d-dimensional multivariate distribution. Let $\boldsymbol{x} = \{x_i\}_{i=1}^d$ denote a random vector from this distribution. Let $\boldsymbol{\mu} \in \Re^d$ denote the mean of this distribution and let $\boldsymbol{\Sigma}$ denote the $d \times d$ *covariance matrix* for the distribution [9],

$$\boldsymbol{\Sigma} = E((\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T).$$

This is a symmetric positive definite matrix, and hence has d positive eigenvalues. We can express this as $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, where $\boldsymbol{U}$ is a $d \times d$ orthogonal matrix whose columns $\{\boldsymbol{u}_i\}_{i=1}^d$ are the eigenvectors of $\boldsymbol{\Sigma}$ and $\boldsymbol{\Lambda}$ is a $d \times d$ diagonal matrix, whose entries $\{\lambda_i\}_{i=1}^d$ are the eigenvalues of $\boldsymbol{\Sigma}$. We may assume that the eigenvalues are sorted in decreasing order, so that λ_1 is the largest eigenvalue. By applying the transformation $\boldsymbol{y} = \boldsymbol{U}^T(\boldsymbol{x} - \boldsymbol{\mu})$, we map the points to a reference system in which the samples have mean $\mathbf{0}$, and a diagonal covariance matrix with the prescribed eigenvalues. The coordinates are now uncorrelated. We may assume henceforth that all point coordinates are given relative to this new reference system, the λ_i is the distribution variance for the ith coordinate. Henceforth, let $\sigma_i = \sqrt{\lambda_i}$.

Given a parameter $\gamma > 0$ define the *pseudo-dimension* $\hat{d}$, to be minimum integer such that $1 \le \hat{d} \le d$ and

$$\sum_{\hat{d} < i \le d} \frac{\sigma_i^2}{\sigma_1^2} \le \gamma^2.$$

Under the assumption that the points are clustered in a low-dimension subspace, we would expect that $\hat{d}$ is small relative d. Let $\widehat{H}$ be the $\hat{d}$-dimensional hyperplane spanned by the first $\hat{d}$ eigenvectors of $\boldsymbol{\Sigma}$. Given a query point $\boldsymbol{q} = (q_i)_{i=1}^{d}$ sampled from the same distribution as the data points, let $\hat{\boldsymbol{q}}$ denote its orthogonal projection onto $\widehat{H}$, and let $\epsilon(\boldsymbol{q}) = ||\boldsymbol{q} - \hat{\boldsymbol{q}}||$, where $||\cdot||$ denotes Euclidean distance. It is well known [9] that the expected value of $\epsilon^2(\boldsymbol{q})$ is

$$E(\epsilon^2(\boldsymbol{q})) = \sum_{\hat{d}<i\leq d} \sigma_i^2 \leq (\gamma\sigma_1)^2.$$

Our analysis is based on the following *distribution assumptions*. The first is that the pseudo-dimension $\hat{d}$ is significantly smaller that the dimension d of the space. The second is a type of Lipschitz condition on the distribution function, which states that point densities are bounded relative to the principal components. In particular, consider the restriction of the point distribution to the first $\hat{d}$ coordinates. We assume that there exist positive constants c_1 and c_2 such that given any point $\hat{\boldsymbol{q}} \in C \cap \widehat{H}$ and for all sufficiently small positive r, (1) the probability of a point lying within a ball of radius r centered at $\hat{\boldsymbol{q}}$ is at least $(c_1 r/\sigma_1)^{\hat{d}}$, and (2) for all positive x, the probability that $|q_1| \leq x$ is at most $c_2 x/\sigma_1$. Note that these conditions are satisfied for a uniform distribution assuming that the first $\hat{d}$ eigenvalues are bounded away from 0.

Recall that in the construction of the os-tree, we first partition the points into equal sizes using a separating hyperplane that is orthogonal to the largest eigenvector of the sample covariance matrix. We assume that n is large enough that the differences in the sample covariance matrix and the distribution covariance matrix are negligible. In the os-tree construction, we use SVM to compute the best orientation for the boundary hyperplanes. To simplify the analysis, let us assume that the boundary hyperplanes are chosen to be orthogonal to the largest eigenvector. Due to space limitations the proof has been omitted. It will be presented in the full version of the paper.

Theorem 1. *For any b, $0 < b \leq 1$ and for pseudo-dimension $\hat{d}$ defined by*

$$\gamma \leq \frac{b^{1.5}}{c_2 4\sqrt{3}}.$$

there is a value n_b (depending on b, c_1, c_2, and $\hat{d}$), such that for any node δ in the os-tree associated with at least n_b points and satisfying distribution assumptions stated earlier, the probability that either (1) a random query point $\boldsymbol{q}$ visits both children in the os-tree search or (2) the search returns an incorrect result from this node is at most b.

4 Experimental Results

We used both synthetic and real data sets. Because the os-trees require a relatively large number of training points, one advantage of synthetic data sets is that we can generate training sets of arbitrary size. To emulate real data sets, we choose a distribution that is clustered in subspaces having low intrinsic dimensionality, which we call *clustered rotated flats*. In this distribution, points are distributed among clusters, whose centers are sampled uniformly from a hypercube $[-1, 1]^d$. Each cluster contains a low dimensional flat. We denote *fat dimension* for each dimension on the flat, and *thin dimension* for others. The fat dimensions are randomly chosen among the coordinates of the full space. Points are distributed evenly to each cluster. In fat dimensions, point are drawn from uniform distribution in the range of $[-1, 1]$. In thin dimensions, we use a Gaussian distribution with a standard deviation of σ_{thin}. Each flat is then rotated independently. This is done by repeatedly applying the rotation matrix A to all points in the cluster. A is an identity matrix with four elements $A_{ii} = A_{jj} = \cos(\theta), A_{ij} = -A_{ji} = \sin(\theta)$, where i and j are randomly chosen axes and θ is randomly chosen from $-\pi/2$ to $\pi/2$. In the experiments, the number of clusters is fixed at 5, the number of fat dimensions is $\lfloor 3d/10 \rfloor$.

Two real data sets are used in the experiments. The first set is *NASA MODIS satellite images*. MODIS is a sensor aboard a satellite to acquire spatial data in 36 spectral bands of which 26 were usable. The data from the sensor are archived into files according to region of the earth and particular time. Both data and query sets contained around 13K points. The second set is *NOAA World Ocean Atlas* This data set contains information about some basic attributes of the oceans of the world. Examples of attributes are temperature, salinity, and so on. We use 8 attributes in the data set. The data type of all attributes is floating point. Both data and query set contained around 11K points.

4.1 Eigenvalues of Point Sets

In our analysis, we assumed that if we sort the eigenvalues of the data set associated with each node in decreasing order, these values decrease rapidly. This assumption is generally valid for most of the synthetic data sets. It is also true in both real data sets we use. We recorded all eigenvalues of the point set in each node in os-trees during tree construction. These eigenvalues are sorted and normalized. We computed these relative eigenvalues averaged over the nodes at each level of the os-tree.

In Fig. 3 we show the first, second, third, fifth eigenvalues and every five eigenvalues after that. Note that the plot is log-scale along the y-axis. Level 0 represents the root node, level 1 represents the children of the root node and so on. Only the first five eigenvalues are consistently greater than 0.1 in several levels in the tree. The rest decrease rapidly. Some of the lower eigenvalues are left unplotted at near leaf levels because they are zero. This is because there are not enough points in such nodes to span all the dimensions. The results of the NOAA ocean data set are quite similar, relatively few (three) eigenvalues out of 8 eigenvalues are greater than 0.1.

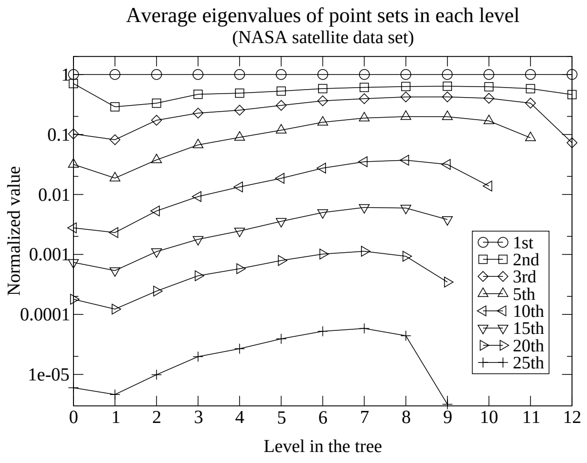

Fig. 3. Eigenvalues of point sets in each level in the tree. NASA satellite data set.

The results of a synthetic data set are shown in Fig. 4. We generated a data set with 32K points in 30 dimensional space and set $\sigma_{\text{thin}} = 0.05$. The number of rotations is 30. The clusters in this data set are 9-dimensional flats. The results show that there is a significant gap between 9th and 10th eigenvalues. The first nine eigenvalues capture the major characteristics of the data set. Observe that in the first few levels of the tree each node spans multiple clusters, and hence the rapid reduction in eigenvalues is not as evident as it is in latter levels, where clusters are better isolated.

4.2 Number of Points in the Overlap Region

In the second set of experiments, we investigated the fraction of data points of each node that fall in the overlap region between the bound-

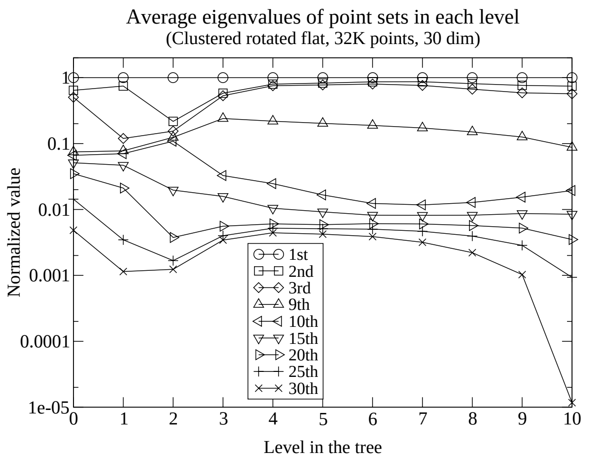

Fig. 4. Eigenvalues of point sets in each level in the tree. Synthetic data set

ary hyperplanes. These are points for which the search may visit both children, and hence is related to the value of b described in Section 3.

Fig. 5 shows the fraction of points that are in the overlap region from our real data sets. Again, they show the average value for nodes at the same level, where level 0 is the root. The size of the overlap region depends on the value of desired failure probability. To achieve a small failure probability, a large training set is needed. Additional points in the training set may widen the overlap region. Consequently, the overlap region may contain more data points. The fraction usually falls between 0.3 and 0.7 in various levels in the tree. Note that near the leaf level this fraction drops significantly. This is because there are very few points in the node relative to the dimension of the space. Therefore the Voronoi diagram can be approximated much better by a single hyperplane. Similar behaviors are also observed for NASA satellite data and the synthetic data set that we tested.

4.3 Comparison with kd-tree Search

We considered the search performance of the os-tree against an approximate version of the the well-known kd-tree data structure [8] The difficulty in comparing these data structures is that the search models are different: probably-correct search in the os-tree and approximate nearest neighbor search in the kd-tree. To produce a realistic comparison, we adjusted the approximation factor (ϵ) of the kd-tree so that the resulting failure probability of the kd-tree matches that of the os-tree.

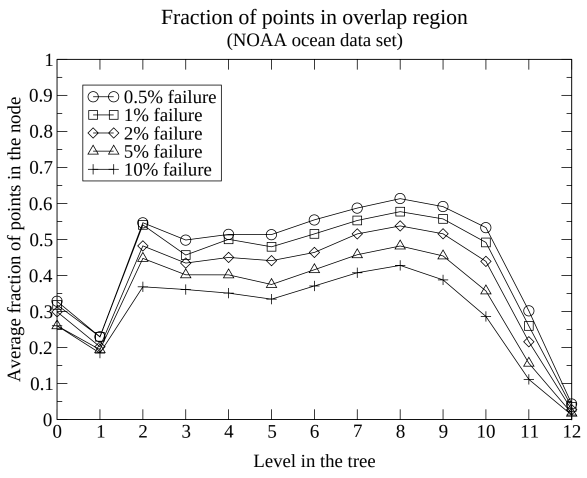

Fig. 5. Fraction of points in overlap region. NOAA ocean data sets.

We show the results for the synthetic data sets only. The comparative results of the real data sets as well as other synthetic data sets were presented in [12]. We used different parameters from the other experiments. The number of points is varied from 2K to 32K, the number of rotations is equal to the dimension, $\sigma_{\text{thin}} = 0.01$.

Fig. 6 compares the performance of both trees as we vary the number of dimensions, and the number of points. The average query time is used as the metric. From the figure, we see that the os-tree is competitive with the kd-tree except for high dimensional instances. If the number of leaf nodes visited is used as the measure the search performance, the os-tree search visits fewer nodes than the kd-tree in almost all cases. The reason for the differences in CPU times is largely due to the additional overhead suffered by the os-tree. Through the use of incremental distance calculation [1], the processing time at each node of the kd-tree is independent of dimension, while it is $O(d)$ for the os-tree.

References

1. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 573–582, 1994.
2. P. Ciaccia and M. Patella. Using the distance distribution for approximate similarity queries in high-dimensional metric spaces. In *Proc. 10th Workshop Database and Expert Systems Applications*, pages 200–205, 1999.
3. K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 160–164, 1994.

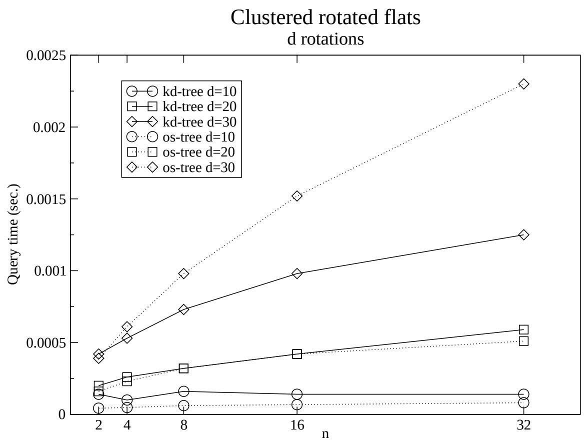

Fig. 6. Performance comparison. Using query time (sec) as the metric.

4. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 1997.
5. R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York, NY, 1973.
6. C. Faloutsos and I. 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.
7. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI Press/Mit Press, 1996.
8. 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.
9. K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 2nd edition, 1990.
10. A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic, Boston, MA, 1992.
11. 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.
12. S. Maneewongvatana and D. Mount. An empirical study of a new approach to nearest nearbor searching. In *ALENEX*, 2001.
13. V. Vapnik. *Statistical Learning Theory.* John Wiley & Sons, New York, NY, 1998.