

# Metric-based Shape Retrieval in Large Databases

Thomas B. Sebastian   Benjamin B. Kimia\*  
Brown University  
Providence, RI, USA

## Abstract

This paper examines the problem of database organization and retrieval based on computing metric pairwise distances. A low-dimensional Euclidean approximation of a high-dimensional metric space is not efficient, while search in a high-dimensional Euclidean space suffers from the “curse of dimensionality”. Thus, techniques designed for searching metric spaces must be used. We evaluate several such existing exact metric-based indexing techniques, and show that they require extensive computational effort. This motivates the development of an approximate nearest neighbor search technique where the  $k$  nearest neighbors are used to approximate the local neighborhood of a point. The resulting  $k$ NN graph is searched in a best-first fashion producing excellent indexing efficiency.

## 1 Introduction

Indexing into image databases is an increasingly important problem in computer vision. The straightforward approach to nearest neighbor search requires  $O(n)$  distance computations, where  $n$  is the number of items in the database, which is prohibitive for large databases. This has motivated the development of efficient nearest neighbor search techniques which can be broadly classified into two classes: (i) *spatial access methods* [4, 11, 18], and (ii) *distance-based indexing methods* [6, 20, 23, 5]. Our interest is in retrieval by shape content, Figure 1.

*Spatial access methods* are useful in shape retrieval applications where each shape is represented by a finite set of “labeled” features, and where distance between a pair of shapes is defined as the Euclidean distance between those features, *e.g.*, using moments, Fourier descriptors [14], invariants [17], active shape models [1], *etc.*. The Euclidean structure of the embedding space is used to divide the database elements into spatial clusters, *e.g.*, as in KD-tree [4], R-Tree [11] and Quadtree [18], thus avoiding the search of some clusters when querying the database [8, 10].

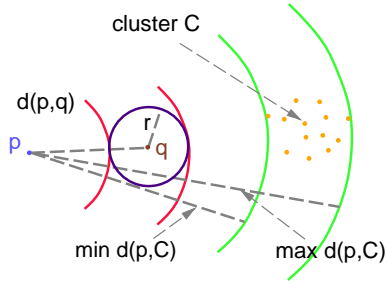
\*The support of NSF grants IRI-0083231 and BCS-9980091 is gratefully acknowledged.

550	551	560	567	572	589	593	613	616	678	
350	573	581	600	616	618	646	655	720	770	
739	748	753	756	756	777	788	811	812	836	
322	507	572	574	578	589	649	649	704	911	
209	255	265	268	268	273	276	289	299	334	
300	600	607	617	622	628	634	637	641	642	
535	556	558	614	628	637	646	652	685	693	

**Figure 1.** This figure from [19] illustrates indexing results for a database of 99 shapes (9 categories and 11 shapes in each category). The recognition rates are excellent, but each query shape has to be compared against all shapes in the database for finding the nearest neighbors.

One of the main drawbacks of spatial access methods is that their computational complexity is exponentially dependent on the dimensionality of the space (number of features), the so called *curse of dimensionality*. In practice, spatial access methods become intractable when the dimensionality of the database exceeds twenty [22]. A second drawback of spatial access methods for shape retrieval is that they cannot be used in conjunction with shape matching algorithms where shapes are not represented in terms of a finite set of attributable features, and only pairwise distances are available [3, 2, 19]. For example, matching methods using point clouds [3], elastically deformed shape outlines [2], and skeletal graphs [19] which only give a dissimilarity distance between shapes cannot be organized using spatial access methods.

*Distance-based indexing methods* rely solely on the pairwise distances for database organization and retrieval. Database elements with “similar” distances are grouped into clusters, and the triangle inequality is used to prune some clusters [6, 20, 23, 5], Section 2. However, we find that these methods have poor indexing efficiency for shape-based indexing. Another class of approaches that rely solely on distances are the *distance-preserving* methods like multidimensional scaling (MDS) [13] and Fastmap [9], which



**Figure 2.** This figure (adapted from [5]) illustrates how some clusters can be pruned using the triangle inequality. Let  $p$  be a pivot and  $C$  be one of the clusters. Let the range of distances from pivot  $p$  to the cluster  $C$  be  $[\min d(p, C), \max d(p, C)]$ . Let  $q$  be the query and  $r$  be the search tolerance of the query. If  $[d(p, q) - r, d(p, q) + r] \cap [\min d(p, C), \max d(p, C)] = \emptyset$ , then cluster  $C$  need not be examined by applying the triangle inequality.

project elements in a Euclidean structure while attempting to optimally preserve pairwise distances. Once the coordinates for each element are estimated, nearest neighbor is found using spatial access methods. MDS is not useful in our application because of its high computational complexity. Fastmap addresses the complexity issue, but performs poorly for non-Euclidean spaces [12].

The above finding motivates the development of a new approach. We attribute the computational complexity of MDS, which approximates a metric space with a finite dimensional Euclidean space, to the *global* characterization of the space. Alternatively, the topology of the space can be characterized *locally*, thus offering a degree of flexibility without necessarily requiring a large number of dimensions. Specifically, we represent the local neighborhood topology of each shape in terms of its  $k$  nearest neighbors. This results in a graph, namely, the  $k$ NN graph, where each element in the database is represented by a node, and is connected to its  $k$  nearest neighbors. The  $k$ NN graph is searched in a best-first fashion, starting from a few well-separated nodes (“seeds”) to find the nearest neighbors. We find that this approach gives excellent indexing efficiency. However, the drawback of this approach is that it is not exact in that it is not guaranteed to find the nearest neighbor every time. This approach is developed in Section 3 and results are presented in Section 4.

## 2 Distance-based Indexing Methods

This section reviews and evaluates several distance-based indexing methods for shape retrieval. We present these methods in terms of the following problem: given a query  $q$  and a search tolerance  $r$ , find all elements in the database  $S$  whose distance to  $q$  is less than  $r$ , *i.e.*, find

$\{p \in S \mid d(p, q) \leq r\}$ . The problem of finding the closest element to a query can be cast in similar terms, with the minimum distance at each step playing the role of the tolerance radius.

One of the earliest distance-based indexing approaches is the Burkhard-Keller tree (BK-Tree) [6]. In the BK-Tree, an arbitrary element  $p$  is chosen as the *pivot*, and other database elements are split into clusters based on their distances to the pivot. The triangle inequality is used to prune some clusters while querying the database, Figure 2.

Another approach that relies on partitioning the database elements is the *geometric near-neighbor access tree* (GNAT) [5]. In GNAT,  $k$  well-separated pivots (called split-points in [5]) are used to separate the database elements into  $k$  clusters, depending on the distance to the closest pivot. GNAT is then recursively built for each cluster. At each level, the triangle inequality is used to prune some clusters, Figure 2. The special case of GNAT with  $k = 2$  is the *generalized hyper-plane tree* (GH-Tree) [20].

Metric tree [20] or vantage point tree (VP-Tree) [23] uses one element (called vantage point in [23]), and the remaining elements are split into two clusters using the median distance as the threshold. The VP-Tree is then built recursively for the two clusters. When searching for the nearest neighbor of a query, some of the subtrees can be pruned using the triangle inequality. Note that VP-Tree is not a dynamic structure, in that it cannot be updated with the addition of elements to the database, due to the use of global measures like median distance.

Spatial approximation tree (SA-Tree) [15] approximates the Voronoi region for each pivot  $p$ , by identifying all elements that are closer to the pivot than to any other element. This problem is NP-complete, and hence the following heuristic is used. All elements are sorted in ascending order of distances from  $p$ , and are added to  $N(p)$  one at a time if they are closer to  $p$  than to any element in  $N(p)$ . All unassigned elements are then split into clusters, based on which element in  $N(p)$  they are closest to. The process is repeated for each cluster. SA-Tree is searched in a best-first fashion, and the triangle inequality is used to eliminate some subtrees. Note that SA-Tree is not dynamic as the tree construction algorithm requires all elements.

The Approximating Eliminating Search Algorithm (AESAs) [21] uses the matrix of pairwise distances between all elements in the database as the basic data structure. When searching with a query  $q$  and range  $r$ , an object  $p$  in the database is picked at random, and every object  $u$  that does not satisfy  $d(q, p) - r \leq d(q, u) \leq d(q, p) + r$  is eliminated. This process of picking a random object and eliminating more objects is repeated until only a few objects remain.

Most distance-based indexing methods attempt to create balanced data structures by forming equal-sized clus-

Method	Distance computations avoided		
	Average	Max	Min
VP-Tree	56 (23%)	228	0
<b>AESA</b>	<b>94 (39%)</b>	<b>235</b>	<b>6</b>
Unb. BKT-7	58 (24%)	231	0
Unb. BKT-15	22 (9%)	224	0
GH-tree	13 (5%)	216	0
GNAT-13	23 (10%)	227	0
GNAT-20	17 (7%)	213	0
GNAT-39	47 (20%)	201	0

**Table 1.** This table reports the indexing efficiency of VP-Tree [20, 23], AESA [21], unbalanced BK-tree [7] GH-Tree [20] and GNAT [5] for shape retrieval. The results are averaged over 140 queries. The unbalanced BK-Tree [7] and GNAT [5] depend on user-specified parameters. In the unbalanced BK-tree we have to set the maximum number of elements in the left subtree, and in GNAT the number of pivots has to set. Indexing performance for a few representative values are shown in both cases.

ters. A recent paper [7] claims that “unbalancing” is central to tackling the curse of dimensionality, where the histogram of pairwise distances is narrow [5]. This causes equal-sized clusters to have small distance ranges, the triangle inequality eliminates very few clusters. Hence, clusters with fixed distance ranges are used in [7] with some clusters will have having most of the elements. As in [7], we use a BK-Tree to evaluate unbalancing.

We have implemented and evaluated the following methods: VP-Tree [20, 23], AESA [21], unbalanced BK-Tree [7], GH-Tree [20] and GNAT [5]. The database consists of 384 shapes of which 240 shapes form a core database (30 categories and 8 shapes in each category), while the rest (144 distinct shapes) are used for querying. The distances between shapes are computed as the edit distance of their shock graphs [19]. All methods are exact, in that they are guaranteed to find the nearest neighbor of the query. Hence, the performance measure of interest is the *indexing efficiency* (the number of distance computations avoided in finding the nearest neighbor of the query). The results indicates some saving, Table 1. However, the best method avoids only 39% of distance computations, and is not useful for shape retrieval, as distances between shapes are expensive to compute. This motivates the development of a technique that keeps the number of distance evaluations to a minimum.

### 3 Approximating the Local Topology with kNN Graph

A key concept underlying the search of a database embedded in a Euclidean space is that the underlying space can be partitioned into regions. Specifically, given a set of

elements in a Euclidean space, the Voronoi tessellation associates a Voronoi region with each element, Figure 3a. This gives rise to a notion of a neighborhood, which can be represented in terms of the Delaunay graph<sup>1</sup>. The significance of this neighborhood is that if a query is closer to a database element than all its neighbors, then we have found the nearest element in the whole database [15]. Thus, gradient descent from an arbitrary node can be used to find the optimal solution. We now formalize these concepts.

**Definition 1** *The neighborhood of a node  $p$  in the Delaunay graph, denoted by  $\mathcal{N}(p)$ , is the set of elements that share a Delaunay edge with  $p$ .*

**Proposition 1** *Let  $S$  be the database represented using its Delaunay graph. Let  $p$  be a node in Delaunay graph, and  $\mathcal{N}(p)$  its neighborhood. For a given query  $q$ , if  $d(p, q) < d(a, q)$  for all  $a \in \mathcal{N}(p)$ , then  $d(p, q) < d(a, q)$  for all  $a \in S$ .*

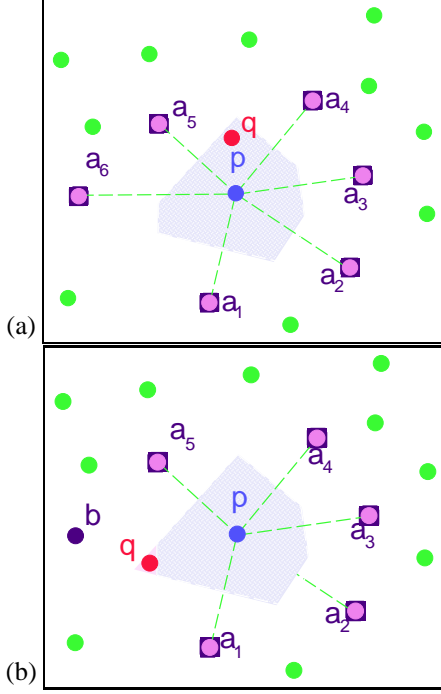
This proposition leads to the following algorithm to find the nearest neighbor in a Euclidean space using the Delaunay graph: For a query  $q$ , start at an arbitrary node in the Delaunay graph. Move along a Delaunay edge to the adjacent node with minimum distance to query  $q$ . Repeat this step until the current element is closer to the query than all its neighbors, and from Proposition 1 we have found the closest element to the query.

The key question is whether a similar technique can be adopted for indexing into high-dimensional metric spaces, in particular, for shape databases<sup>2</sup>. It has been shown that the Delaunay graph of an arbitrary metric space constructed from pairwise distances alone is the complete graph [15], *i.e.*, an edge is present between every pair of nodes. Hence,  $\mathcal{N}(p) = S \setminus \{p\}$ , and distances to all the elements have to be computed, in order to process the first node, which is exactly what we set out to avoid.

Observe that the main feature of the Delaunay graph-based representation of a Euclidean database is that the local topology is established for each element in terms of a few neighbors, which allows for an efficient nearest neighbor search by using local comparisons. Similarly, we need to approximate the local topology of each shape in the database, which is done in terms of its  $k$  nearest neighbors. In other words, we use the *k nearest neighbor graph (kNN graph)* as a local approximation to the Delaunay graph. Note how this local approximation contrasts with MDS [13] where each element is represented relative to all other elements. In the  $k$ NN graph each element in the database is a node, and is connected to its  $k$  nearest neighbors. With this

<sup>1</sup>Delaunay graph is the dual of the Voronoi diagram, where each element  $p$  is connected to all elements that share a Voronoi edge with  $p$  [16].

<sup>2</sup>Recall that shapes typically do not have a coordinate representation, and indexing has to rely solely on the pairwise distances.

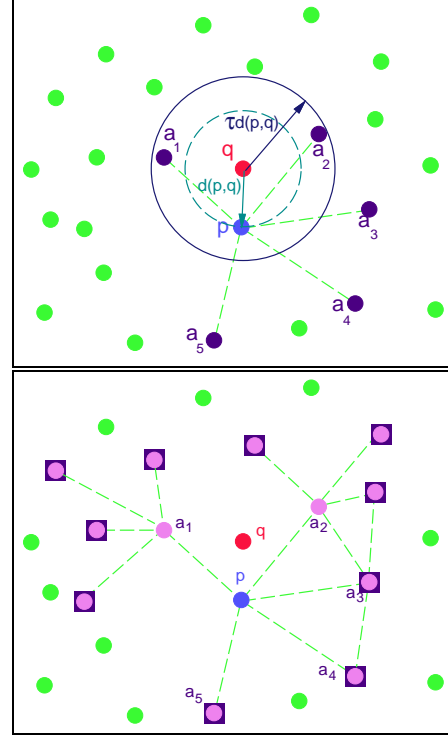


**Figure 3.** (a) This figure shows the Voronoi region (shaded) of an element  $p$  in 2D given the set of generator points (all dots). The neighborhood of  $p$  is defined using the Delaunay graph. Delaunay edges (dotted lines) connect  $p$  and its neighbors  $a_i, i = 1, \dots, 6$  (dots with a boxed surround). (b) This figure illustrates that if the  $k$ NN graph is used to approximate the Delaunay graph, Proposition 1 need not hold. In the above example, the element  $p$  is connected to five nearest neighbors  $a_i, i = 1, \dots, 5$ , and  $q$  is closer to  $p$  than all five, *i.e.*,  $d(p, q) < d(a_i, q)$ . However, the nearest neighbor of the query  $q$  is element  $b$ .

approximation, however, the analogue to Proposition 1 need not hold, *i.e.*, even when the current element is closer to the query than all its neighbors, we are not guaranteed that it is the nearest neighbor of the query, as illustrated in Figure 3b. To account for this approximation error, we increase the number of elements in the neighborhood by relaxing the definition of “closeness” in Proposition 1. A fixed threshold  $\mathcal{T} (\geq 1.0)$  is used to relax the definition of “closeness” as follows (see Figure 4).

**Definition 2  $\mathcal{T}$ -closer:** An element  $p_i$  is  $\mathcal{T}$ -closer to  $q$  with respect to  $p$  if  $d(p_i, q) \leq \mathcal{T}d(p, q)$ .

Note that if we set  $\mathcal{T} = 1$ , we get the definition of “closer” that is used in Proposition 1. We use the notion of  $\mathcal{T}$ -closer to define an *extended neighborhood*  $\mathcal{EN}(p, q)$  of a node  $p$  in the  $k$ NN graph with respect to query  $q$ .  $\mathcal{EN}(p, q)$  contains all neighbors of node  $p$ . In addition, it recursively includes all the neighbors of each neighbor of  $p$ , that is  $\mathcal{T}$ -closer to  $q$  with respect to  $p$ . This implies that a node belongs to



**Figure 4.** This figure illustrates how  $\mathcal{T}$ -closer is used to define the extended neighborhood  $\mathcal{EN}(p, q)$  of the node  $p$  in a  $k$ NN graph, with respect to a query  $q$ . We set  $k = 5$ , and the five nearest neighbors of  $p$  are  $a_i, i = 1, \dots, 5$ . Top: The dotted circle centered at  $q$  of radius  $d(p, q)$  includes no other element in  $\mathcal{EN}(p)$ , which is the stop criterion in the Delaunay graph search. For the  $k$ NN graph search, we relax this criterion by a factor of  $\mathcal{T}$ , which is indicated by the bold circle (centered on  $q$  and radius  $\mathcal{T}d(p, q)$ ). In this example,  $a_1$  and  $a_2$  are  $\mathcal{T}$ -closer to  $q$  with respect to  $p$ . Bottom: The neighbors of  $a_1$  and  $a_2$  are included in the extended neighborhood  $\mathcal{EN}(p, q)$ . Note that the elements  $a_3, a_4$ , and  $a_5$  are not  $\mathcal{T}$ -closer to  $q$  with respect to  $p$  and form the “boundary” of  $\mathcal{EN}(p, q)$ , which is denoted by dots with a boxed surround.

$\mathcal{EN}(p, q)$  if all nodes in its path to  $p$  are  $\mathcal{T}$ -closer to  $q$  with respect to  $p$ .

**Definition 3 Extended neighborhood:** An element  $p_n$  belongs to the extended neighborhood  $\mathcal{EN}(p, q)$  of  $p$  with respect to query  $q$ , if there exists a path  $p_0, p_1, \dots, p_n$ , where  $p = p_0$  and  $p_i$  is  $\mathcal{T}$ -closer to  $q$  with respect to  $p$  for all  $i = 1, \dots, n - 1$ .

The definitions of  $\mathcal{T}$ -closer and extended neighborhood allow us to use the following heuristic to search the  $k$ NN graph, in analogy to Proposition 1 for the Delaunay graph.

**Heuristic 1** Let  $S$  be the database and  $q$  be a query. Let  $p$  be a node in this graph, and let  $\mathcal{EN}(p, q)$  be the extended

neighborhood of  $p$ . If  $d(p, q) < d(a, q)$  for  $a \in \mathcal{EN}(p, q)$ , then  $p$  is the nearest neighbor of the query in the database, i.e.,  $d(p, q) < d(a, q)$  for all  $a \in S$ .

Note that the choice of  $\mathcal{T}$  determines how good the extended neighborhood approximation is. If a sufficiently large value of  $\mathcal{T}$  is used, the extended neighborhood will contain all the elements in the database, which leads to no saving over linear search. On the other hand, if  $\mathcal{T}$  is very small, the extended neighborhood will contain very few elements, and Heuristic 1 will not be true for some queries, and we will make erroneous choices for their nearest neighbor.

Our approach to search the  $k$ NN graph to find the nearest neighbor of a query is based on the above heuristic. We select a few well-separated elements in the database as “seeds”, from which the graph search starts. The seeds represent a coarse sampling of the database. From these seeds, the nearest neighbor search iteratively moves to neighboring elements in a best-first fashion, as long as Heuristic 1 holds. To that end, the front (the element that has the smallest distance to the query) is represented as a priority queue sorted by distance to the query. At each step, we pick the element at the front of the priority queue and attempt to move towards its neighbors that are  $\mathcal{T}$ -closer with respect to the query. The search stops when the element at the front of the priority queue is not  $\mathcal{T}$ -closer to the current nearest neighbor with respect to the query. Recall that in Euclidean spaces, when the Delaunay graph search stops, all the neighbors of the nearest neighbor to the query have been examined. The analogous region in the  $k$ NN graph for the best-first search is the extended neighborhood of the nearest neighbor. This implies that if Heuristic 1 is satisfied for the nearest neighbor, we will find the current nearest neighbor.

## 4 Results

In this section, we present the results of applying our algorithm to two databases, one is a synthetically generated database of points from a high-dimensional Euclidean space, and the other is a realistic database of shapes used for evaluating distance-based indexing techniques in Section 2. The first database is created by randomly sampling 20,000 points from a 50-dimensional Euclidean space. Another 200 points were used as queries. Since our approach is dependent on the choice of the  $\mathcal{T}$ ,  $k$  and the number of start points used, we examine the sensitivity of our approach to these parameters. The results, as summarized in Figure 5, indicate that our approach is relatively insensitive to the number of seed points and number of neighbors. Also, the indexing efficiency is good. The nearest neighbors for all queries are correctly found, while computing distances to about 30% of the elements in the database.

We now examine the performance of our approach for indexing into the database of shapes used in Section 2. Recall

Start Nodes	$\mathcal{T}$	Distance computations avoided			Errors
		Average	Max	Min	
9	1.40	172 (72%)	223	43	0
9	1.35	181 (75%)	223	61	0
<b>9</b>	<b>1.30</b>	<b>187 (78%)</b>	<b>223</b>	<b>107</b>	<b>0</b>
9	1.25	194 (81%)	223	114	0
12	1.40	175 (73%)	224	43	0
12	1.35	184 (77%)	224	61	0
12	1.30	191 (80%)	224	98	0
12	1.25	198 (83%)	224	114	1

**Table 2.** This table reports the indexing efficiency of the proposed for indexing into a shape database. The results are averaged over 144 queries. In all cases,  $k = 15$ . Observe that indexing efficiency is good, and the nearest neighbors for all queries are correctly identified by computing distances to less than 25% of the shapes in the database.

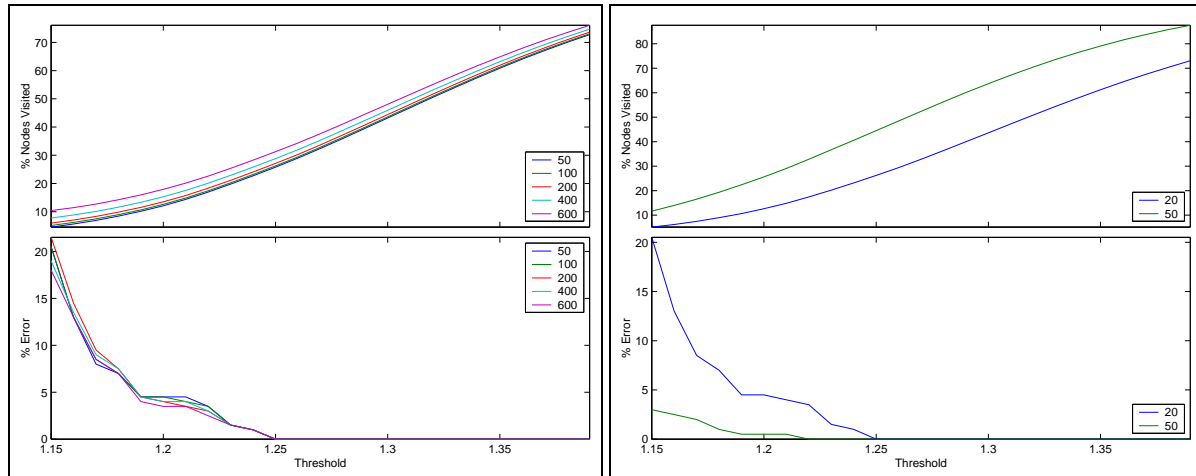
GH-Tree	GNAT	VP-Tree	AESA	$k$ NN graph
5%	20%	23%	39%	<b>78%</b>

**Table 3.** This table compares the indexing efficiency of the different methods for the shape database.

that our shape database consists of 384 shapes of which 240 form the core database and 144 are used as queries. Table 2 summarizes the performance of our approach for finding the nearest neighbor in the shape database for different values of threshold, and number of start-nodes. Observe that we are able to find the nearest neighbor correctly for all queries while avoiding 78% of distance computations which is a significant improvement over the best exact distance-based indexing method, namely AESA with 39%, Table 3.

## 5 Conclusion

In this paper, we have evaluated several distance-based indexing methods for shape retrieval. The need for improvements in indexing efficiency motivated the development of an approach based on a local approximation of the topology of the metric space, which led to a  $k$ NN graph. This graph is searched using a best-first strategy, starting from a few well-separated seeds. With this method, we obtained good indexing efficiency for a synthetic Euclidean database and a shape database. While this method represents a significant improvement over other distance-based indexing techniques for shape retrieval, it is not sufficient to make shape-based indexing practical. In the future, we plan to integrate the notion of categorization and prototypes, and coarse-to-fine scale matching with this proposed nearest neighbor search technique to develop a practical shape retrieval framework.



**Figure 5.** This figure examines the sensitivity of our approach to the number of seed points used in a range of 50 to 600 for  $k = 20$  (left) and the number of neighbors used ranging from 20 to 50 for 100 seed points (right). The percentage of distance computations required and the percentage of errors in finding the nearest neighbor in each case are plotted versus threshold  $\mathcal{T}$ . Observe that the indexing efficiency is good. Specifically, when using a threshold of 1.25, nearest neighbors are correctly found for all the queries while computing distances to about 30% of the elements in the database.

## References

- [1] A. Lanitis, C. J. Taylor, and T. F. Cootes. Automatic interpretation and coding of face images using flexible models. *PAMI*, 19(7):743–756, 1997.
- [2] R. Basri, L. Costa, D. Geiger, and D. Jacobs. Determining the similarity of deformable shapes. *Vision Research*, 38:2365–2385, 1998.
- [3] S. Belongie, J. Malik, and J. Puzicha. Matching shapes. *ICCV*, pages 454–461, 2001.
- [4] J. L. Bentley and J. H. Friedman. Data structures for range searching. *ACM Comp. Surveys*, 11(4):397–409, 1979.
- [5] S. Brin. Near neighbor search in large metric spaces. *VLDB*, pages 574–584, 1995.
- [6] W. Burkhard and R. Keller. Some approaches to best-match le searching. *Comm. ACM*, 16(4):230–236, 1973.
- [7] E. Chavez and G. Navarro. Unbalancing: The key to index high dimensional metric spaces. Technical report, Universidad Michoacana, 1999.
- [8] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petrkovic, and W. Equitz. Efficient and effective querying by image content. *J. Intell. Info. Systems*, 3:231–262, 1994.
- [9] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *ACM SIGMOD*, pages 163–174, 1995.
- [10] W. I. Groski and R. Mehrota. Index-based object recognition in pictorial data management. *CVGIP*, 52:416–436, 1990.
- [11] A. Guttman. R-tree: A dynamic index structure for spatial searching. *ACM SIGMOD*, pages 47–57, 1984.
- [12] G. R. Hjaltason and H. Samet. Contractive embedding methods for similarity searching in metric spaces. Technical Report TR-4102, CS Department, Univ. of Maryland, 2000.
- [13] J. B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Publications, Beverly Hills, CA, 1978.
- [14] C. Lin and R. Chellappa. Classification of partial 2-D shapes using Fourier descriptors. *PAMI*, 9(5):686–690, 1987.
- [15] G. Navarro. Searching in metric spaces by spatial approximation. *SPIRE*, pages 141–148, 1999.
- [16] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons Inc., 2nd edition, 2000.
- [17] E. Rivlin and I. Weiss. Local invariants for recognition. *PAMI*, 17(3):226–238, March 1995.
- [18] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.
- [19] T. B. Sebastian, P. N. Klein, and B. B. Kimia. Recognition of shapes by editing shock graphs. *ICCV*, pages 755–762, 2001.
- [20] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *IPL*, 40:175–179, 1991.
- [21] E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. *PRL*, 4:145–157, 1986.
- [22] R. Weber. Similarity search in high-dimensional data spaces. *Grundlagen von Datenbanken*, pages 138–142, 1998.
- [23] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. *SODA*, pages 311–321, 1993.