Chapter 6

Four new techniques for approximate similarity search in metric spaces

6.1 Introduction

In this chapter we discuss and propose new techniques for approximate similarity search. These approaches offer a high improvement of efficiency at the price of inprecision in the results. Our investigation focuses on approximate similarity search of data represented in metric spaces. As previously stated, this also includes the more specific case of vector spaces, thus our techniques can also be applied to them. We suppose that tree-based access methods for metric spaces, as for examples M-Trees [CPZ97] or Slim-Trees [TTSF00], are used to improve performance of similarity search queries. The techniques that we propose can be easily applied to these access methods by modifying their (exact) similarity search algorithms. Our results show that inprecision can be effectively controlled still guaranteeing high performance.

6.2 Overview of the approaches

We have investigated four new techniques for approximate similarity search. They are based on specific definitions for the approximate pruning or stop conditions (see Section 5.3.2). Specifically, two of these approximation techniques work for nearest neighbor queries and use a stop condition as approximation strategy. The other two techniques work both for nearest neighbor and range queries and use a pruning condition as approximation strategy.

First method The first technique of approximate similarity search uses a user defined parameter as an upper bound on the relative error between the distances of the query object from the objects retrieved by the exact algorithm and the distances of the query object from the objects retrieved by the approximate algorithm. The approximate similarity search algorithm decides which nodes of the tree can be pruned, even if their bounding regions overlap the query region, guaranteeing that the obtained relative error on distances does not go above the specified upper bound. This method was defined both for nearest neighbor and range queries and its details are discussed in Section 6.6.

Second method The second technique of approximation retrieves k approximate nearest neighbors of a query object by returning k objects that statistically belong to the set of the n ($n \geq k$) exact nearest neighbors of the query object. The value n is specified by the user as a fraction of the whole data set. The approximate similarity search algorithm stops when it determines, by using the overall distance distribution (see Section 2.3.4), that the current k retrieved objects belong to the specified fraction of the objects nearest to the query. This

method was defined for nearest neighbor queries only and it is discussed in Section 6.7.

Third method The third type of approximation is based on the pragmatic observation that similarity search algorithms on tree structures are defined as an iterative process where in each iteration the result, that is the set of objects retrieved, is improved until no further improvement is possible. In case of k nearest nearest neighbor queries, at each iteration the current k retrieved objects are nearer than those of the previous iteration. This can be made explicit measuring the distance between the current k-th object and the query objects. This distance decrease at each iteration and it can be noticed that it decreases rapidly in the first iterations then it slows down and remains almost stable for several iterations before the similarity search algorithm stops. The approximate similarity search algorithm exploits this behaviour and stops the search algorithm when the distance decreasing of the current k-th object seems to be stopping. In this case a user defined value is used as a threshold to decide when the distance decreasing can be considered to be stopping. This method was defined for nearest neighbor queries only and its description is given in Section 6.8.

Fourth method The fourth technique uses the proximity measure, discussed in Chapter 4, to decide which nodes of the tree can be pruned, even if their bounding regions overlap the query region, guaranteeing a low probability to loose qualifying objects. In fact, when the proximity of a region, bounding a tree's node, and the query region is small, the probability that qualifying objects are found in their intersection is small. A user specified parameter is used as a

Method	Range queries	NN queries	Approx. strategy
First	yes	yes	pruning condition
Second	no	yes	stop condition
Third	no	yes	stop condition
Fourth	yes	yes	pruning condition

Table 6.1: Characteristics of the approximation methods

threshold to decide if a node should be accessed or not. When the proximity is below the specified threshold the node is not accessed. This method was defined both for nearest neighbor and range queries and we discuss it in details in Section 6.9.

A summary of the main features of the proposed approximation methods is given in Table 6.1.

All four techniques of approximation were experimentally tested on real and synthetic data sets, and their efficiency was compared. The results obtained are encouraging, and efficiency improvement of even two orders of magnitude has been achieved. In other words, whereas a precise similarity search may take several minutes, in an approximated search this can be reduced to seconds, while the precision of approximation typically remains quite high. Consider as a preliminary illustrative example Figure 6.1. It presents search results for 10 nearest neighbors of query objects O_{q1} and O_{q2}, separately for the exact (NN) and the proximity based approximate algorithms (ANN) a proximity threshold of 0.01 in HV1 (see Section 2.5) data set. For each retrieved object, the object identifier (OID) and its distance from the query are reported. Objects that are simultaneously found by the exact and approximate algorithms are typed in bold. The last column reports the costs needed to execute the queries as the number of tree node reads. If we consider the first three objects

		OID Dist	OID Dist	OID Dist	OID Dist	OID Dist	OID Dist	OID Dist	OID Dist	OID Dist	OID Dist	Cost
O_{q1}	NN	7083	1561	7079	**2843**	8849	7077	6755	**7035**	2830	**257**	*1465*
		1298.2	1312.4	1329.35	1359.44	1362.76	1366.64	1376.8	1390.41	1404.95	1409.77	
	ANN	**2843**	**7035**	**257**	522	1206	839	567	1590	205	1069	*11*
		1359.44	1390.41	1409.77	1453.15	1517.73	1521.83	1549.45	1591.93	1631.66	1675.41	
O_{q2}	NN	3493	8623	1139	**3494**	4902	3954	**3504**	4010	8608	7255	*1503*
		1012.21	1032.24	1062.61	1063.53	1118	1131.31	1220.31	1233.08	1238.64	1250.13	
	ANN	**3494**	**3504**	636	1046	902	520	696	480	1286	1227	*15*
		1063.53	1220.31	1389.44	1389.88	1441.54	1495.74	1508.13	1597.27	1611.6	1615.37	

Figure 6.1: Comparison between the 10 nearest neighbors obtained by the precise and the proximity based approximate algorithms for two specific queries, using 0.01 as proximity threshold in the HV1 data set.

in the approximate response to O_{q1}, we can see that these objects are in the precise response on positions four, eight, and ten. However, the cost of the approximate algorithm is only 11 while that of the exact one is 1465. In a similar way, the first two approximate results of query O_{q2} correspond, respectively, to objects in positions four and seven in the exact response. The cost of the approximate algorithm is 15 while the exact search needs 1503 tree node reads, i.e. disk accesses.

In the following, we first outline the generic approximate similarity search algorithms for range and nearest neighbor queries used to implement the four approximation techniques (Section 6.3), we define the performance measures for objectively comparing the proposed techniques (Section 6.4), we describe the testing environment (Section 6.5), we give the details of the four approximation techniques and we discuss the obtained results (Sections 6.6, 6.7, 6.8, 6.9).

6.3 Generic approx. similarity search algorithms

In this thesis we propose four techniques for approximate similarity search. This section discusses the generic approximate range and nearest neighbors algorithms that can be used to support them. These algorithms are obtained by modifying Algorithms 3.3.1 and 3.3.2 for exact similarity search on tree-based access methods discussed in Section 3.3.2. Since we define techniques that can be used in generic metric spaces, we suppose that nodes, of the tree structure, are associated with ball regions (see Section 2.3.2), however the algorithms are still simplified and generic, not strictly related to any specific implementation. The pseudo-code of the algorithms for approximate similarity range queries and approximate nearest neighbors queries are respectively presented in Algorithms 6.3.1 and 6.3.2.

The difference between the four approximation techniques that we propose rely on the specific definition of the pruning or stop condition. Accordingly the approximation can be controlled by two different approximation parameters. The approximation parameter x_s is used by the stop condition strategy, while the approximation parameter x_p is used by the pruning condition strategy. Of course, the specific meaning of these two parameters and their use strictly depend on the specific techniques used to implement the stop or the pruning condition. The generic pruning condition $Prune(\mathcal{R}_q, \mathcal{R}_d, x_p)$ takes as arguments the query region $\mathcal{R}_q$, a data region $\mathcal{R}_d$ and the approximation parameter x_p. It returns true when the pruning strategy determines that the node covered by the data region can be pruned according to the approximation parameter x_p. The generic stop condition $Stop(RS_c, x_s)$ takes as arguments the current result set RS_c (the set of qualifying objects found up to the current iteration)

and the approximation parameter x_s. It returns true when the stop strategy determines that the current result set satisfies the approximation requirements, according to the approximation parameter x_s.

When the *Prune* function is defined as an overlap test and the *Stop* function is defined to be always false, the algorithms have the behaviour of the exact similarity search algorithms. In the other cases, it may happen that some nodes, which contain qualifying results, are pruned or that the algorithm is stopped before all results were found. Therefore, false dismissal can occur.

The thresholds x_s and x_p are used to tune the trade-off between the efficiency and effectiveness. Values corresponding to high performance give a less effective approximation, because more qualifying objects may be dismissed. Values that give very good approximations correspond to more expansive query execution since few node's accesses are discarded.

In all approximation techniques proposed, when x_s and x_p are set to zero, exact similarity searches are performed.

6.3.1 Generic approximate range search algorithm

Our approximate range search algorithms use only the pruning condition strategy, so they only depend on the x_p approximation parameter. The generic definition of the approximate range search algorithm is given in Algorithm 6.3.1. The algorithm takes as input values the query region, composed of the query object O_q and the query radius r_q, and the approximation parameter x_p. It returns the set of objects[1]

[1]The range search algorithm presented in Section 3.3.2 returns a set of pairs (p_j, O_j). Here for simplicity we omit the pointers p_j to the records.

Algorithm 6.3.1. **Range**
Input: query object O_q; query radius r_q; approximation parameter x_p.
Output: response set $\mathbf{range}^{x_p}(O_q, r_q)$.

1. Enter pointer to the root node into **PR**; empty $\mathbf{range}^{x_p}(O_q, r_q)$.
2. While $\mathbf{PR} \neq \emptyset$, do:
3. Extract entry N from **PR**.
4. Read N.
5. If N is a leaf node then:
6. For each $O_j \in N$ do:
7. If $d(O_q, O_j) \leq r_q$ then $O_j \rightarrow \mathbf{range}^{x}(O_q, r_q)$.
8. If N is an internal node:
9. For each child node N_c of N, bounded by region $\mathcal{B}_c(O_j, r_j)$ do:
10. If $\neg Prune(\mathcal{B}(O_q, r_q), \mathcal{B}_c(O_j, r_j), x_p)$ then insert pointer to N_c into **PR**.
11. End

$\mathbf{range}^{x_p}(O_q, r_q) = \{O_1^A, O_2^A, ..., O_{l_A}^A\}$ where O_i^A are the objects retrieved by the algorithm and l_A is the number of objects retrieved.

The main differences of the approximate range search algorithm with respect to the exact range search algorithm (see Algorithm 3.3.1) can be seen at Steps 8, 9 and 10. The exact range search algorithm, in fact, simply checks if the data region overlaps the query region. On the other hand, the approximate range search algorithm, if the extracted node is an internal node (Step 8), checks the approximate pruning condition. Each pointed node is inserted in **PR**, to be examined in the next iterations, if the pruning condition is false (Steps 9 and 10). Therefore, the approximate range search algorithm, depending on the definition of the approximate pruning condition, may discard some nodes even if they contain qualifying objects. Of course, a good pruning condition should suggest to discard nodes only when the chances that they contain

qualifying objects are very low.

6.3.2 Generic approximate nearest neighbors search algorithm

The approximate nearest neighbors algorithms may use both stop and pruning condition strategies so it depends on both x_s and x_p approximation parameters. The generic algorithm for approximate nearest neighbors search takes as input values the query object O_q, the number of neighbors required k, and the approximation parameters x_p and x_s. It returns the set of objects[2] $\mathbf{nearest}^{x_p,x_s}(O_q,k) = \{O_1^A, O_2^A, ..., O_k^A\}$ where O_i^A are the objects retrieved by the algorithm. The generic approximate nearest neighbors search algorithm is defined by Algorithm 6.3.2.

The main differences of the approximate nearest neighbors search algorithm with respect to the exact nearest neighbors search algorithm (see Algorithm 3.3.2) can be seen at Steps 4, 8 and 11 of Algorithm 6.3.2. At Steps 4 and 11 the approximate pruning condition is used instead of the simple overlap test, as discussed for the approximate similarity range search algorithm. In addition, at Step 8, the approximate stop condition is used: if the stop strategy determines that the approximate result set satisfies the approximation requirements, according to the x_s parameter, then the algorithm stops before the natural end, eventually missing other qualifying objects. The approximate stop condition, is accurate if it is able to stop the algorithm when the current result set is a good approximation of the exact result set.

[2] As we also said for the approximate range search algorithm, differently from the algorithms presented in Section 3.3.2, for simplicity we omit the pointers p_j to the real data.

Algorithm 6.3.2. **Nearest neighbors**
Input: query object O_q; number of neighbors k; approximation thresholds x_p and x_s.
Output: response set $\mathbf{nearest}^{x_p,x_s}(O_q,k)$.

1. Enter pointer to the root node into **PR**; fill $\mathbf{nearest}^{x_p,x_s}(O_q,k)$ with k (random) objects; determine r_q as the max. distance in $\mathbf{nearest}^{x_p,x_s}(O_q,k)$ from O_q.
2. While $\mathbf{PR} \neq \emptyset$, do:
3. Extract the first entry N from **PR**. Suppose that N is bounded by region $\mathcal{B}(O_i,r_i)$.
4. If $\neg Prune(\mathcal{B}(O_q,r_q),\mathcal{B}(O_i,r_i),x_p)$ then read N, go to 2 otherwise.
5. If N is a leaf node then:
6. For each $O_j \in N$ do:
7. If $d(O_q,O_j) < r_q$ then update $\mathbf{nearest}^{x_p,x_s}(O_q,k)$ by inserting O_j and removing the most distant from O_q; set r_q as the max. distance of objects in $\mathbf{nearest}^{x_p,x_s}(O_q,k)$ from O_q.
8. If $Stop(\mathbf{nearest}^{x_p,x_s}(O_q,k),x_s)$ exit.
9. If N is an internal node:
10. For each child node N_c of N bounded by region $\mathcal{B}_c(O_j,r_j)$ do:
11. If $\neg Prune(\mathcal{B}(O_q,r_q),\mathcal{B}_c(O_j,r_j),x_p)$ then insert pointer to N_c into **PR**.
12. Sort entries in **PR** with increasing distance to O_q.
13. End

6.4 Efficiency and accuracy measures

Assessing the quality of an approximate similarity search algorithm involves the measurement of the improvement of efficiency achieved and the accuracy of the approximate results.

Of course, the improvement of efficiency alone is not sufficient to assess the performance of the approximate similarity search algorithms, because of the natural tradeoff between the efficiency and effectiveness. In fact, high improvement of efficiency typically results in low accuracy of the approximate results. The performance

improvement by using approximate algorithms is obtained at the cost of degrading the quality of retrieval, because some of the objects of the exact query response set may not be included in the approximate one. A good approximate algorithm should be able to provide reasonable accuracy with a high improvement of efficiency with respect to the exact algorithm. Therefore, we also need some measurements to evaluate the quality of the approximate similarity search algorithms.

In the following we discuss these two aspects and we define the corresponding measures.

6.4.1 Efficiency

Approximate similarity search algorithms were introduced since some applications need responses to similarity query faster than what offered by exact similarity search algorithms. The *Improvement of Efficiency*, IE measures how much faster an approximate similarity search algorithm is than the corresponding exact similarity search algorithm. The improvement of efficiency is defined as

$$IE = \frac{cost(\mathbf{oper})}{cost(\mathbf{oper}^A)} \tag{6.4.1}$$

where *cost* is a function that gives the number of tree nodes accessed during query execution, **oper** is either $\mathbf{range}(O_q, r_q)$ or $\mathbf{nearest}(O_q, k)$, and $\mathbf{oper}^A$ corresponds to the approximate versions. Alternatively, *costs* could be defined as the number of distance computations performed during the query execution, but experiments demonstrate that the number of nodes accessed and the number of distance computations are linearly correlated.

6.4.2 Accuracy

We believe that, depending on the two types of similarity queries, nearest neighbor and range search, the quality of approximation should be evaluated from two different points of view:

Nearest neighbor queries: in the case of nearest neighbor queries, a useful information is the difference between the position of an object in the approximate result set and the position of the same object in the exact result set with respect to the query. This helps to judge how many objects were dismissed because of the approximation. For instance, when the first approximate nearest neighbor is the 20-th object in the exact ordering, it means that 19 objects were lost, by the approximation.

Range queries: in the approximate range search algorithms that we propose, false hits never occur so the approximate result set is always a subset of the exact result set. Therefore, it is useful to consider the percentage of objects of the exact result set that are also present in the approximate set. For instance, in the case when the exact result set contains 200 objects, while the approximated one contains 150, the percentage is $150/200 = 0.75$, that is 75% of the exact result set was retrieved.

According to the previous observations, we define and use in our experiments two measurements to assess the quality of approximation. The first one, called *error on the position*, EP, is used to determine the discrepancy between the position occupied by objects in the approximate and exact sets of nearest neighbors. The second measurement, called *number of exact results*, NE, is used to determine the percentage of

objects shared by the exact and approximate result sets of range queries.

In order to provide a more formal definition of these measurements, we need the following notation. Given a metric space $\mathcal{M} = (\mathcal{D}, d)$, we intend to support similarity search (range and nearest neighbor) queries on a set $\mathcal{DS} \subseteq \mathcal{D}$ of n elements. The exact range search retrieves the set $\mathbf{range}(O_q, r_q) = \{O_1, O_2, \ldots O_l\}$ with $l \leq n$, and $d(O_q, O_i) \leq d(O_q, O_j)$, for $i < j$. The approximate range search search algorithm retrieves the set $\mathbf{range}^{x_p}(O_q, r_q) = \{O_1^A, O_2^A, \ldots O_{l_A}^A\}$, where $l_A \leq l$ and $d(O_q, O_i^A) \geq d(O_q, O_i)$. In our approximate range search we have that $\mathbf{range}^{x_p}(O_q, r_q) \subseteq \mathbf{range}(O_q, r_q)$, since false hits never occur. The nearest neighbors search retrieves k objects $\mathbf{nearest}(O_q, k) = \{O_1, O_2, \ldots O_k\}$ with $d(O_q, O_i) \leq d(O_q, O_j)$, for $i < j$. The approximate nearest neighbors search algorithm also retrieves k objects, $\mathbf{nearest}^{x_p, x_s}(O_q, k) = \{O_1^A, O_2^A, \ldots O_k^A\}$, where $d(O_q, O_i^A) \geq d(O_q, O_i)$. We can now provide a definition of EP and NE.

EP: To compute EP we first consider EP_i as the position error of the i-th object O_i^A of the approximate set. It is computed as the difference between the position of the object in the exact result set and the approximated one, divided by the cardinality of the whole data set (otherwise, this measurement would depend on the size of the data set considered). It is easy to show that the position of object O_i^A in the exact result set $\mathbf{nearest}(O_q, k)$ is $\#(\mathbf{range}(O_q, d(O_q, O_i^A)))$. Therefore, the position error for the i-th approximate object is

$$EP_i = \frac{\#(\mathbf{range}(O_q, d(O_q, O_i^A))) - i}{\#(\mathcal{DS})}$$

The position error EP is obtained as the average of the position errors of all

objects in $\mathbf{nearest}^{x_p,x_s}(O_q, k)$

$$EP = \frac{\sum_{i=1}^{k} EP_i}{k} \tag{6.4.2}$$

NE: To compute NE we have to obtain the percentage of objects of the exact result set that are included in the approximate result set. This is simply the number of objects contained in $\mathbf{range}^{x_p}(O_q, r_q)$ divided by the number of objects contained in $\mathbf{range}(O_q, r_q)$

$$NE = \frac{\#(\mathbf{range}^{x_p}(O_q, r_q))}{\#(\mathbf{range}(O_q, r_q))} \tag{6.4.3}$$

It is worth noting that in other works [AMN+98] the measurement used was the *relative error on distances*. This gives the percentage of error on the distance from the query object of the approximate result with respect to the exact result. For the nearest neighbor search it is defined as

$$\epsilon = \frac{d(O_A, O_Q) - d(O_N, O_Q)}{d(O_N, O_Q)} = \frac{d(O_A, O_Q)}{d(O_N, O_Q)} - 1$$

where O_A is the approximate nearest neighbor, O_N is the real nearest neighbor, and O_Q is the query object. This measurement can be easily generalized to the case of k-nearest neighbors search and range queries.

However we believe this measurement not to be suitable to objectively asses the quality of an approximate similarity search algorithm for three reasons:

i) It does not give an idea of the number of object missed.

ii) In case of high dimensional vector spaces it tends to give good results in any case.

iii) Results obtained using different data sets might not be directly compared.

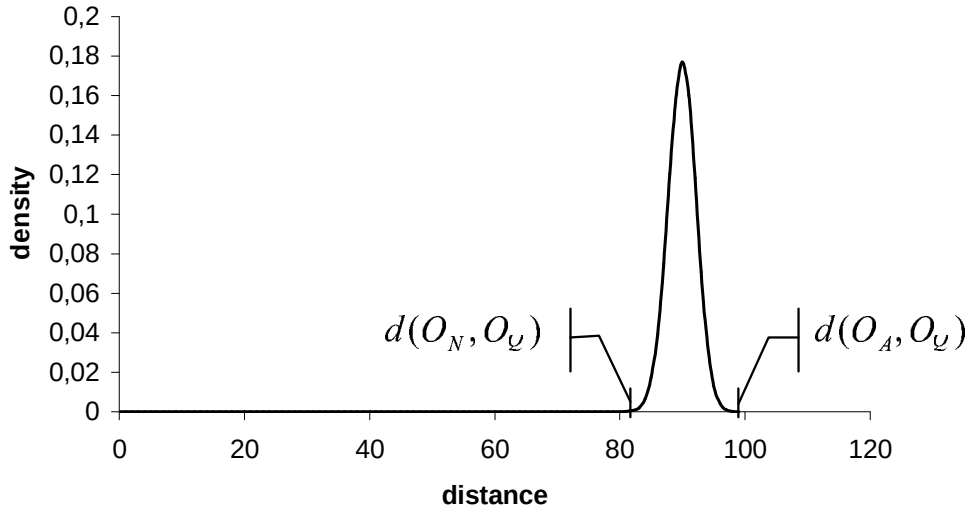


Figure 6.2: The relative distance error is not a reliable measure of the approximation accuracy. Even though the relative distance error is small, almost all objects are missed by the approximate search algorithm.

Let us discuss these three points. i) the relative error on distances gives the percentage of error on the distance of the approximate result with respect to the exact result. However, it does not give any suggestion on the amount of objects missed by the approximate algorithm, since it does not take into account distribution of distances. Let suppose that the distance between the first and the second nearest neighbor is large. Let also suppose that the approximate search algorithm misses the first nearest neighbor and returns the second. In this case the relative error on distances is high even if just one object was missed. Vice versa, suppose that relative error on distance is small, but distances of the approximate nearest neighbor O_A and the exact nearest neigbor O_N from O_Q are such that almost all remaining objects are included in between, as depicted in Figure 6.2. In this case the small relative error is misleading since all objects are in fact missed. ii) in high dimensional

vector spaces all objects tend to have similar distances. It is easy to show that as a consequence of the central limit theorem [HPS71], the peak of the distance density tends to move toward high values of distances as the number of dimensions increases. For illustration consider again Figure 6.2. The consequence is that the relative error on distances tend to be smaller anyway, since the farthest object is always relatively close to the nearest object. iii) the relative error on distances does not take into account the range of distances of the specific data set considered and its distance distribution. A particular relative error, considered to be large in a certain data set, might be considered negligible in a data set where the range of distances is much larger, or distances are distributed differently.

6.5 Experimentation settings

We have implemented the proposed approximate similarity search algorithms in the M-tree [CPZ97] framework. The approximate similarity search algorithms replaced the original search algorithms of an existing C++ implementation of M-Tree – the original M-tree code is available from http://www-db.deis.unibo.it/research/Mtree/.

Data sets UV HV1 and HV2, described in Section 2.5, were used as the test-bed. We executed similarity range queries with radii such that the smallest and the largest radii retrieved approximately between 1% and 20% of objects in the data sets. Nearest neighbors queries were executed varying k between 1 and 50.

For every obtained configuration, various values of x_s and x_p were tested. The used values depend on each specific approximation strategy and are specified later, when the strategies themselves are defined. In order to reduce statistical errors, each approximation parameter was tested for 50 different query objects, using the same

radius in case of range queries, or the same k in case of nearest neighbor queries. Query objects were not part of the data set, but followed the same data distribution.

Average values were computed for NE, EP, and the improvement of efficiency IE. For range queries we provide graphs where we show how IE depends on the approximation parameters, how IE depends on NE and how NE depends on the approximation parameters. For nearest neighbor queries we provide graphs where we show how IE depends on the approximation parameters, how IE depends on EP and how EP depends on the approximation parameters.

6.6 Method 1: Approximate similarity search using relative error of distances

To improve efficiency of similarity search algorithms, access methods for metric spaces partition the searched data and bound such elements of the partition by ball regions. Regions are then hierarchically stored in nodes of a tree. Search algorithms find qualifying objects by only accessing nodes corresponding to regions that overlap the query region. In fact, regions that overlap the query region may potentially contain objects that are also covered by the query regions, while regions that do not overlap the query region can be discarded.

Given a query region $\mathcal{B}(O_q, r_q)$ and a data region $\mathcal{B}(O_i, r_i)$, the simplest overlap test, to decide if the data region (the corresponding node of the tree) should be accessed, is to check if the distance between the centers of the regions is smaller or equal than the sum of their radii as follows

$$d(O_q, O_i) \leq r_q + r_i. \tag{6.6.1}$$

In M-trees [CPZ97], as discussed in Section 3.4.4, this test is further improved. Let $\mathcal{B}(O_p, r_p)$ be the parent region of $\mathcal{B}(O_i, r_i)$, so that the region $\mathcal{B}(O_p, r_p)$ covers $\mathcal{B}(O_i, r_i)$, and the node corresponding to $\mathcal{B}(O_p, r_p)$ is accessed before the node corresponding to $\mathcal{B}(O_i, r_i)$. The overlap test in Equation 6.6.1 is not needed to be performed, and the data region can be immediately pruned, when the difference between the distance of O_q from O_p and the distance of O_i from O_p is greater than the sum of the radii r_q and r_i, that is when

$$|d(O_q, O_p) - d(O_j, O_p)| > r_q + r_i. \tag{6.6.2}$$

Note that, in M-trees it is possible to store in the node the distance $d(O_j, O_p)$ of the object O_p from each entry (from the centers O_i of the corresponding covered regions). Moreover, the distance $d(O_q, O_p)$ between the parent object O_p and the query object O_q is computed when the node is accessed, before accessing its entries. Therefore, previous test in Equation 6.6.2 allows the search algorithm, in some cases, to discard a region without even computing the distance between its center and the query object $d(O_q, O_i)$.

These two pruning tests can be conveniently modified to obtain approximate similarity search algorithms, based on an approximation in the pruning condition, where the result set is constrained by a user-defined relative error of distances ϵ. The idea to obtain this is the following.

Let O_N be the nearest neighbor of O_q, and O_A some other object in the searched

collection. Obviously, provided $0 < d(O_N, O_q) \leq d(O_A, O_q)$, the equation

$$\frac{d(O_A, O_q)}{d(O_N, O_q)} = 1 + \epsilon$$

defines that the distance from O_q to O_A is $(1 + \epsilon)$ times the distance from O_q to O_N. Now assume that O_A is the *approximated* nearest neighbor of O_q. In such case, ϵ represents the *relative error* of the distance approximation, obtained considering O_A as the nearest neighbor of O_q instead of O_N.

Using the relative error of the distance approximation ϵ as an upper bound we can define an object O_A to be an *(1+ϵ)-approximate-nearest-neighbor* [AMN+98] of object O_q, if its distance from O_q is within a factor of $(1 + \epsilon)$ of the distance of the true nearest neighbor O_N, that is when

$$\frac{d(O_A, O_q)}{d(O_N, O_q)} \leq 1 + \epsilon.$$

This idea can be generalized to the case of the j-th nearest neighbor of O_q, for $1 \leq j \leq n$, where n is the size of the database. Using respectively O_A^j and O_N^j to designate, the j-th approximate and the nearest neighbors, the constraint should be modified as follows

$$\frac{d(O_A^j, O_q)}{d(O_N^j, O_q)} \leq 1 + \epsilon.$$

If this constraint is satisfied, O_A^j is called the *(1+ϵ)-j-approximate nearest neighbor* of O_q. However, even if $d(O_q, O_N^j)$ is unique for a given O_q, there may be several objects in the database which, when considered as O_A^j, satisfy the previous equation. This means that the candidate set for approximate results is not necessarily singular. In the limit case, $d(Q, O_A^j) = d(Q, O_N^j)$ or even $O_A^j = O_N^j$.

To see how the search pruning tests of M-trees can be relaxed by respecting a

relative distance error ϵ, let us consider another form of Equations 6.6.2 and 6.6.1 as follows:

$$\frac{r_q}{d(O_i, O_q) - r_i} < 1 \tag{6.6.3}$$

and

$$\begin{cases} \frac{r_q}{|d(O_p,O_q)-d(O_i,O_p)|-r_i} < 1 & \text{if } \mid d(O_p, O_q) - d(O_i, O_p) \mid -r_i > 0 \\ false & \text{elsewhere} \end{cases} \tag{6.6.4}$$

Looking at these fractions, the numerators specify, the distance to the k-th nearest neighbor of O_q discovered so far, in case of k nearest neighbors search, or the maximum accepted distance from O_q, in case of range search. In case of k nearest neighbors search, provided the search has not finished, this distance can also be considered as the distance to the current approximate neighbor. The denominators, on the other hand, put the *lower bounds* (using the corresponding information about distances at hand) on possible nearest neighbors in the region $\mathcal{B}(O_i, r_i)$ with respect to O_q. In other words, the denominators represent the minimum distance that an object in the given region might have, with respect to O_q. Naturally, if the lower bounds (i.e. the denominators) are higher than the current radius of O_q, the region considered cannot contain any qualifying object and therefore can be ignored in the search from this point on.

In order to modify these tests to the case of approximate search, that is when $\epsilon > 0$, the lower bounds can be relaxed by the relative factor ϵ in the following way.

$$\frac{r_q}{d(O_i, O_q) - r_i} < 1 + \epsilon \tag{6.6.5}$$

and

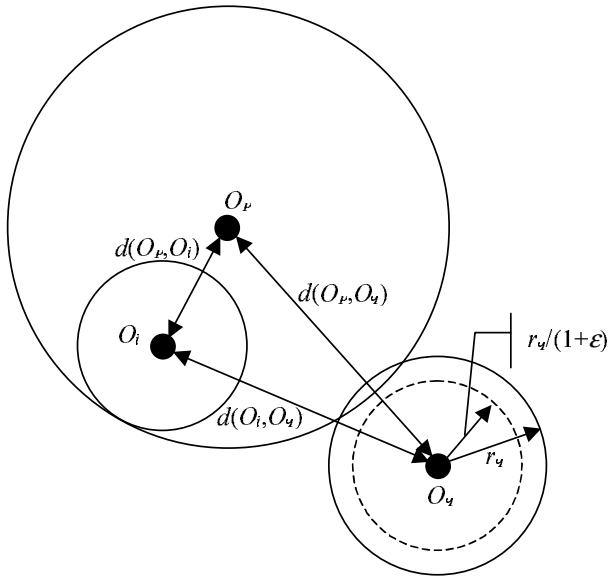


Figure 6.3: The region $\mathcal{B}(O_i, r_i)$, its parent region $\mathcal{B}(O_p, r_p)$, the query region $\mathcal{B}(O_q, r_q)$, and the reduced query region $\mathcal{B}(O_q, r_q/(1+\epsilon))$

$$\begin{cases} \frac{r_q}{|d(O_p,O_q)-d(O_i,O_p)|-r_i} < 1+\epsilon & \text{if } \mid d(O_p,O_q) - d(O_i,O_p) \mid -r_i > 0 \\ false & \text{elsewhere} \end{cases} \tag{6.6.6}$$

Naturally, relaxing these tests in the above way can never increase the similarity search costs, because both the number of distance computations and the number of node reads can only be reduced. In fact, relaxing the pruning tests corresponds to use a smaller query region $\mathcal{B}(O_q, r_q/(1+\epsilon))$, instead of the original query region, as shown in Figure 6.3.

Let us call $\epsilon Prune(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i), \epsilon)$ the approximate pruning test defined in Equation 6.6.5 and $\epsilon PrePrune(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i), \epsilon)$ the one defined in Equation 6.6.6. The pruning condition of Algorithms 6.3.1 and 6.3.2 is defined as follows:

$$Prune(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i), x_p) = \text{ if } \epsilon PrePrune(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i), x_p)$$
$$\text{return } true$$
$$\text{else}$$
$$\text{return } \epsilon Prune(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i), x_p)$$

This method uses only the pruning condition, therefore the stop condition is always false:

$$Stop(RS_c, x_s) = false$$

We have tested this method both with range and nearest neighbor queries. The obtained results are discussed in the next section.

6.6.1 Results

Results of the tests performed by using the relative distance error based approximate similarity search algorithms are sketched in Figures 6.4, 6.5, and 6.6 for range queries and in Figures 6.7, 6.8, and 6.9 for nearest neighbor queries.

Using this approximation technique, the threshold x_p is interpreted as the upper bound on the relative error of the distances in the approximate result set with respect to the exact result set. In these experiments we ranged the threshold in the interval between 0 and 8. The approximation threshold x_s for the stop condition was not used, since the stop condition is always false.

A first general observation on the obtained results is that no high improvement of efficiency is obtained with this method. The approach tends to saturate before than valuable improvement of efficiency is obtained. In fact, even if high values of

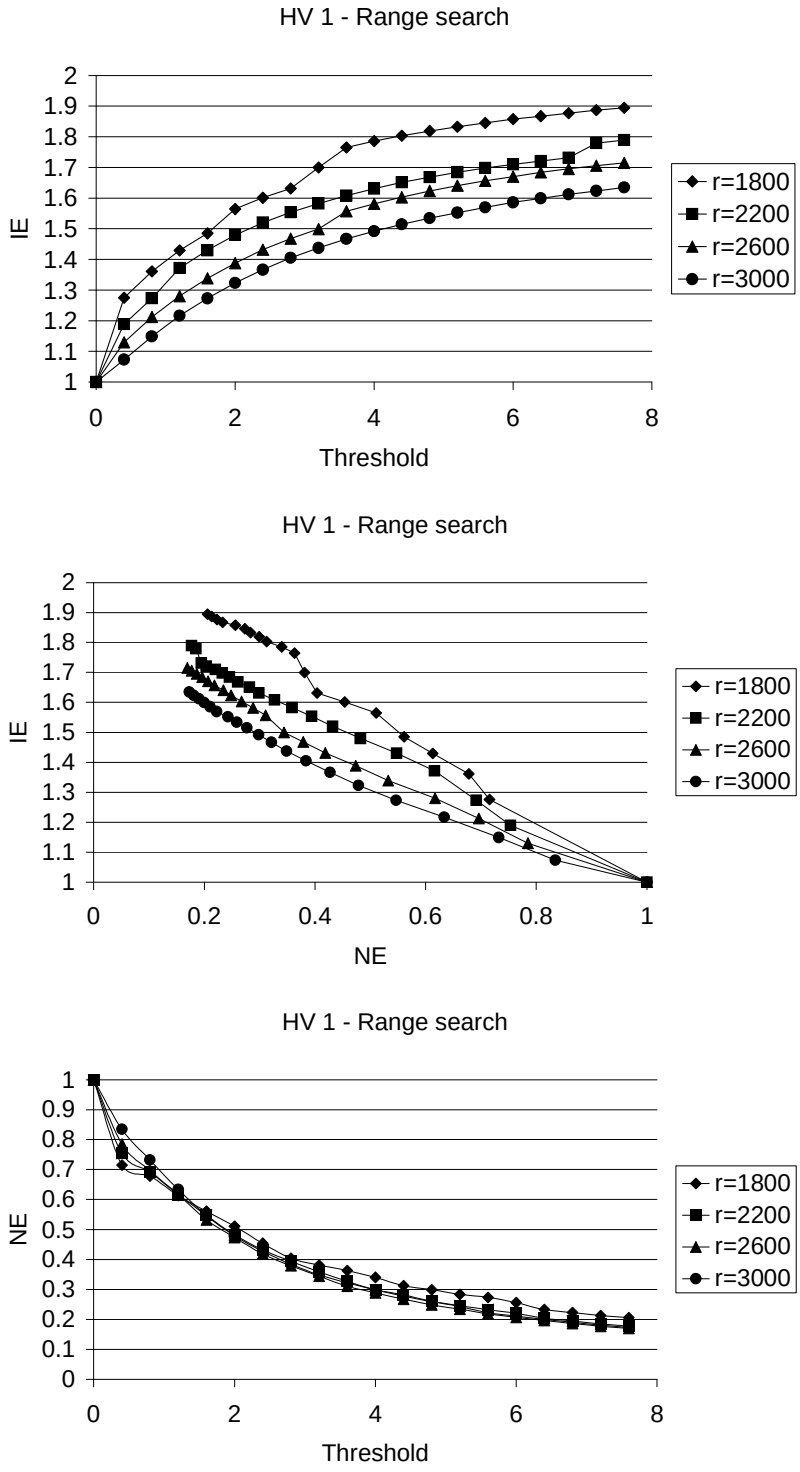
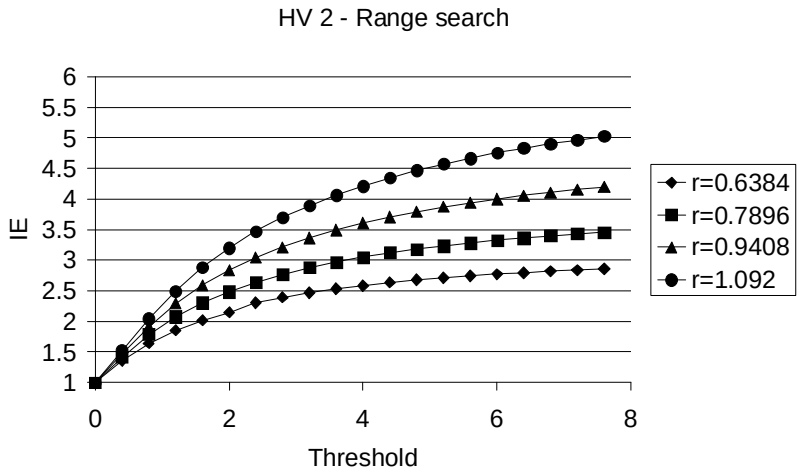
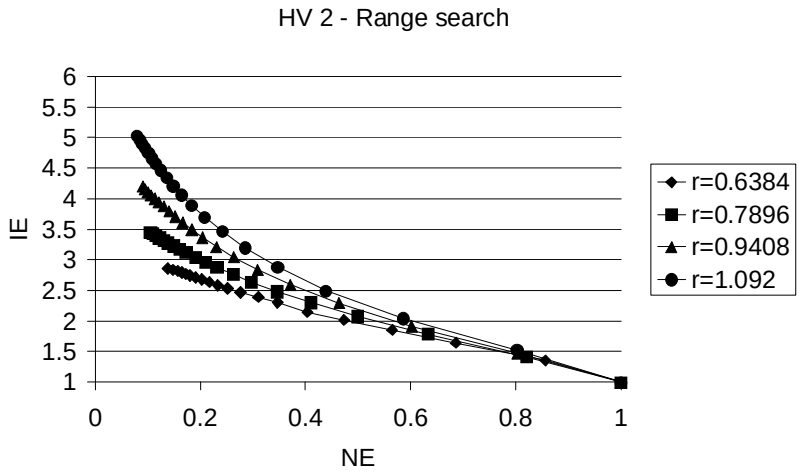


Figure 6.4: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the fraction of exact results (NE). Range queries, HV1 data set.





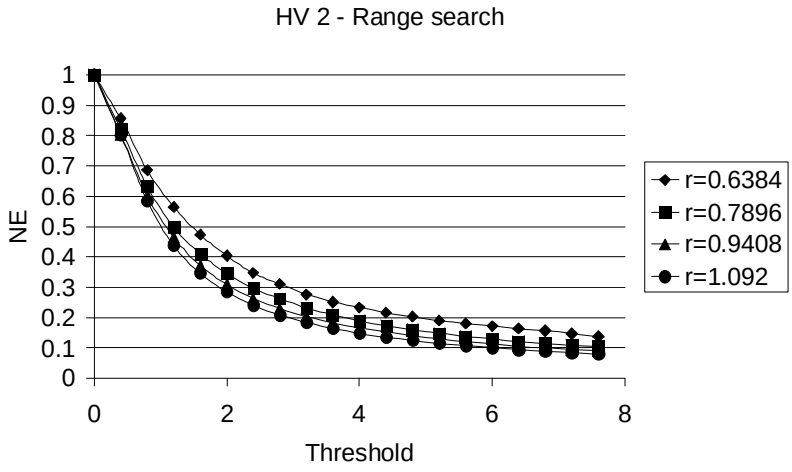


Figure 6.5: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the fraction of exact results (NE). Range queries, HV2 data set.

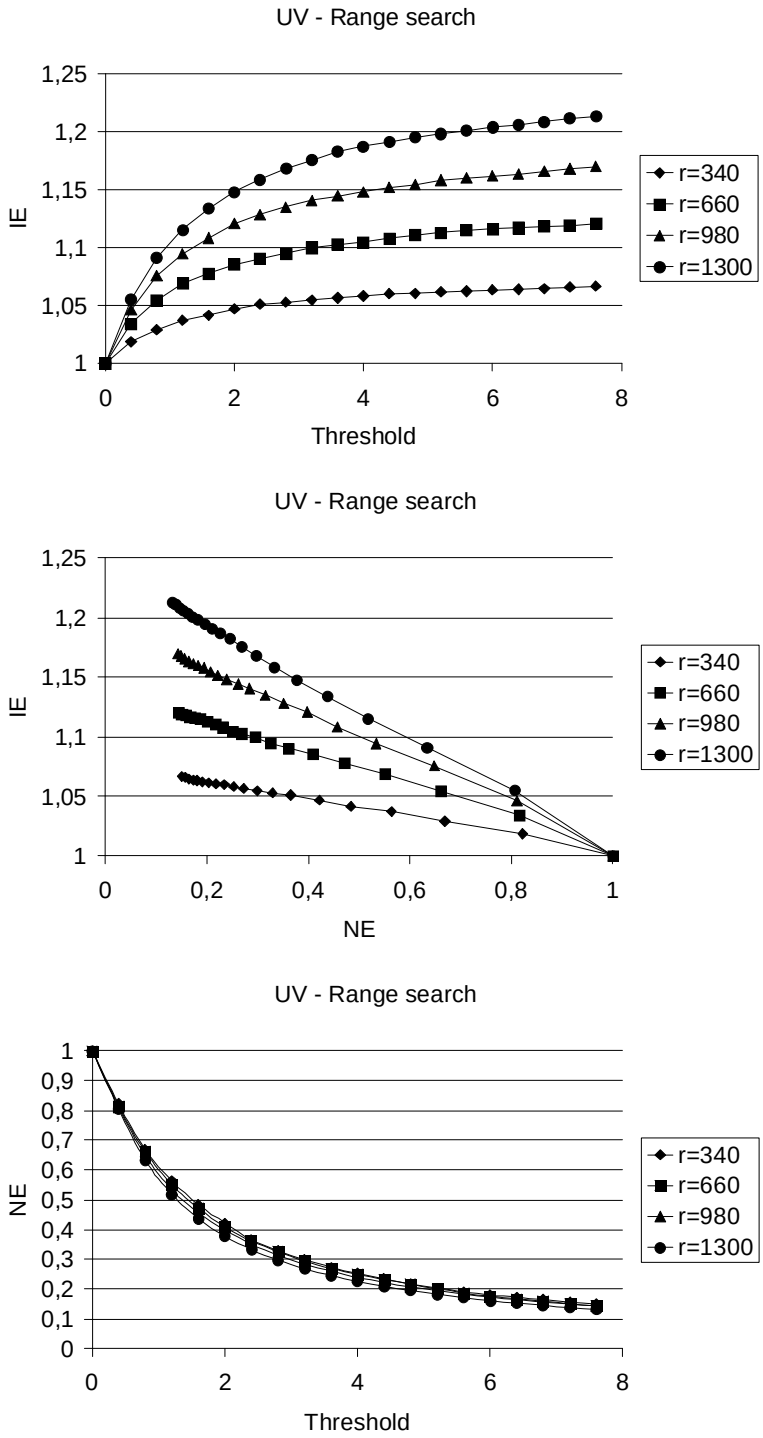
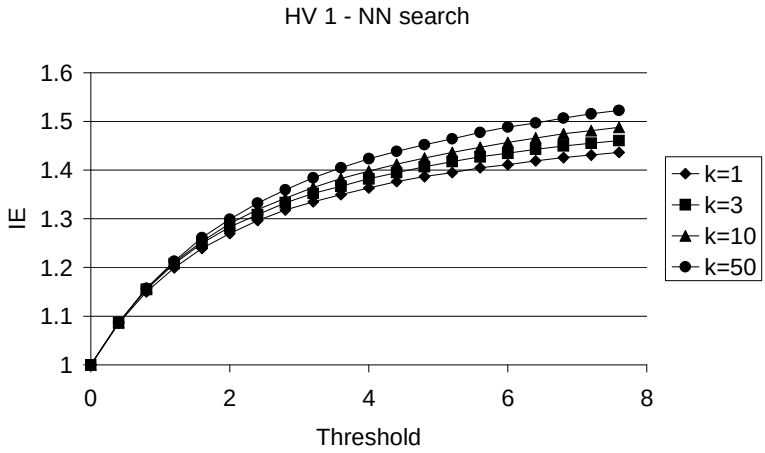
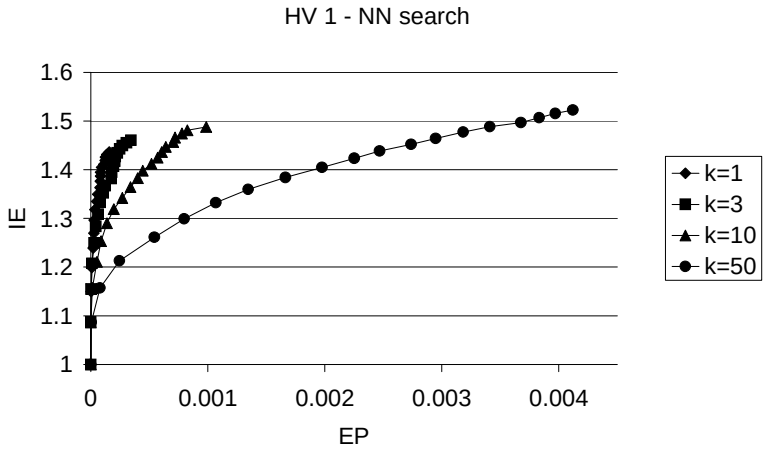


Figure 6.6: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the fraction of exact results (NE). Range queries, UV data set.





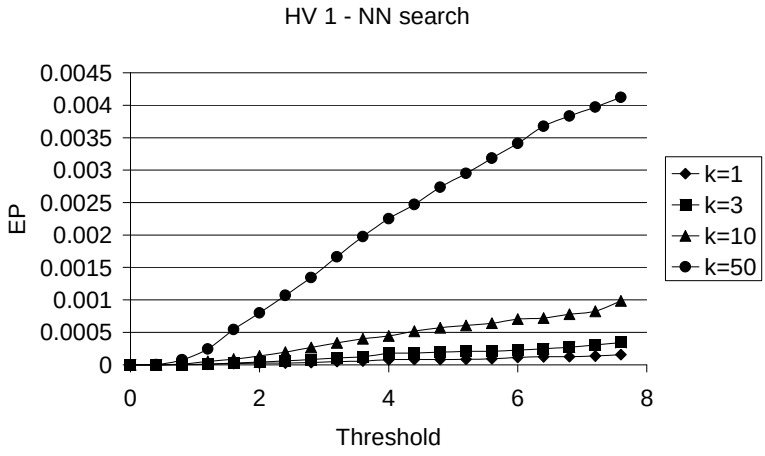


Figure 6.7: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the position error (EP). Nearest neighbor queries, HV1 data set.

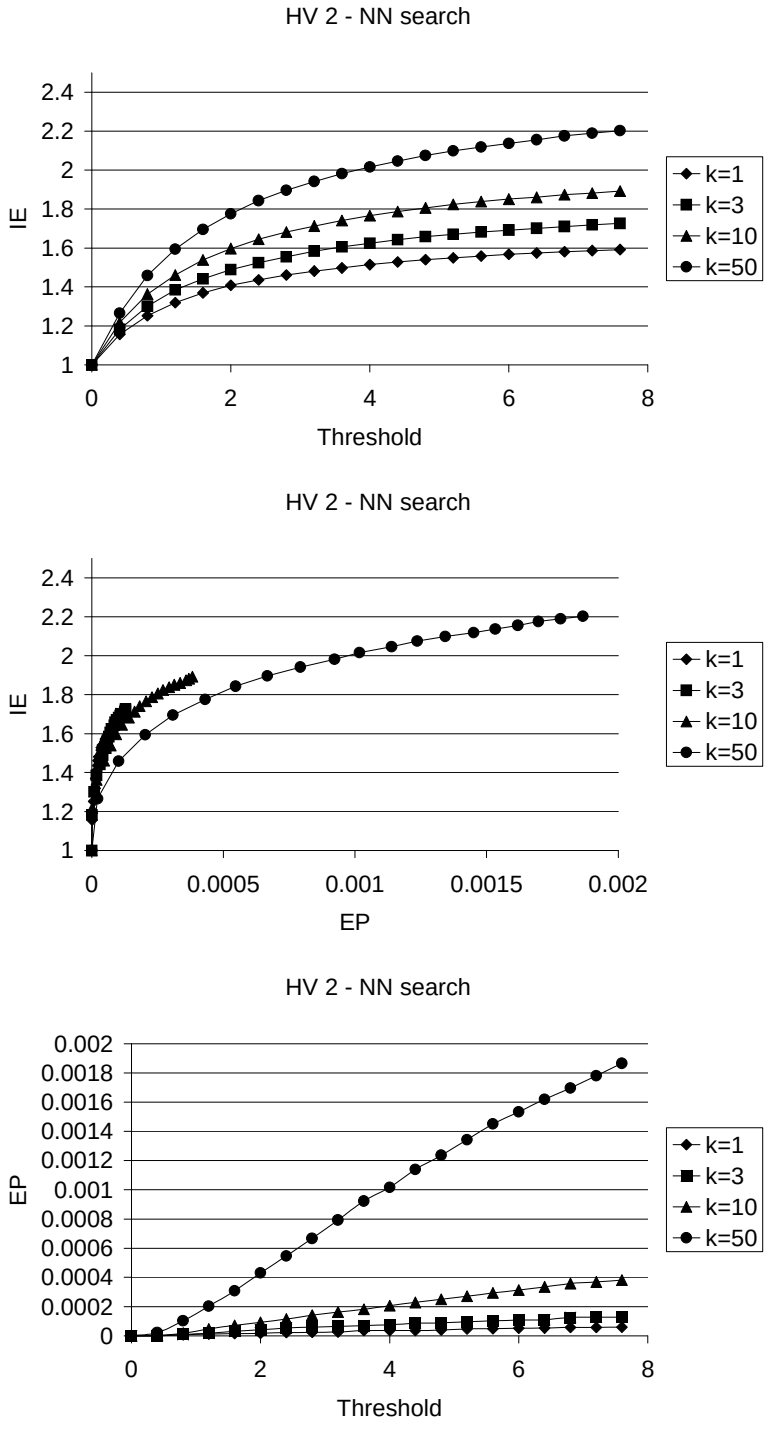


Figure 6.8: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the position error (EP). Nearest neighbor queries, HV2 data set.

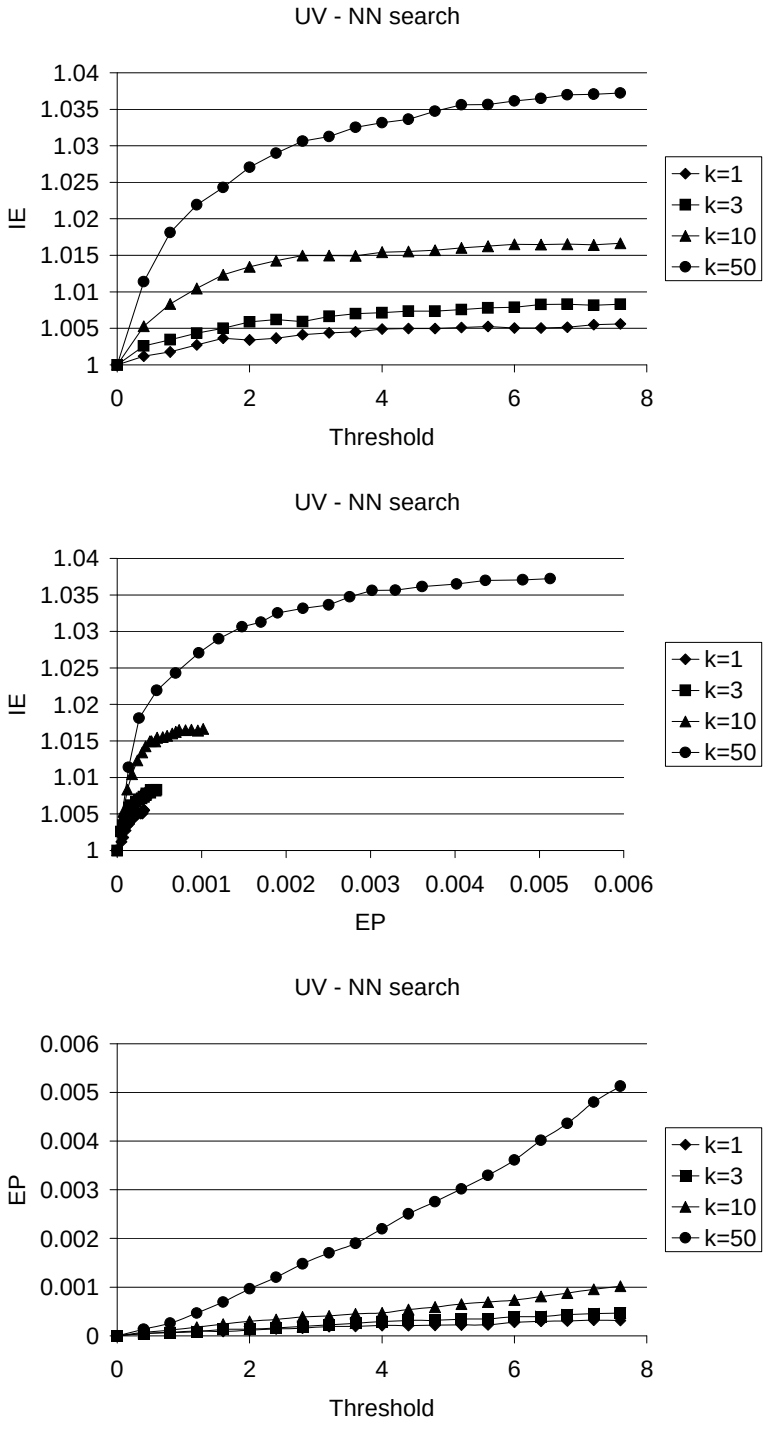


Figure 6.9: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the position error (EP). Nearest neighbor queries, UV data set.

the threshold are used, no considerable improvement is achieved. The graphs show that the best results where obtained in the HV2 data set, with range queries, where the approximate range search was executed only five time faster than the exact range search.

Another observation is that there is not evidence of a common trend in the different data sets that we have used. In fact, in case of range queries, the graphs relating NE with IE show that in HV1 performance degrades when the radius of the query increases, while in HV2 and UV performance degrades when the radius decreases. On the other hand, in case of nearest neighbors queries, the graphs relating EP with IE show that in HV1 and HV2 performance degrades when k increases, while in UV performance degrades when k decreases. This means that in order to tune the algorithms, to optimize performance, a deep knowledge of the used data set should be obtained, and no general guidelines can be exploited.

Let us consider specifically the behavior of range queries. In HV1, the best results where obtained when the radius was small. For instance, when the query radius was 1800, a query was executed on average 1.9 times faster with $NE = 0.2$, that is 20% of objects retrieved by the exact search were found by the approximate search. In HV2, the best results were obtained when the radius was large. In fact, when the query radius was 1.092, improvement of efficiency was 3.7 with a value of the the NE measure of 0.2. In case of the UV data set, again, the best results were obtained when the radius was large. In fact, when the query radius was 1300, improvement of efficiency was 1.2 and the NE measure was 0.2.

Let us now consider results of nearest neighbors queries. In HV1, the best results where obtained when k was small. In fact, when k was 1, queries were executed

on average 1.43 times faster, with $EP = 0.0001$, that is, since HV1 contains 10000 objects, the approximate nearest neighbor was on average in the 2-nd position. In HV2, again the best results were obtained when k was small. In fact, with k set to 1, improvement of efficiency was 1.6 with a value of the the EP measure of 0.00005, that is, in almost all cases the real nearest neighbor was found. Finally, in case of the UV data set the best results were obtained when k was large. In fact, with k set to 50, improvement of efficiency was 1.027 and the EP measure was 0.001.

Notice the dependence between the threshold and the NE measure in Figures 6.4, 6.5, and 6.6. NE can be directly controlled by the approximation parameter, independently of the radius of the query. However this property cannot be observed for the nearest neighbors query. In fact, it can be noticed in figures 6.7, 6.8, and 6.9, that, with the same value of x_p, when k is set to 50 the error on the position is on average much higher than using smaller values for k.

6.7 Method 2: Approximate similarity search using distance distribution

As discussed in Section 6.3.2, the k-nearest neighbors search algorithm obtains the final result set by improving iteration after iteration a current result set, say RS_c. In fact, in every iteration, if a new object O is found whose distance from the query object O_q is shorter than that of some of the objects in RS_c, then the k-th object O_k in RS_c is removed and O is inserted in RS_c.

The idea at the basis of the approximate similarity search technique discussed in this section is to stop the search algorithm when the objects of RS_c belong to a user

specified fraction of the closest object to the query object O_q. As an example, let us suppose that the considered data set $\mathcal{DS}$ contains 10000 objects and that objects O_1, O_2,...,$O_{10000} \in \mathcal{DS}$ are ordered with respect to their distance from the query objects O_q. If the user chooses 1/200 (that is 0.5%) as the wanted fraction, then the algorithm should stop as soon as all objects in RS_c belong to the set $\{O_1, O_2, ..., O_{50}\}$ of the 50 closest objects to O_q (since $10000/200 = 50$).

The previous idea can be realized by using a probabilistic approach and exploiting the overall distance distribution (see Section 2.3.4) of the objects of the considered metric space. Let suppose that we have a metric space $\mathcal{M} = (\mathcal{D}, d)$. The distribution function $F_{O_i}(x)$ (the distance distribution relative to O_i, or the O_i's *viewpoint*) gives the probability that chosen a random object $\mathbf{O}$ from $\mathcal{D}$, its distance from O_i is smaller than x, that is $F_{O_i}(x) = \mathtt{Pr}\{d(O_i, \mathbf{O}) \leq x\}$.

Let us suppose that our data set $\mathcal{DS} \subseteq \mathcal{D}$ contains a sample of n objects of $\mathcal{D}$ selected in such a way that for any O_i, belonging to $\mathcal{DS}$, its distance distribution F_{O_i}, computed considering the objects of $\mathcal{DS}$ only, is the same than that computed considering all objects of $\mathcal{D}$[3]. According to this, $F_{O_i}(x)$ represents the fraction of objects in $\mathcal{DS}$ for which the distance to O_i is smaller than or equal to x. In fact, since the number of objects in $\mathcal{DS}$ is n, then the expectation is that $n \cdot F_{O_i}(x)$ objects should have a distance to O_i not greater than x.

Let us now consider the k-nearest neighbor search algorithm and the current result set RS_c obtained at a certain intermediate iteration. Let O_c^k be the current k-th object in RS_c and $d(O_q, O_c^k)$ its distance from O_q. As a consequence of previous

[3]This can be generally considered true when the number of objects of the data set is large. In these cases we can suppose that the data set characterizes the entire metric space which it belongs to (see Section 2.3.4).

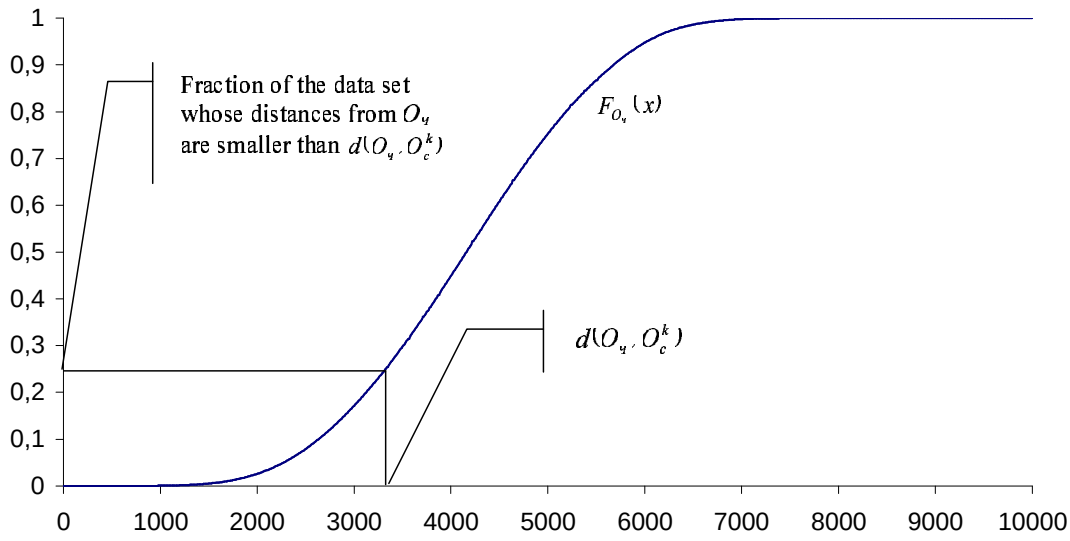


Figure 6.10: An estimation of the fraction of the objects closest to O_q, whose distances from O_q are smaller than $d(O_q, O_c^k)$, can be obtained by using $F_{O_q}(x)$.

discussions, we have that $F_{O_q}(d(O_q, O_c^k))$ corresponds to the fraction of objects in $\mathcal{DS}$ whose distances from O_q are smaller than or equal to $d(O_q, O_c^k)$, see Figure 6.10. Since all other objects in RS_c have a distance from O_q smaller or equal than $d(O_q, O_c^k)$ (since O_c^k is the k-th object), then all objects in RS_c are included in that fraction. For instance, when $F_{O_q}(d(O_q, O_c^k)) = 1/200$, RS_c is expected to be included in the set corresponding to the 0.5% of the objects closest to O_q.

This behavior can be exploited to obtain an approximate nearest neighbor search based on a stop condition. In fact, here the user can specify the value of the approximation parameter x_s corresponding to the wanted fraction, and the stop condition is defined as follows:

$$Stop(RS_c, x_s) = F_{O_q}(d(O_q, O_c^k)) \leq x_s$$

where O_c^k is the k-th object in RS_c.

Of course, since the stop condition may only stop the algorithm 6.3.2 earlier, it might only reduce the cost with respect to the exact similarity search. However, when $x_s < F_{O_q}(d(O_q, O_N^k))$, where O_N^k is the exact k-th nearest neighbor, no improvement is obtained and the exact similarity search is performed. In particular, when $x_s = 0$ the exact similarity search is always performed.

So far, we have assumed that the distribution function F_{O_q} is known. However computing and maintaining this information for any possible query object is totally unrealistic – query objects are not known a priori. A solution is to use, instead of F_{O_q}, the overall distance distribution defined as $F(x) = \mathtt{Pr}\{d(\mathbf{O_1}, \mathbf{O_2}) \leq x\}$, where $\mathbf{O_1}$ and $\mathbf{O_2}$ are random objects of $\mathcal{DS}$. In fact, as discussed in Section 2.3.4, F can be reliably used in these cases as a substitute of any F_{O_q}, given the high index of homogeneity of viewpoints in typical data sets. Furthermore, obtaining F does not present hard problems since it should be computed only once in $O(n^2)$, where n is the size of the considered data set.

The resulting stop condition uses the overall distance distribution F and is consequently defined as follows:

$$Stop(RS_c, x_s) = F(d(O_q, O_c^k)) \leq x_s$$

The pruning condition performs the usual exact overlap test, since this approximation method is only based on a stop condition:

$$Prune(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i), x_p) = d(O_q, O_i) > r_q + r_i.$$

where, of course, the parameter x_p is not used.

6.7.1 Results

Results of the tests performed using the approximation method that exploits distance distribution are shown in Figures 6.11, 6.12 and 6.13.

The approximation threshold x_s specified by the user, is interpreted as the fraction of the objects closest to the query which the current result set should belong to when the algorithm is stopped. Here the approximation threshold ranges in the interval between 0 and 0.07 (that is a fraction corresponding to 7% of the objects closest to the query). The approximation threshold x_p for the pruning condition is not used, as we said previously.

A general observation is that the approximate algorithm degrades its performance when the number of objects retrieved increases. In fact when k is set to one, the improvement of efficiency almost arrives up to 800 (that is almost three orders of magnitude), while when k is set to 50 the improvement of efficiency arrives up to 45.

Very good results were obtained with HV1 and HV2, even though HV1 gave the best performance. With these two data sets, on average, improvement of efficiency up to 2 orders of magnitude was obtained, still maintaining good quality search results. However, in UV was difficult to obtain improvements comparable with the other two data sets. For instance, in HV1 the approximate algorithm can find the nearest neighbor 423 times faster with $EP = 0.004$. This means that, if the exact algorithm needs 7 minutes to find the nearest neighbor, the approximate algorithm needs only one second to find an approximate nearest neighbor that is, on average, the 40-th (out of 10000) actual nearest neighbor. In HV2, the approximate nearest neighbor can be found on average 100 times faster with an error on the position of 0.004. On the other hand, in the UV data set the nearest neighbor can be found only 45 times

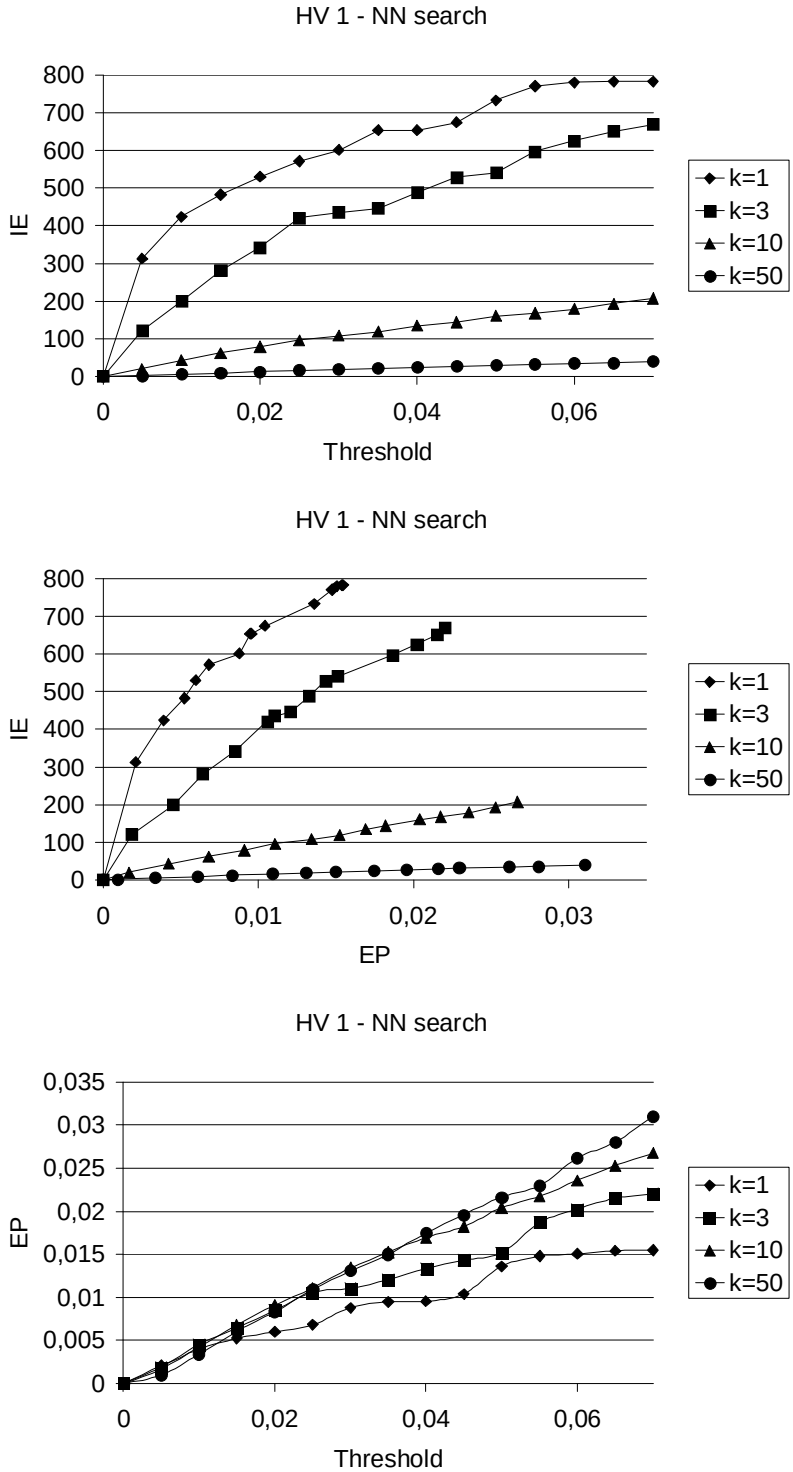
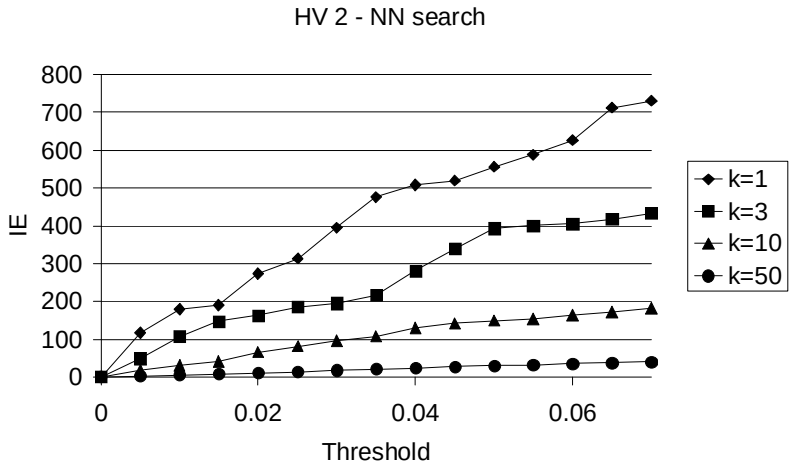
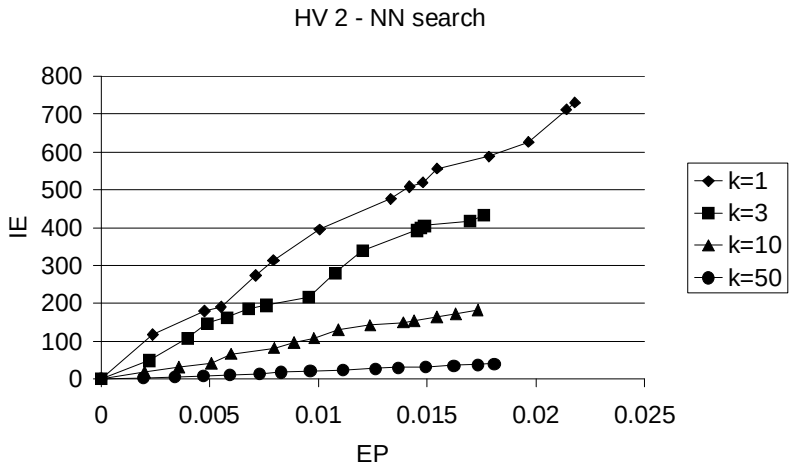


Figure 6.11: Improvement of efficiency (IE) as a function of the derivative threshold (x_s) and the position error (EP). Nearest neighbor queries, HV1 data set.





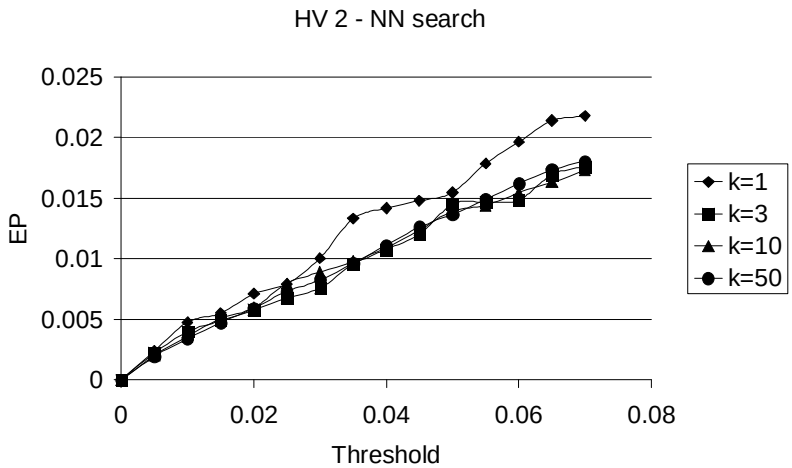


Figure 6.12: Improvement of efficiency (IE) as a function of the derivative threshold (x_s) and the position error (EP). Nearest neighbor queries, HV2 data set.

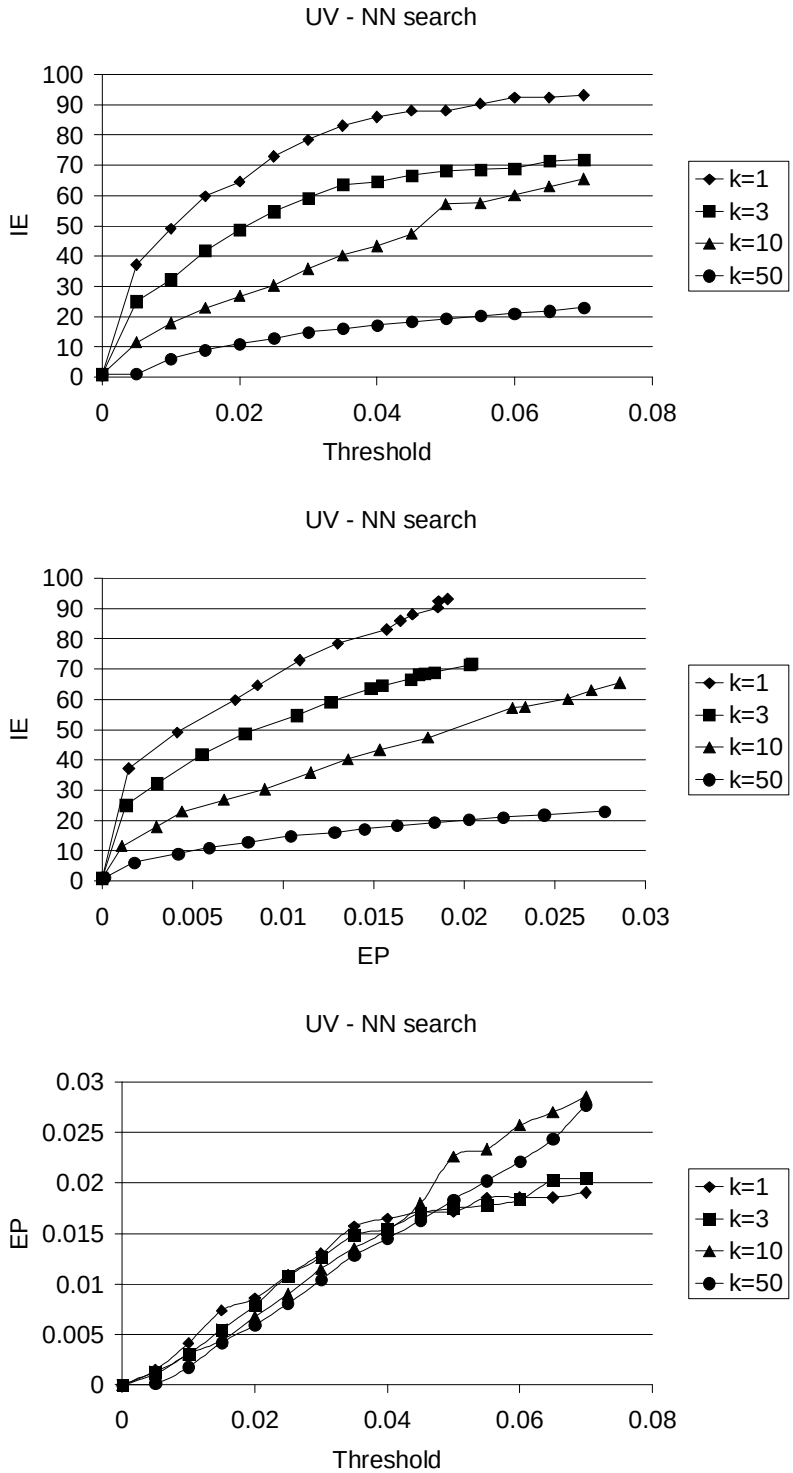


Figure 6.13: Improvement of efficiency (IE) as a function of the derivative threshold (x_s) and the position error (EP). Nearest neighbor queries, UV data set.

faster with $EP = 0.004$.

Figures 6.11, 6.12, and 6.13 show also the dependency between the approximation threshold and the EP measure. All curves are close enough to allow the user to control the quality of the approximation almost independently of k. Notice also that, since the dependency is almost linear, the user can linearly control the quality of approximation using the threshold parameter. However the improvement of efficiency, as we said before, strictly depends on the value of k. So, while the user might ignore k to control the quality of approximation by using the approximation parameter, she should also consider the value of k when she wants to control the improvement of efficiency.

6.8 Method 3: Approximate similarity search using the slowdown of distance improvement

The algorithm for nearest neighbors search (Algorithm 6.3.2) builds the result set through several iterations. In every iteration a new current result set can be obtained by improving the one resulting from the previous iteration. In fact, in every iteration some of the k current nearest objects, found in previous iterations, are possibly replaced by new nearest objects found in current iteration. This improvement can be observed as a reduction of the distance $d(O_k, O_q)$ of the k-th current object O_k from the query object. In fact, distances of the new objects found from the query object are shorter than distances of the replaced objects.

Let us define $d_{it}^{O_q,k}(iter)$ as follows

$$d_{it}^{O_q,k}(iter) = d(O_k(iter), O_q)/d_m$$

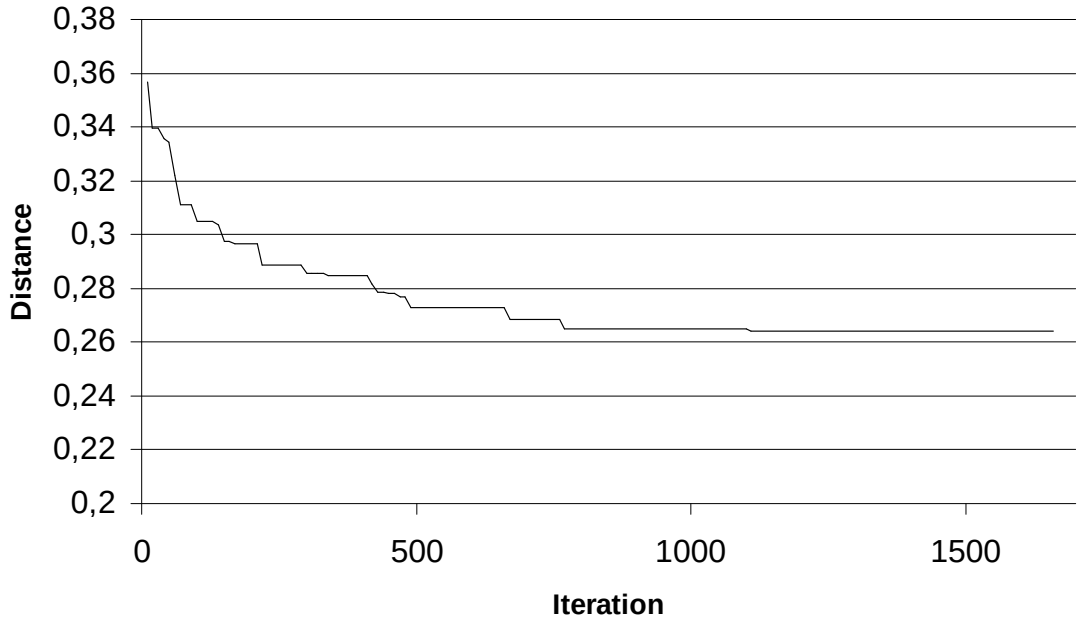


Figure 6.14: Trend of $d_{it}^{O_q,k}(iter)$, when k is 3, in HV1.

where $O_k(iter)$ is the k-th object in the $iter$ iteration and d_m is the maximum distance in the considered data set. $d_{it}^{O_q,k}(iter)$ is the function representing the normalized[4] distance of the current k-th object from the query object O_q at the iteration $iter$. As an example, Figure 6.14 shows the graph of $d_{it}^{O_q,k}(iter)$ for a 3 nearest neighbors search in data set HV1 for a particular query object. Considering this figure we can make some observations. On several iterations, the algorithm is not able to improve the current result set since no better objects are found. In fact, $d_{it}^{O_q,k}(iter)$ mainly assumes constant values in consecutive iterations. The iterations where an improvement of the current result set is obtained are a minority of the whole iterations. It can also be noticed that the improvement is very fast in the first iterations of the algorithm, then

[4]Different data sets have different range of distances. By the normalization, obtained by dividing the actual distance by d_m, the function is defined in a range that does not depend on the particular data set used.

it slows down and almost no improvement, or negligible improvement, occurs after a certain iteration.

We exploit this behavior of the nearest neighbors search algorithm to define a new approximate similarity search algorithm. The idea here is to stop the similarity search algorithm before the natural end, when the improvement slows down below a certain threshold specified by the user. Given the nature of this approximation technique, it cannot be applied to the range search algorithm.

Unfortunately, $d_{it}^{O_q,k}(iter)$ is a function that is not known a priori, in fact, its values become available as the search algorithm proceeds. In addition, as previously observed, it is a piecewise constant function, that is, there are portions where it is rigorously constant. Therefore, it cannot be directly used to effectively infer when improvement slows down. For instance, its derivative would be 0 as soon as in some consecutive iterations no improvement occurs, which may also happen when the algorithm is very far from the final result. To solve these problems we compute, as the algorithm proceeds, a regression curve, say $reg_d(iter)$, which approximate $d_{it}^{O_q,k}(iter)$, and we use it for deciding if the algorithms should be stopped. When the derivative $reg'_d(iter)$ of $reg_d(iter)$ is above[5] the user specified (negative) threshold x_s, in correspondence of the current iteration, the algorithm stops. The parameter x_s is used to control the tradeoff between approximation quality and performance improvement. Of course, small absolute values of x_s result in bad performance but high approximation quality, since the algorithm might stop close to the natural end. Large absolute values of x_s, on the other hand, result in high performance and bad quality, since the algorithm may stop too early, when the current result set is not close to the final

[5]Note that $reg_d(iter)$ will be defined to be monotonically decreasing so it will always have a negative derivative

result set. When the threshold is set to 0 the algorithm behaves as the exact nearest neighbors search, since the regression curve is defined in such a way that it has never a positive derivative.

This method is only based on the stop condition so the pruning condition performs the usual (exact) overlap test to discard a node only when its bounding region does not overlap the query region:

$$Prune(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i), x_p) = d(O_q, O_i) > r_q + r_i$$

The stop condition, on the other hand, iteration after iteration, refines the regression curve and checks if the derivative of the regression curve, on the current iteration, is above the approximation threshold:

$Stop(RS_c, x_s) =$ if this is the first iteration
set $iteration = 1$
else
set $iteration = iteration + 1$;
let O_k be the k-th element of RS_c;
compute $reg_d(iter)$
using the new point $(iteration, d(O_k, O_q))$
in addition to the points previously used;
if $iteration == 1$
return $false$
else
return $(reg'_d(iteration) > x_s)$;

The mathematical details about the computation of the regression curve $reg_d(iter)$, which approximates the trend of the improvement of distances, is discussed in next

subsection.

6.8.1 Approximating the improvement of distances by a regression curve

The function $d_{it}^{O_q,k}(iter)$ cannot be directly used to decide when the search algorithms can be stopped, since it is a discrete piecewise constant function. However, $d_{it}^{O_q,k}(iter)$ has a shape that can be effectively approximated by a continuous function. We obtain such an approximation by performing the linear regression [Hal52] of points of $d_{it}^{O_q,k}(iter)$ using the method of the discrete least squares approximation [BFR78].

Let us suppose that we know the values of a discrete function $f(i)$ when $i = x_1, ..., x_j$. Then, we can chose n $(n < j)$ real functions $\varphi_1(i), ..., \varphi_n(i)$ to obtain, as a linear combination, a regression curve $\varphi(i)$ that approximate $f(i)$:

$$\varphi(i) = c'_1 \cdot \varphi_1(i) + ... + c'_n \cdot \varphi_n(i)$$

The least squares method says that $c'_1, ..., c'_n$ should be chosen such that $\phi(c'_1, ..., c'_n)$, defined as follows, is minimum:

$$\phi(c_1, ..., c_n) = \sum_{i=1}^{j} \left(\sum_{s=1}^{n} c_s \cdot \varphi_s(x_i) - f(x_i) \right)^2$$

Let us see how this can be applied to our problem. Let us suppose that search algorithm arrived at iteration j. This means that we have the pairs $(i, d_{it}^{O_q,k}(i))$, for $1 \leq i \leq j$, where i corresponds to the i-th iteration and $d_{it}^{O_q,k}(i)$ to the distance of the k-th object from the query in the i-th iteration. These points can be used to obtain a regression curve that approximate $d_{it}^{O_q,k}(iter)$.

In our case the regression curve is obtained by using two real functions $\varphi_1(i)$ and $\varphi_2(i)$. The specific definition of $\varphi_1(i)$ is discussed later, while $\varphi_2(i) = 1$. Therefore, our regression curve $reg_d(iter)$ has the following form:

$$reg_d(iter) = \varphi(iter) = c_1 \cdot \varphi_1(iter) + c_2$$

According to the least square approximation method we should find c_1' and c_2' such that $\phi(c_1', c_2')$, defined as follows, is minimum:

$$\phi(c_1, c_2) = \sum_{i=0}^{j} \left(c_1 \cdot \varphi_1(i) + c_2 - d_{it}^{O_q,k}(i) \right)^2$$

It can be easily shown that if $\varphi_1(i)$ is chosen such that $\varphi_1(i) \geq 0$ when $1 \leq i \leq j$, then $\phi(c_1, c_2)$ is minimum when its partial derivatives are equal to 0. This corresponds to find c_1' and c_2' as a solution to the following:

$$\begin{cases} \frac{\partial \phi(c_1,c_2)}{\partial c_1} = 2 \left(c_1 \sum_{i=1}^{j} g^2(i) + c_2 \sum_{i=1}^{j} \varphi_1(i) - \sum_{i=1}^{j} \varphi_1(i) d_{it}^{O_q,k}(i) \right) = 0 \\ \frac{\partial \phi(c_1,c_2)}{\partial c_2} = 2 \left(c_1 \sum_{i=1}^{j} \varphi_1(i) + j \cdot c_2 - \sum_{i=1}^{j} d_{it}^{O_q,k}(i) \right) = 0 \end{cases}$$

that is

$$c_1' = \frac{j \sum_{i=1}^{j} \varphi_1(i) d_{it}^{O_q,k}(i) - \left(\sum_{i=1}^{j} d_{it}^{O_q,k}(i) \right) \left(\sum_{i=1}^{j} \varphi_1(i) \right)}{j \cdot \sum_{i=1}^{j} g^2(i) - \left(\sum_{i=1}^{j} \varphi_1(i) \right)^2}$$

and

$$c_2' = \frac{\left(\sum_{i=1}^{j} d_{it}^{O_q,k}(i) \right) \left(\sum_{i=1}^{j} g^2(i) \right) - \left(\sum_{i=1}^{j} \varphi_1(i) d_{it}^{O_q,k}(i) \right) \left(\sum_{i=1}^{j} \varphi_1(i) \right)}{j \cdot \sum_{i=1}^{j} g^2(i) - \left(\sum_{i=1}^{j} \varphi_1(i) \right)^2}$$

Notice that the number of iterations j, might be so high that computing $reg_d(iter)$ might be inefficient. However, here we used an optimization. As we said previously,

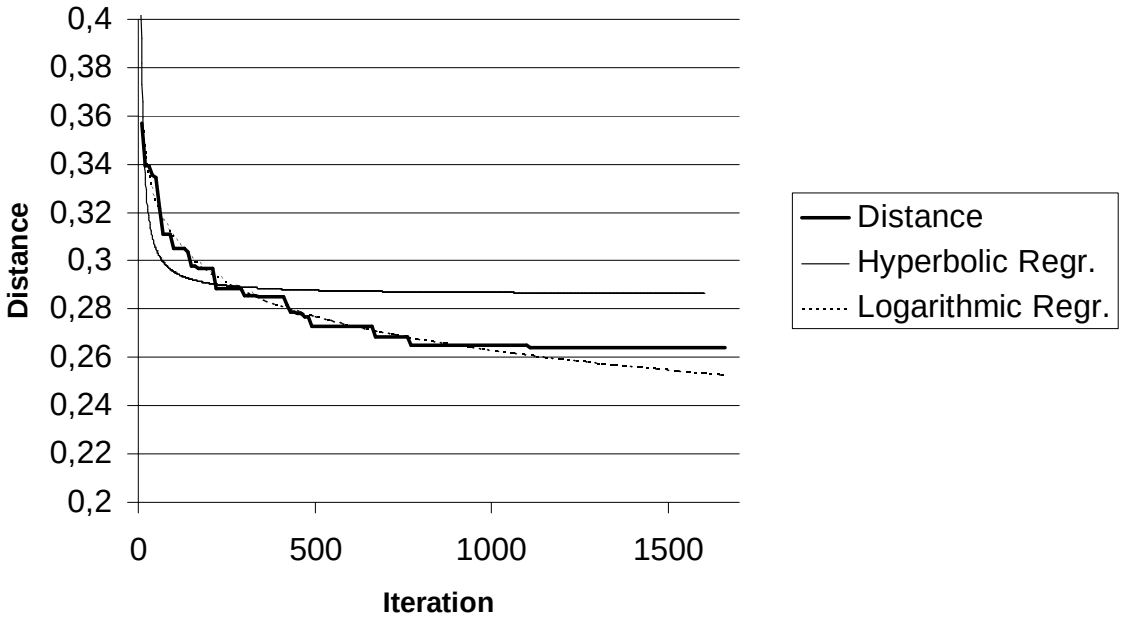


Figure 6.15: Trend of $d_{it}^{O_q,k}(iter)$, when k is 3 in HV1, and two possible regression curves.

$d_{it}^{O_q,k}(iter)$ is constant for several consecutive iterations. Therefore, instead of using all points, $reg_d(iter)$ can be computed by only using points where the value of $d_{it}^{O_q,k}(iter)$ changes. Specifically, $reg_d(iter)$ is computed using a set of points obtained as follows. Let i be the current iteration and $(i, d_{it}^{O_q,k}(i))$ the last point added to the set of points. At iteration $i+1$, if $d_{it}^{O_q,k}(i) = d_{it}^{O_q,k}(i+1)$, then the point $(i, d_{it}^{O_q,k}(i))$ is replaced by $(i{+}1, d_{it}^{O_q,k}(i{+}1))$, elsewhere, the point $(i, d_{it}^{O_q,k}(i))$ is maintained and $(i{+}1, d_{it}^{O_q,k}(i{+}1))$ is added to the list of points. This reduces enormously the number of points to be considered, maintaining a good approximation of $d_{it}^{O_q,k}(iter)$.

We still have to discuss the definition of function $\varphi_1(i)$. Its choice depends on the peculiar properties of $d_{it}^{O_q,k}(iter)$. In fact, different choices of $\varphi_1(i)$ might give different approximation qualities depending on their capability of miming the trend

of $d_{it}^{O_q,k}(iter)$. Given the typical shape of $d_{it}^{O_q,k}(iter)$, we have tried the hyperbola, that is $\varphi_1(i) = 1/i$ and the logarithm, that is $\varphi_1(i) = log(i)$. Figure 6.15 shows graphically $d_{it}^{O_q,k}(iter)$ and the two resulting approximations. We have used both choices in the approximate similarity search algorithm, and the average behaviour was practically the same in both cases. In fact, both choices gave almost overlapped results in term of IE and EP, even tough different ranges of the threshold parameters x_s had to be used. Next subsection presents the obtained results.

6.8.2 Results

We have tested the the approximate similarity search algorithm with the slow-down of distance improvement, by obtaining the regression curve through both the logarithm and the hyperbola as a definition for $\varphi_1(i)$. As previously anticipated, the obtained results were practically the same, so here we only discuss those obtained by using the logarithm. The same observations hold also for the other possibility considered. Results are summarized in Figures 6.16, 6.17 and 6.18.

In these experiments the approximation threshold x_s, is interpreted as a threshold on the derivative of $reg_d(iter)$, the curve that approximate the trend of distance of the current k-th object from the query object. The approximation threshold ranges in the interval between 0 and -0.004. When the derivative of $reg_d(iter)$, in the current iteration, is above the specified threshold, the algorithm stops. The approximation threshold x_p for the pruning condition is not used, since the pruning condition performs the exact overlap test.

A first general observation is that, even if the improvement of efficiency might arrives up to two orders of magnitude, the approximation threshold x_s is not easy

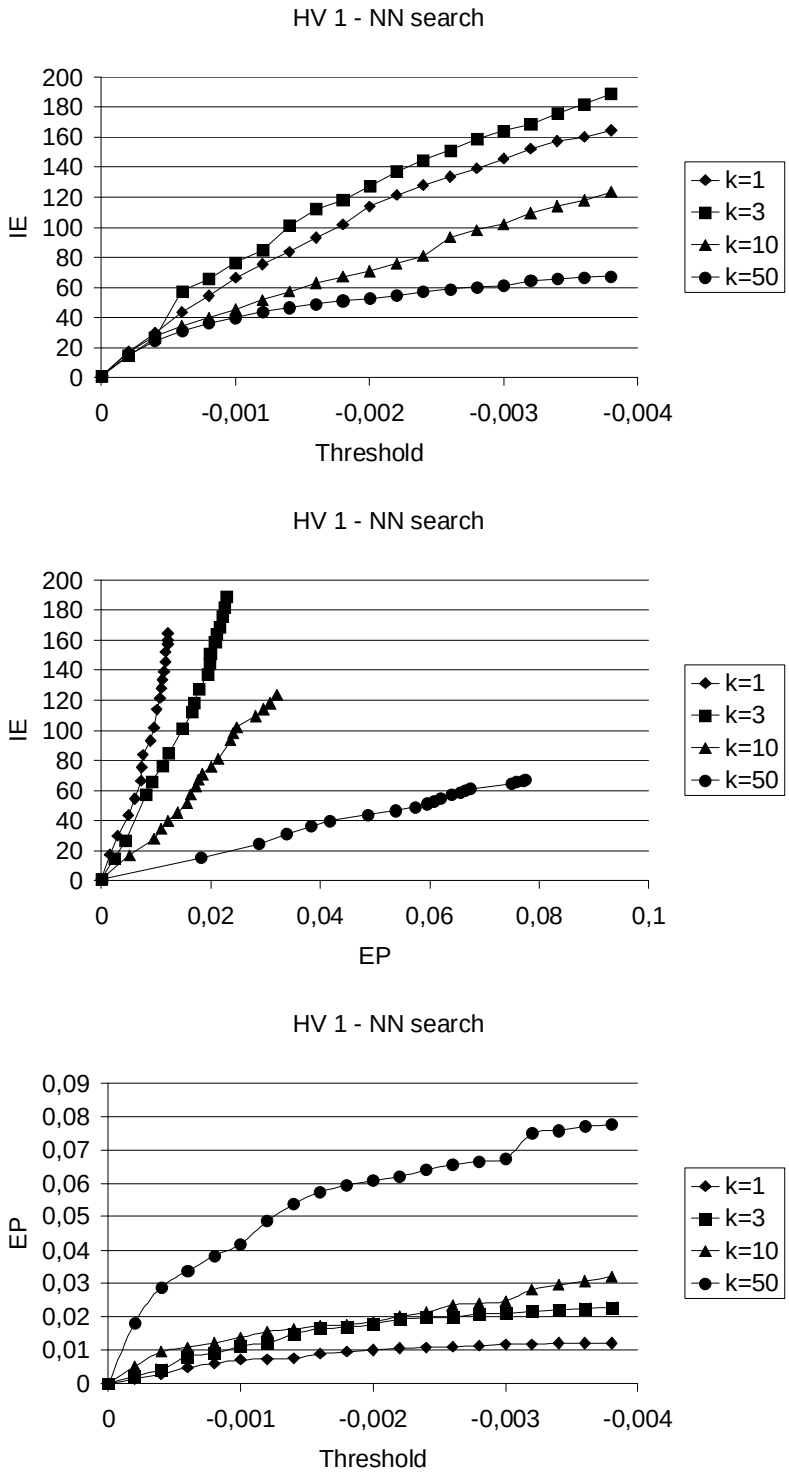


Figure 6.16: Improvement of efficiency (IE) as a function of the derivative threshold (x_s) and the position error (EP). Nearest neighbor queries, HV1 data set.

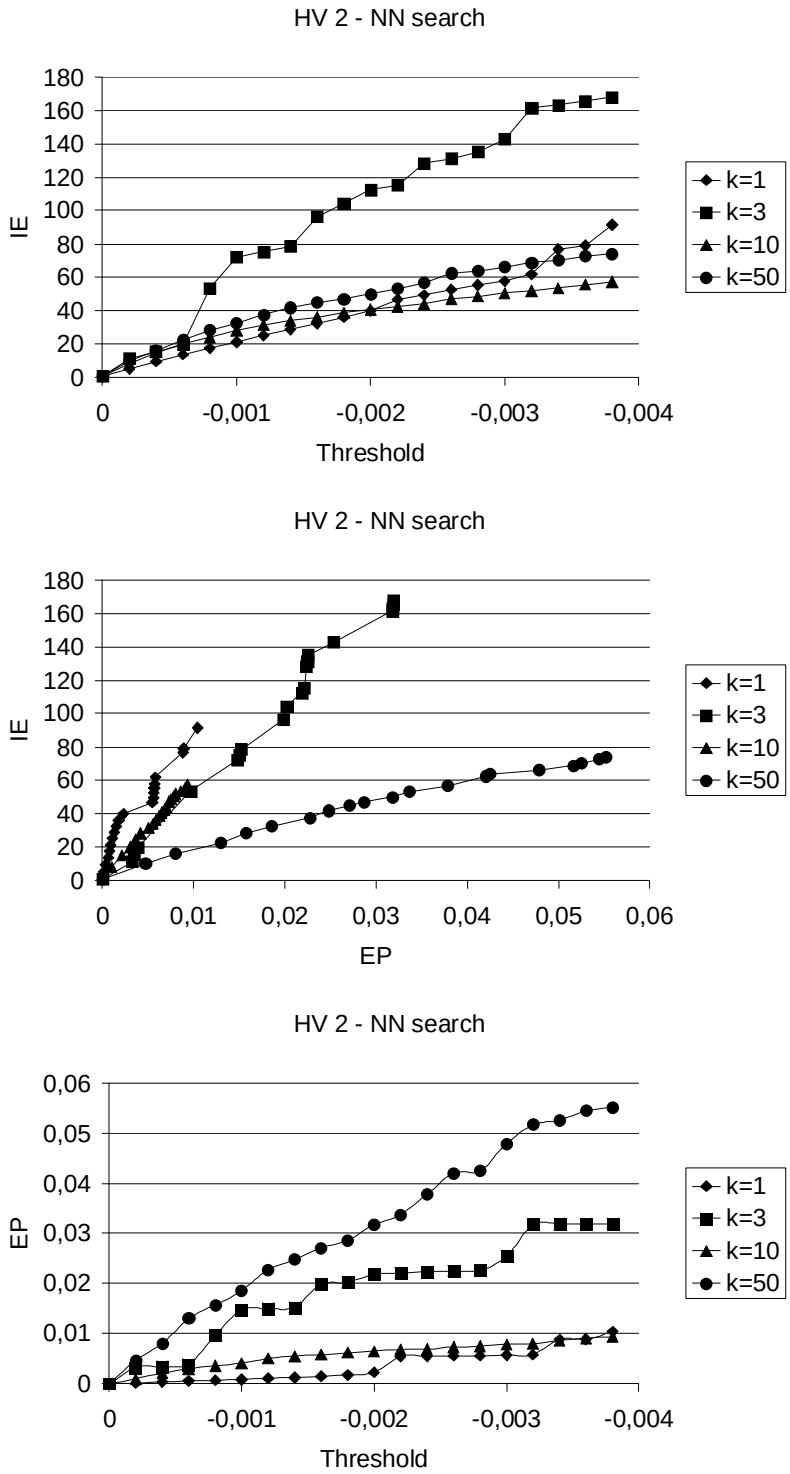


Figure 6.17: Improvement of efficiency (IE) as a function of the derivative threshold (x_s) and the position error (EP). Nearest neighbor queries, HV2 data set.

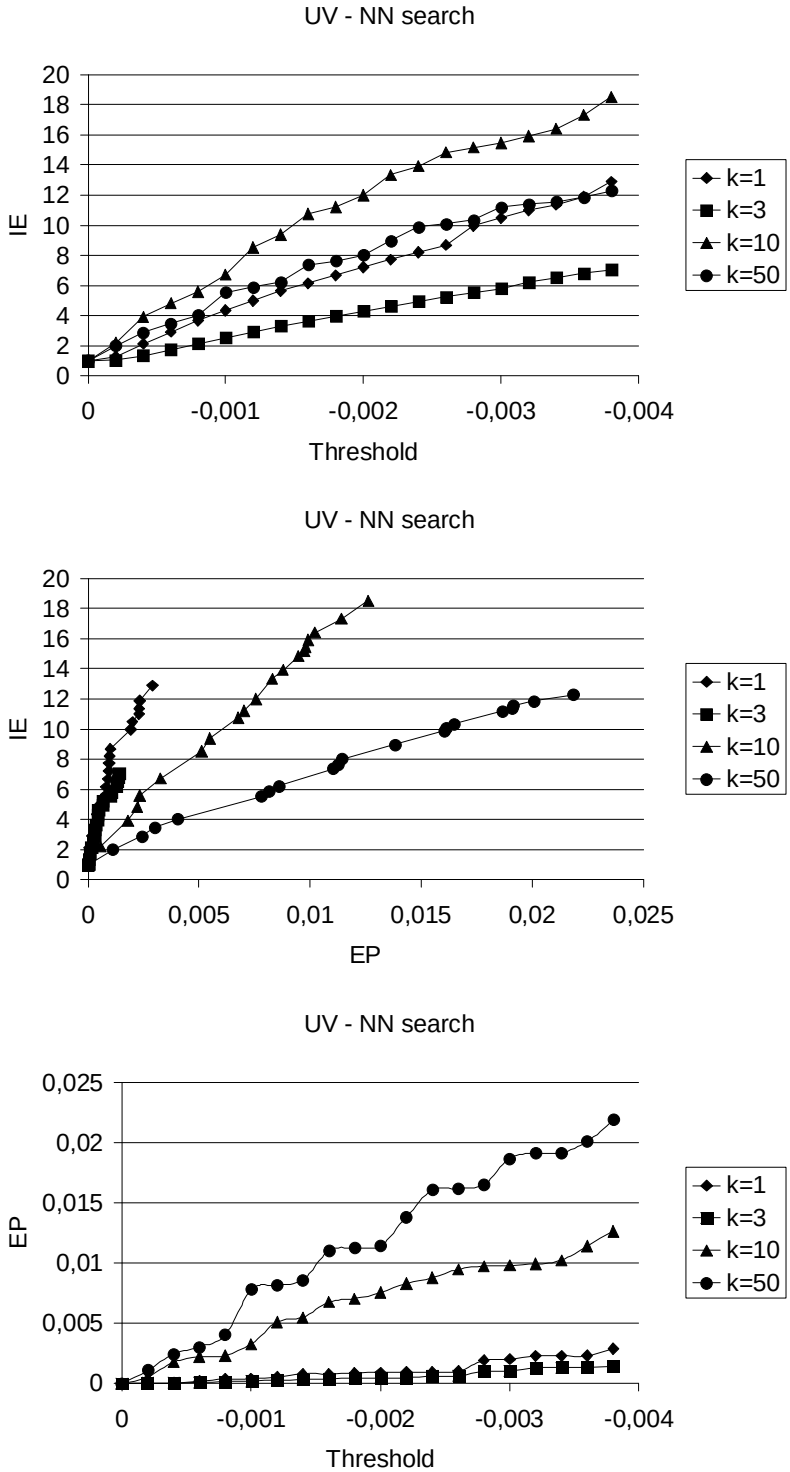


Figure 6.18: Improvement of efficiency (IE) as a function of the derivative threshold (x_s) and the position error (EP). Nearest neighbor queries, UV data set.

to be used to control the tradeoff between performance and effectiveness, when the number of retrieved objects k is varied. For instance in HV1, given a fixed threshold, the best improvement of efficiency where observed for $k = 3$, then $k = 1$, $k = 10$, and, finally, $k = 50$. That is, the improvement of efficiency depends on k but the corresponding relationship is not trivial. However, the error on the position EP, with the exception of data set UV, is almost directly proportional to k. In fact EP degrades as k increases when the approximation threshold is fixed.

Let us now discuss more in details the obtained results. An improvement of efficiency up to 2 orders of magnitude was obtained, maintaining a relatively good quality search results. In fact, in HV1 the nearest neighbor can be obtained on average 190 times faster, with $EP = 0.023$. Performances obtained for HV1 and HV2 are comparable, even though slightly better results are obtained for HV2. However, in UV results are clearly worse. The approximate algorithm can find the nearest neighbor 60 times faster with $EP = 0.006$ in HV1 and $EP = 0.005$ in HV2. This means that, for instance in in HV1, the approximate nearest neighbor is, on average, the 60-th nearest neighbor (out of 10000). However, when the precise search takes 1 minute to compute, the approximated result is obtained in 1 second. Similar relationships between HV1 and HV2 can be observed also for other values of k. On the other hand, in the UV data set, using the same range of approximation threshold, the improvement of performance arrives only up to one order of magnitude and the results where typically worse than in the other data sets. In fact, in UV, the nearest neighbor was found 13 times faster with $EP = 0.0025$, while in HV1 and HV2 the nearest neighbor was found with the same value of EP respectively 30 and 40 times faster.

6.9 Method 4: Approximate similarity search using the region proximity

Access methods for metric spaces partition the searched data, and bound element of the partition by ball regions. In order to find qualifying objects, search algorithms should access all nodes of the tree corresponding to regions that overlap the query region. All regions that overlap the query region may potentially contain objects that are also covered by the query regions. Search algorithms might ignore all regions that do not overlap the query region in order to be more efficient, since accessing a node (corresponding to a region) has a high cost. However, notice that when a data region and the query region overlap with each other, there is no guarantee that objects are included in their intersection. In fact depending on the data distribution, it may happen that the intersection covers a portion of the space containing very few objects (or no objects at all). Therefore, some of the data regions that overlap the query region may not contain searched data, and some regions are more likely to contain the query response than the others.

In Figure 6.19, it can be seen that, although the query region Q intersects regions $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$, the intersection with $\mathcal{R}_1$ and $\mathcal{R}_3$ is empty and thus it is not necessary to access these regions.

Proximity between ball regions, discussed in Chapter 4 may help to handle this situation. In fact, as defined by Equation 4.2.2, proximity of two regions is the probability that objects can be found in their intersection. The idea here is to use the proximity to decide when data regions should be accessed to find qualifying objects, in fact, the higher the proximity, the more "interesting" the regions.

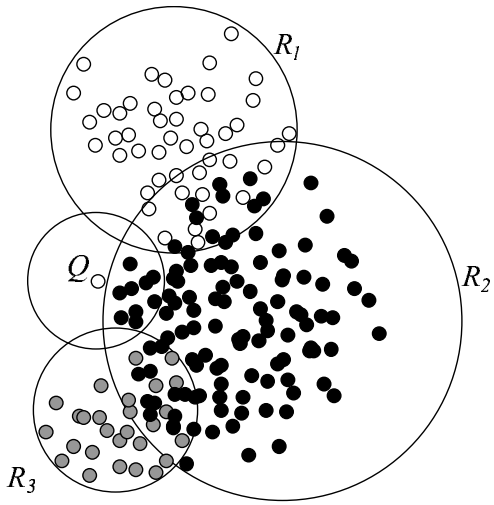


Figure 6.19: Overlap between the query region and data regions: not all data regions that overlap the query region share objects with it.

This idea constitutes the basis for the approximate similarity search by using proximity. In fact, approximated results are obtained by considering, in the similarity search algorithms, only regions with proximity greater than a certain threshold, say x_p, i.e. regions in which the probability of containing a qualifying object is greater than x_p. The approximation threshold x_p is specified by the user and can be used to control the approximation. High values correspond to high performance but bad quality. Small values corresponds to low performance but high quality. When x_p is set to 0, an exact similarity search is performed, since proximity is greater than 0 as soon as two regions overlap.

Approximate range search and approximate nearest neighbor search, which use this specific strategy, can be obtained from Algorithms 6.3.1 and 6.3.2 by defining the pruning condition and the stop condition as follows.

The pruning condition $Prune(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i), x_p)$ is defined as

$$Prune(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i), x_p) = X(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i)) < x_p \quad (6.9.1)$$

The stop condition on the other hand is always false:

$$Stop(RS_c, x_s) = false$$

The proximity $X(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i))$ between the query region and the selected data regions was computed using the parallel heuristic, described in 4.5, which proved to be the more precise among the ones tested.

6.9.1 Results

Results of the experiments using the proximity based approximate similarity search algorithms are summarized in Figures 6.20, 6.21 and 6.22 for range queries and in Figures 6.23, 6.24 and 6.25 for nearest neighbor queries.

In these experiments the approximation threshold x_p, is interpreted as the proximity threshold. When the proximity of the query region and a data region is below the specified threshold, the data region is discarded. The proximity threshold ranges in the interval between 0 and 0.06. The approximation threshold x_s for the stop condition is not used, since the stop condition is always false.

The results show that the best improvement of efficiency is achieved when the size of the result set is small. The number of retrieved objects is explicitly specified for the nearest neighbors queries, but is quite difficult to control by specifying a range. Note that the response sets for our range queries contains on average more than 100 objects (1% of 10,000). In fact, the approach offers better performance for nearest neighbors queries with small k rather than for the range queries.

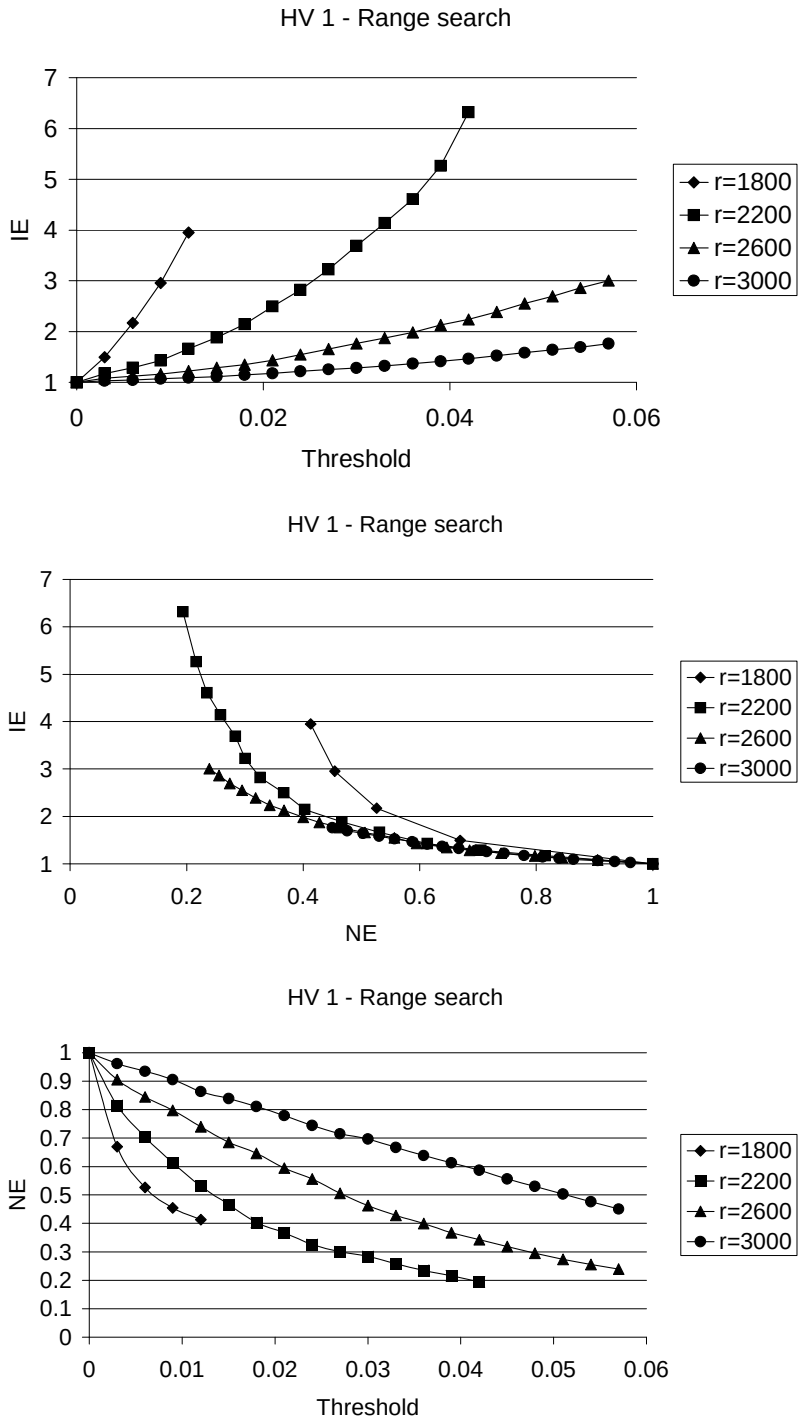
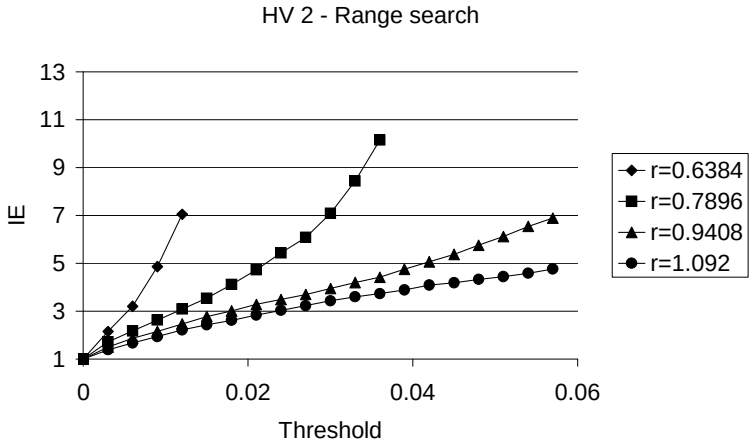
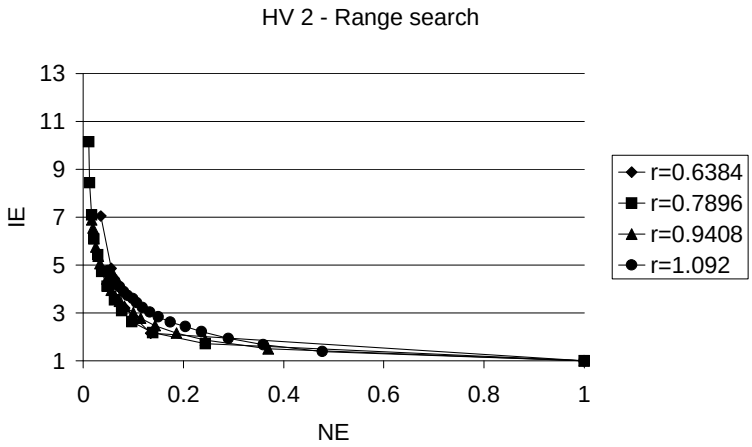


Figure 6.20: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the fraction of exact results (NE). Range queries, HV1 data set.





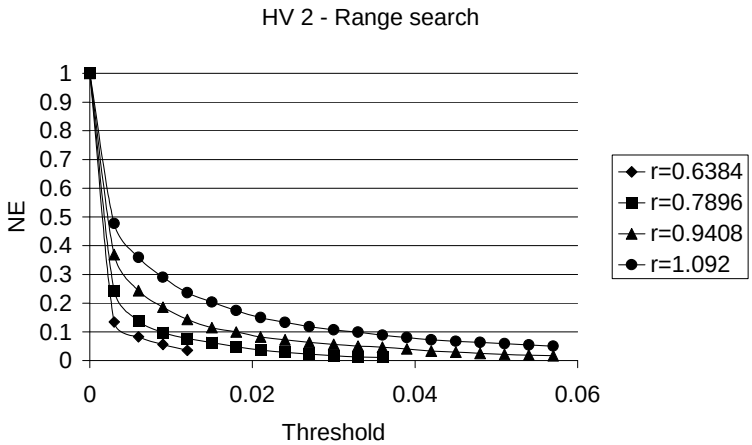


Figure 6.21: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the fraction of exact results (NE). Range queries, HV2 data set.

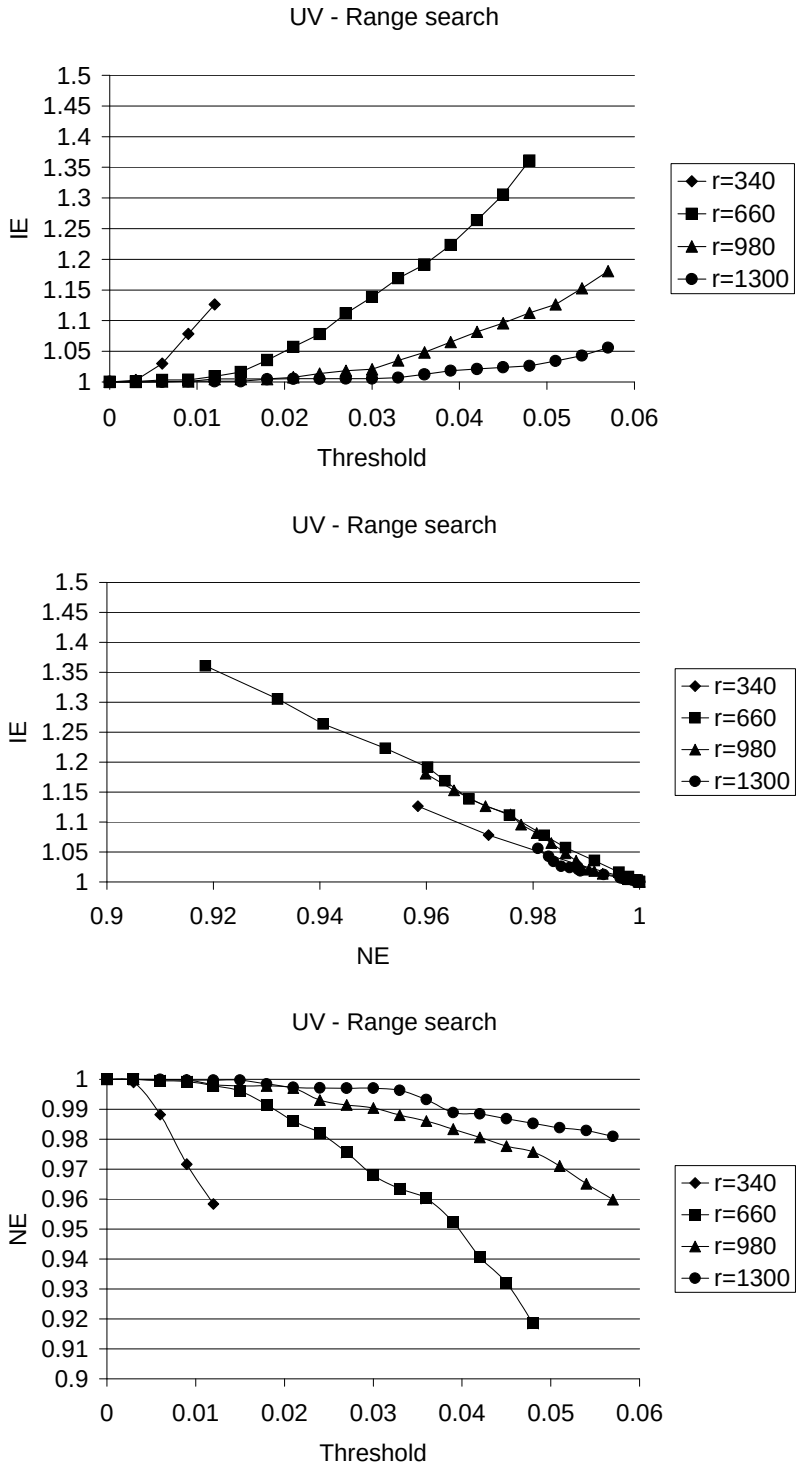
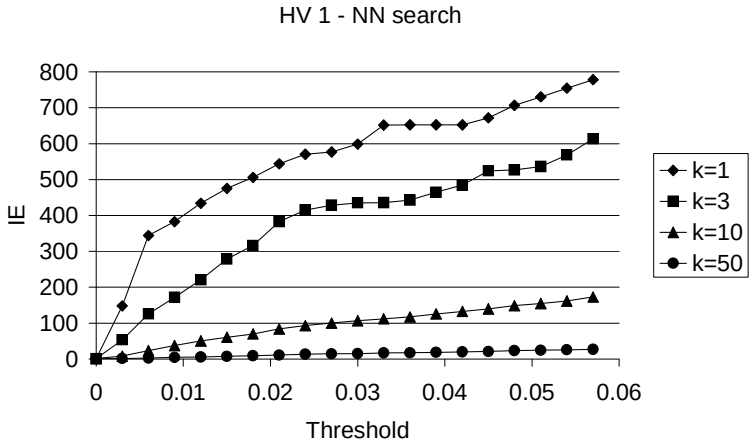
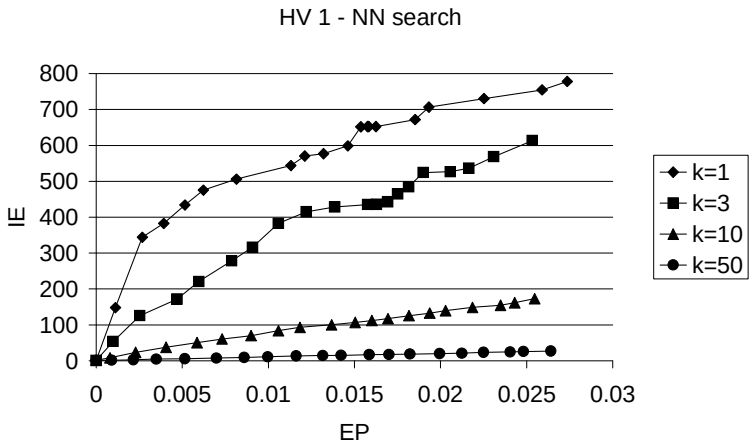


Figure 6.22: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the fraction of exact results (NE). Range queries, UV data set.





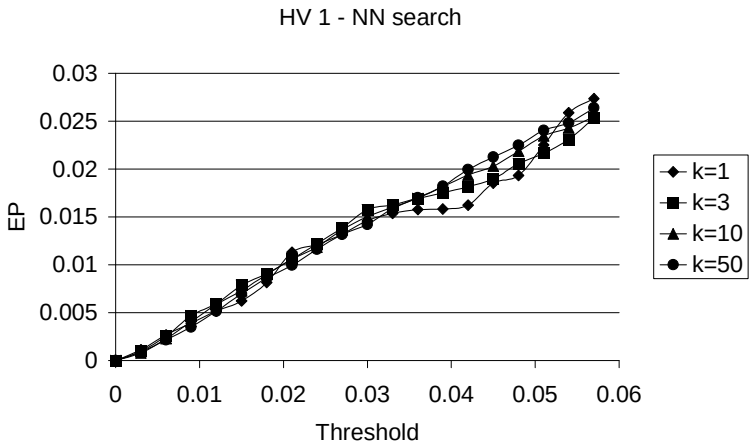


Figure 6.23: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the position error (EP). Nearest neighbor queries, HV1 data set.

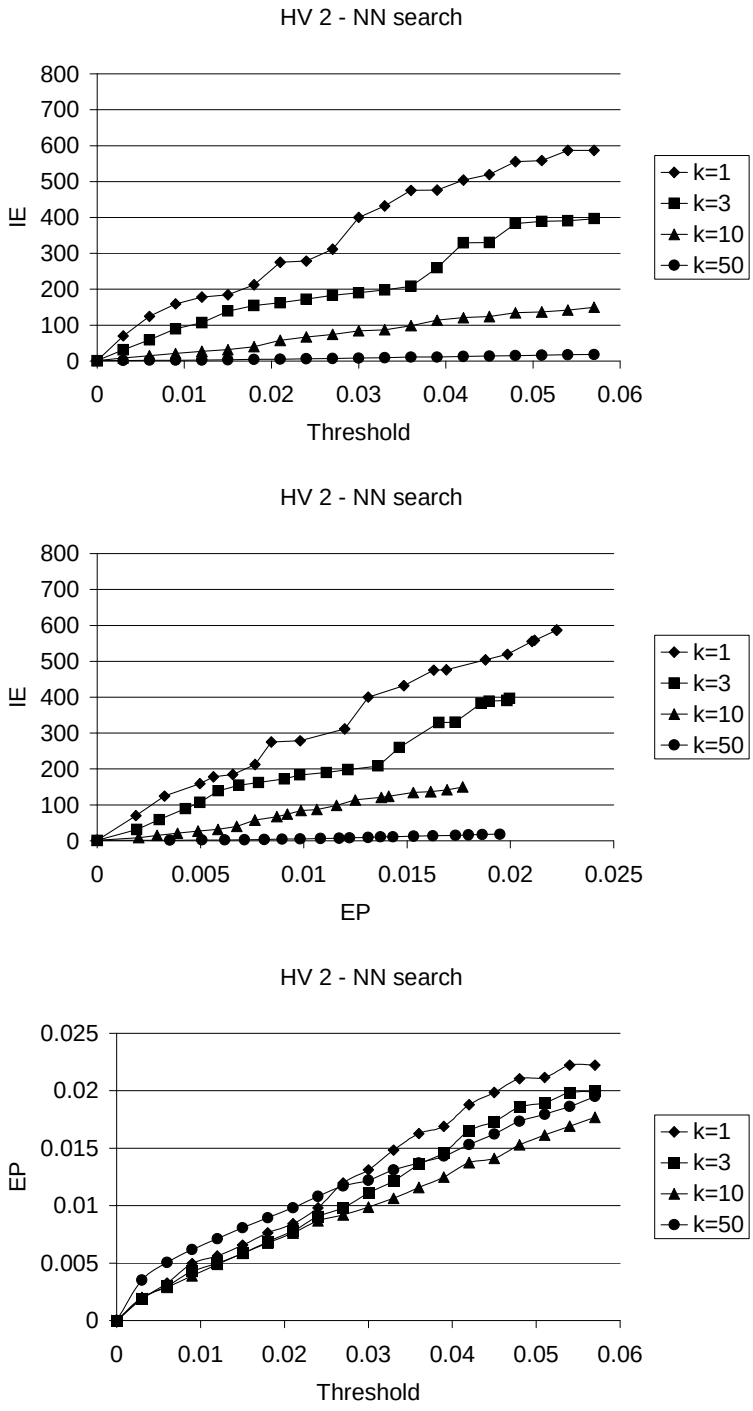


Figure 6.24: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the position error (EP). Nearest neighbor queries, HV2 data set.

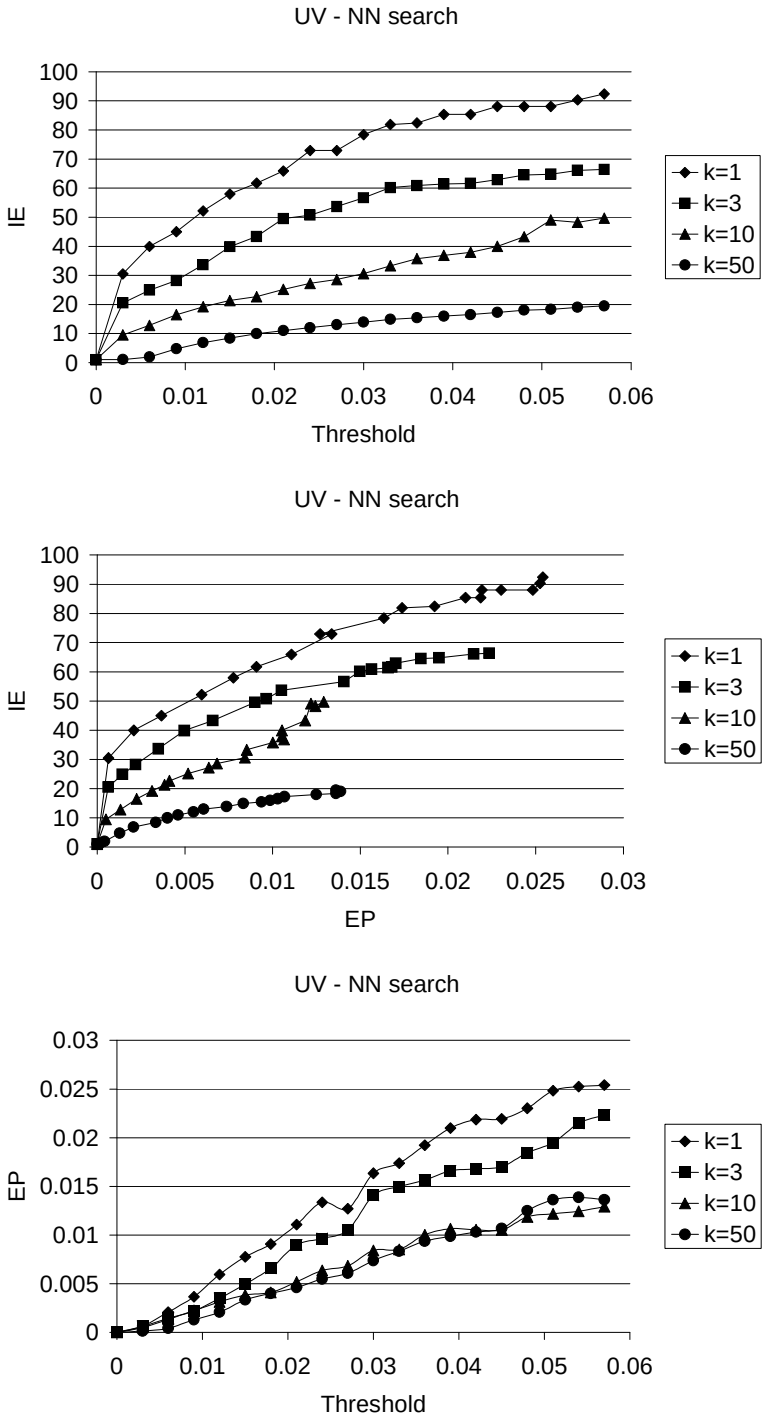


Figure 6.25: Improvement of efficiency (IE) as a function of the proximity threshold (x_p) and the position error (EP). Nearest neighbor queries, UV data set.

Let us now consider the range queries more closely. The improvement of efficiency decreases when the query radius grows. In all data sets, the improvement obtained is less than one order of magnitude. As expected, IE increases as the NE measure decreases. Note that we do not report results for $NE = 0$. Results seem to be better for the HV1 data set rather than the HV2 and UV data sets. In HV1, for instance, when the query radius is 2200, a query can be executed on average 6.5 times faster, with $NE = 0.2$, that is 20% of objects retrieved by the exact search were found by the approximate search. When the HV2 data set is used, results are slightly better in terms of an improvement of efficiency, however the NE measure returns worse results than for the HV1 data set. In fact, improvement of efficiency higher than 3 typically results in NE close to 0. For all radii considered, when the improvement of efficiency is 6.5, the NE measure is about 0.05. In case of the UV data set it was not possible to obtain improvement of efficiency higher than 1.4. In fact, when the approximation parameter was chosen a little bit larger suddenly the number of retrieved objects was 0. Next sub-section gives an explanation for this behaviour.

In the case of nearest neighbor queries, results are much better. In fact, an improvement of efficiency up to 2 orders of magnitude was obtained, still maintaining good quality search results. There is no significant difference between HV1 and HV2, even though slightly better results are obtained for HV1. However, in UV was again difficult to obtain improvements comparable with the other data sets. For instance, the approximate algorithm can find the nearest neighbor in HV1 60 times faster with $EP = 0.0005$. This means that the approximate nearest neighbor is on average actually the 5-th nearest neighbor. However, provided the precise search takes 1 minute to compute, the approximated result is obtained in 1 second. If the requirements

regarding precision are not so high, the approximate algorithm can perform much faster. For example, a 300 times faster approximate search implies an error in position $EP = 0.003$. In this case, the approximate nearest neighbor is the 30-th actual neighbor, but even queries which would require 5 minutes to get precise results, can be performed through approximation in 1 second. On the other hand, in the UV data set the nearest neighbor can be found 60 times faster with $EP = 0.008$. That is, the approximate nearest neighbor is the 80-th nearest neighbor.

Notice the dependency between the proximity threshold and the EP measure in Figures 6.23 and 6.24. All curves are almost overlapped and this suggests that it is possible to control the effectiveness by using the threshold parameter, independently of the value of k. In addition, since the dependency is almost linear, it means that the user can linearly control the quality of approximation using the threshold parameter. However, in UV this linear dependency is less marked, see Figure 6.25, and no profitable dependency between the threshold and NE can be observed in case of the range queries, see Figures 6.20, 6.21, and 6.22.

We have also tested our approach to approximate similarity search by substituting the probabilistic proximity with the trivial one, defined by Equation 4.2.1. Not only did much higher values of approximation threshold have to be used, but also the performance of approximated queries with respect to trivial proximity was systematically worse. When the same values of efficiency are considered, it is evident that the quality of approximation for the probabilistic approach is much higher than for the trivial approach. For example, see Figure 6.26 where the error on the position EP and improvement of efficiency IE are related for $k = 10$, separately for HV1 and

HV2 data sets. In HV1, IE increases almost linearly with EP for the probabilistic approach, while IE is almost constant up to values of $EP = 0.01$ and then it increases slowly. In HV2, again, IE increase almost linearly with EP for the probabilistic approach, while IE is almost constant up to $EP = 0.04$ and then it increases, but more slowly than in HV1. For example, when $EP = 0.01$, the probabilistically approximated query performed 70 times faster than the precise query and about 25 times faster than the approximated query with the trivial proximity. Such behavior was identical both for HV1 and HV2.

6.9.2 Further observations

When the query radius is small, the query response set can become empty, even for small values of the proximity threshold. Though it might look strange, the explanation is easy and provisions can be made to avoid such situations to happen.

It is easy to show that the proximity of two ball regions is smaller than or equal to the probability that a randomly chosen point belongs to the smaller of the two regions. Such probability can be approximated by $F(r)$ (see Section 2.3.4), where F is the overall distance distribution and r is the radius of the smaller region. Since for small values of the query radius $F(r_q)$ is very small, the proximity between the query region and any other region is also small. When $x_p > F(r_q)$, the pruning condition prunes every node of the tree. In fact, in this case, proximity between the query region and any other region is always smaller than x_p. Notice that the result is anyway correct, but empty approximated result is meaningless. In order to avoid such situations, the relationship between the proximity threshold and the query radius must be respected.

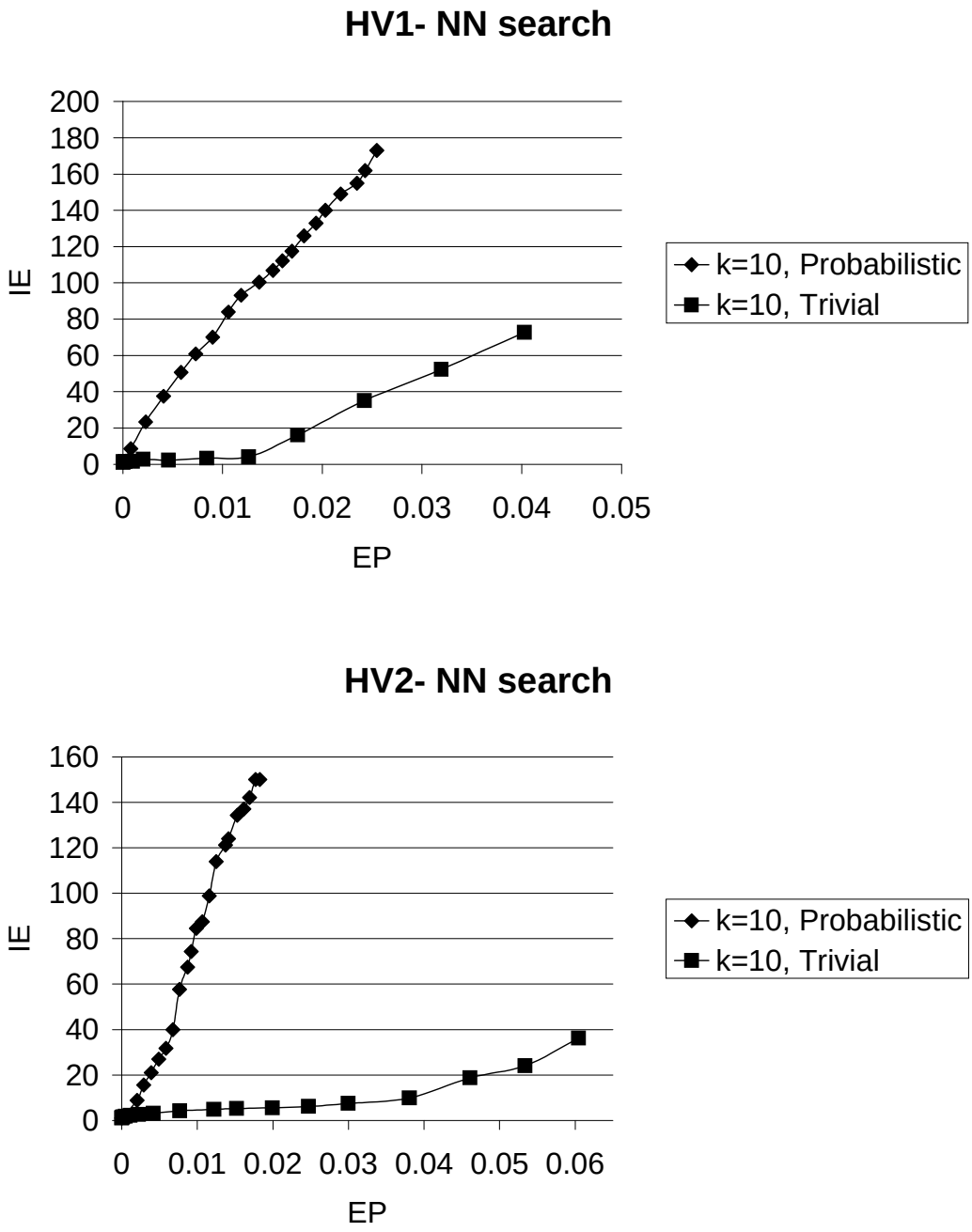


Figure 6.26: Comparison of the trivial and probabilistic approximation techniques

Several strategies can be used to find an appropriate query specification, because the distance distribution function F is known. For example, the system can suggest the smallest threshold that can be specified for a given query radius. Vice-versa, given a threshold, the system can tell which is the smallest radius that can be used with that threshold. Another possibility is to have the threshold automatically corrected (normalized) by the system. For instance the pruning condition $X(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i)) < x_p$, can be substituted by $X(\mathcal{B}(O_q, r_q), \mathcal{B}(O_i, r_i))/F(r_q) < x_p$, obtaining an automatic normalization.

This problem is not relevant for nearest neighbors queries. In fact, for this type of similarity query, the radius of the query region is dynamically changed as the algorithm proceeds. It is very big at the beginning and it becomes small just towards the end of query evaluation, so the influence of the threshold on the number of requested objects is less evident and a response set of k objects is always provided. However, in case of nearest neighbors queries, this phenomenon has another effect. In fact, when the dynamic radius of the query region reduces such that $x_p > F(r_q)$, then all remaining regions are pruned and, since the queue of pending requests become immediately empty, the algorithm immediately stops. The result is that, even if this method is defined as an approximate pruning condition, due to this phenomenon, this method implicitly acts also as if a stop condition was defined.

6.10 Cross comparisons

Previous sections were devoted to the individual description of the various approximation methods proposed in this thesis and to the analysis of the experimental results obtained by testing them. In this section, on the other and, we compare together all

methods proposed and we discuss their advantages and disadvantages. First a direct comparison of the performance of the various methods is presented, separately for range and nearest neighbors search. Then some global considerations not strictly related to performance are discussed.

6.10.1 Range queries

Figure 6.27 compares the results obtained by executing range queries. Only the first approximation method, the one based on the relative error on the distances, and the fourth, the one based on the proximity, are presented, since only these two methods can be used for range queries. To simplify the presentation, we do not show the comparison for each radius considered in the tests, but results obtained for only one radius are shown. In particular the comparison is presented by using the third radius used in each data set, specifically 2600 in HV1 , 0.9408 in HV2, and 980 in UV. The graphs in the figure relate the NE measure with the improvement of efficiency IE.

In HV1 the best results are achieved by the fourth method (proximity). In fact, given any value of NE, the corresponding value of IE is always higher for the proximity based method than for the other. In HV2 the method based on the relative error of distances is better. However, for small values of NE, that is when the accuracy of the approximation is low, the trend of fourth method is to rapidly increase the performance in term of improvement of efficiency. In UV, again, the best results are given by the fourth method. In fact, in this case, it offers high values of IE even when NE is still high.

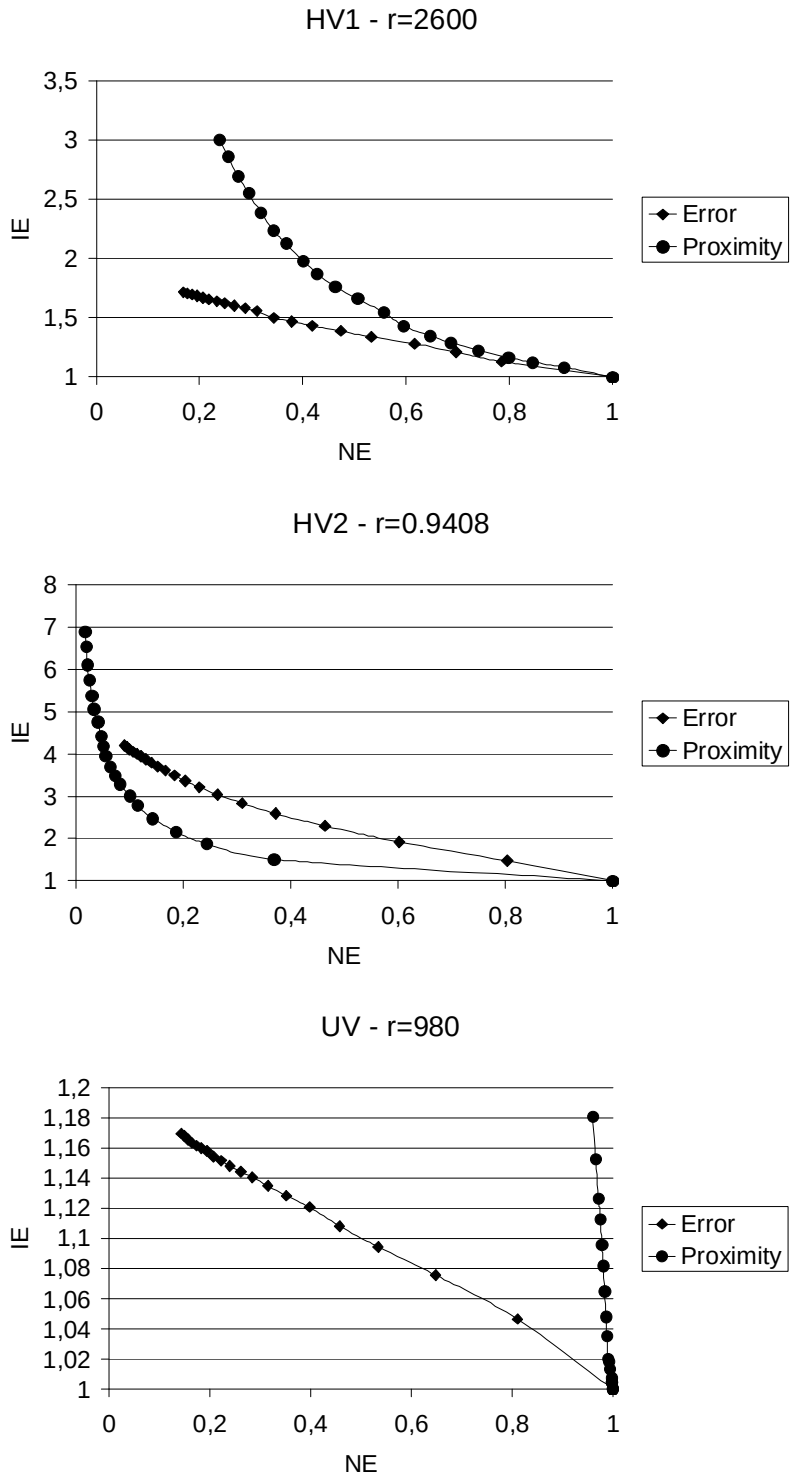


Figure 6.27: Comparison of the approximation methods that support range queries in the various data sets.

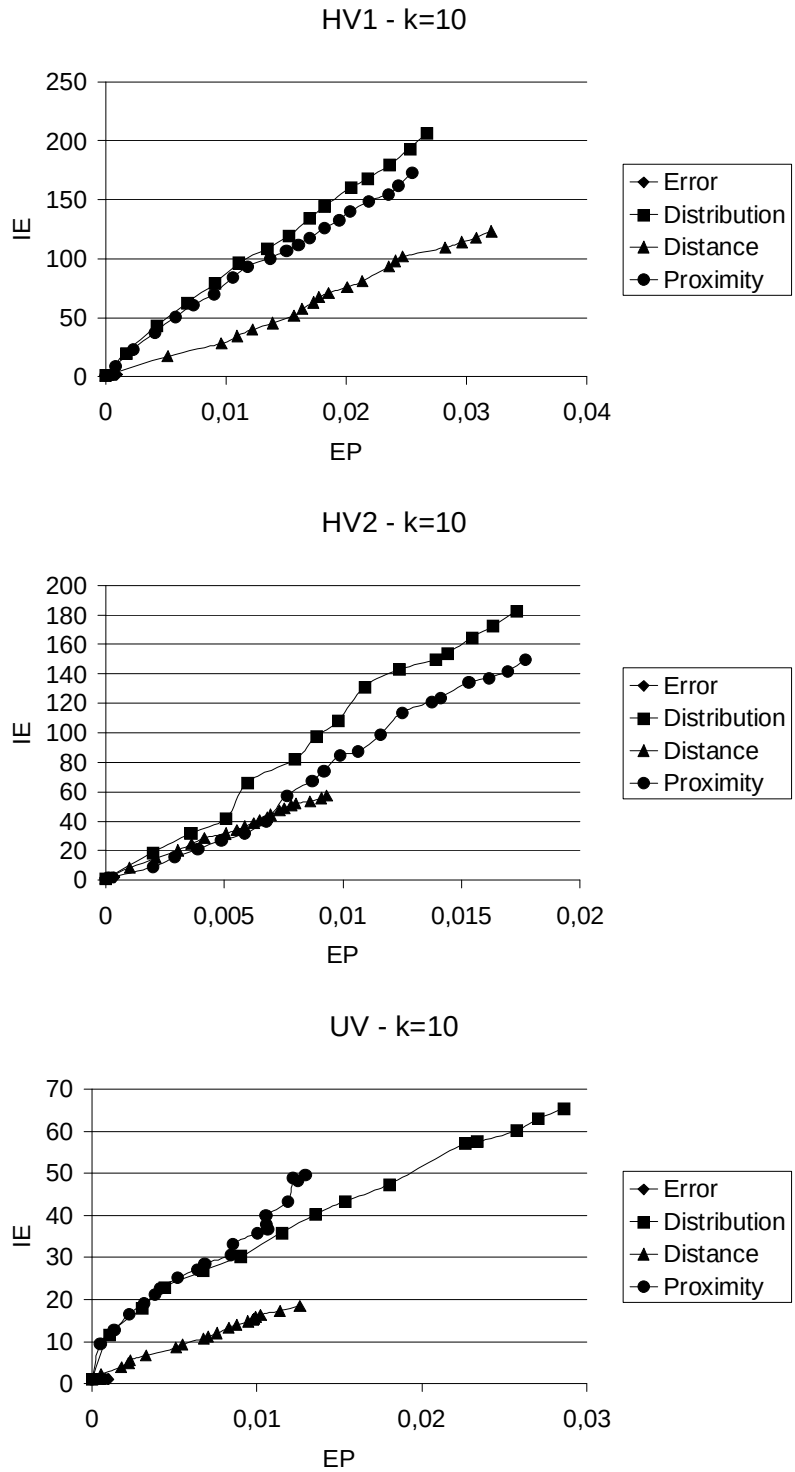


Figure 6.28: Comparison of all approximation methods for nearest neighbor queries in the various data sets.

6.10.2 Nearest neighbors queries

Let us now consider the results of executing nearest neighbor queries. Comparisons are presented in Figure 6.28. All methods can be used to execute nearest neighbor queries so all methods are compared. Also here for simplicity, as we did for range queries, we do not show the comparisons for all k values that were tested. The comparison is presented only for $k = 10$ in all data sets. The graphs presented relate the EP measure with the improvement of efficiency IE.

With exception of the first method (distance error), all methods offer an improvement of efficiency up to two orders of magnitude. The first method offers a very limited improvement in efficiency. Even using high approximation thresholds high improvements cannot be obtained. In the graph, the experimental values of this methods are very close to the origin of the axis and can be barely noticed, given the higher performance of the other methods. As we said in section 6.6, where the first method was presented, it immediately saturates and no relevant improvement is obtained. In HV1 and HV2 the highest performance is offered by the second method (distance distribution), while in UV the best method is the fourth method (proximity). In HV1 and UV, the curves of the second and the fourth method, however, are close to each other, and their performance is definitively better than that of the third method. In HV2, on the other hand, there is no significant difference among the performances of all methods.

Looking at the maximum performance obtained by our approximate nearest neighbors search algorithms in the different data sets, we can see that the improvement of efficiency seems to be more significant for the 45 (that is HV1) rather than the 32-dimensional (that is HV2) vectors. In addition, the improvement in efficiency is

negligible in case of the 2-dimensional vectors (that is UV). We have also experimented with other data sets and the general conclusion is that our methods of approximation for nearest neighbors queries are suitable above all when the precise similarity search tends to access many data nodes in the supporting tree structure. Such a situation often happens when the data partitioning results in highly overlapping regions, which is common for high-dimensional vector spaces. In these cases, the improvement is typically registered in hundreds, which is not possible to achieve for low dimensional spaces where even the exact similarity search algorithms are efficient. For this reason, results with the 2-dimensional UV data set are always worse that those obtained with the other data sets.

As previously pointed out, no clear relationship between the performance and the number of retrieved nearest neighbor was found for the first and the third method in the various data sets. On the other hand, in the second and in the fourth method, performance systematically deteriorates when the number of nearest neighbors retrieved increases. This can be explained as follows. We can observe that at any specific iteration of the search algorithm the distance of the current k-th nearest neighbor from the query object increases as k increases. To illustrate this, consider Figure 6.29 which relates the distance of the current k-th nearest neighbor from the query object and the number of iteration of the exact nearest neighbors search algorithm, separately for $k = 1, 3$, and 10. Observe that these distances for $k = 1$ are systematically below those for $k = 3$, and these are systematically below those for $k = 10$. In case of the second method, this means that the higher k, the larger $F(O_q, O_c^k)$, so more iterations should be executed before that it goes below the approximation threshold. On the other hand, in case of the fourth method, at each iteration of the nearest neighbors

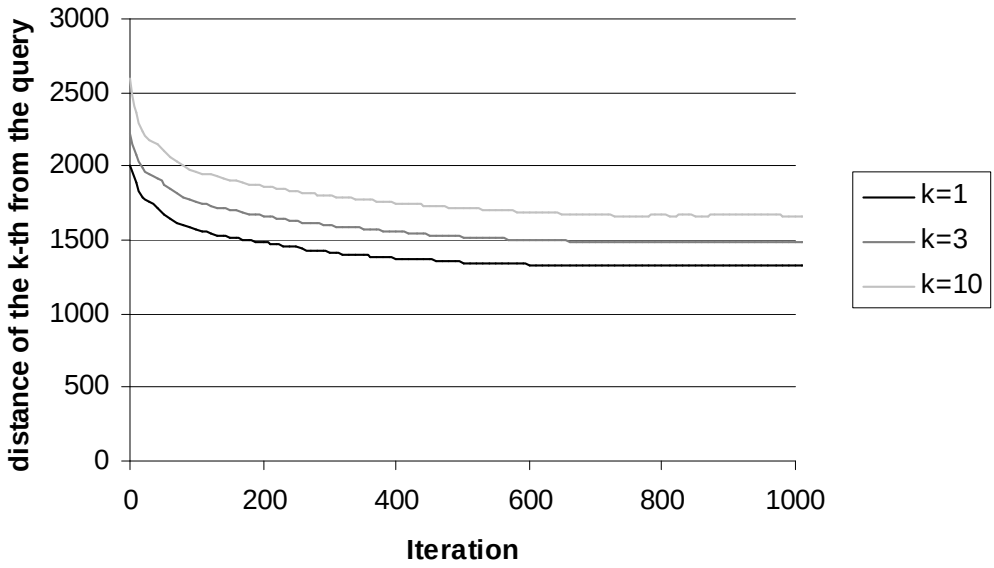


Figure 6.29: Average trend of the distance of the current k-th object from the query object during the exact nearest neighbor search execution in HV1

algorithm, the query radius is set to the distance between the query and the current k-th nearest neighbor. This means that a higher k systematically results in bigger query regions. While searching, the set of data regions is fixed so the bigger the query region, the higher the proximity of specific data and query regions. Consequently, a systematically higher proximity has a greater chance of exceeding the approximation threshold, thus more data regions are accessed.

6.10.3 Global considerations

Summarizing, the approximation methods that we have proposed give moderate improvement of performance for range queries, while very high improvement is obtained for nearest neighbor queries. The method that offers the highest performance is the

second one, however it can only be used for nearest neighbor queries. On the other hand, the fourth method offers similar, even if slightly worse, results for nearest neighbors queries, and it can also be used for range queries. The improvement of performance of the third method also arrives up to two orders of magnitude, but it is always below that of the other two. The first method can hardly be compared with the others given its very low performance. The (minor) drawback of the second and the fourth methods, is that they require to pre-compute, store, and handle distribution functions (both second and fourth) and density functions (the fourth). However, as discussed earlier, this overhead can be easily handled. The first and the third methods do not need any pre-analysis of the data sets and do not require any other storage overhead, however their performance is worse that that of the other two methods and they are also more difficult to be used since, as we said when we discussed in details their results, there is not a clear dependency between the threshold and the improvement of efficiency in the various data sets, when k varies. In conclusion it appears that the most promising method is the fourth one, since it offers very high performance for nearest neighbor queries and it also does a good job in case of range queries.

6.11 Comparison with other techniques

In Section 5.3 we have described the most significant approaches for approximate similarity search, which belong to the category of algorithms reducing the examined result set [AM95, AMN+98, PAL99, PL99, CP00]. In this section we make some performance and qualitative comparisons of our proposed techniques with these techniques.

Notice that a rigorous performance comparison cannot be assessed by using the performance figures reported by the articles describing these techniques. In fact, typically different data sets were used for the specific experiments, different underlying access methods were used, in some cases the efficiency was measured by considering the computational complexity in terms of floating point operations, and different measurements for determining the accuracy were performed. However we are able to make some informal comparison anyway. For what concerns the efficiency, the improvement provided by these techniques is generally lower than that provided by our techniques. Specifically, no other techniques offer improvement of efficiency analogous to that offered by our best techniques, namely the second and the fourth proposed approaches. For what concerns the accuracy, given that some of these methods used as indication of the accuracy the relative distance error, whose use we have contested in Section 6.4, we have also performed some preliminary tests using it, to have similar accuracy figures, and our results confirm that the trade-off between improvement of efficiency and loss of accuracy is generally more advantageous in our cases.

The behaviour of our algorithms is more general than the other techniques. We offer wider applicability since our techniques were defined for generic metric spaces and some of them can be used both for nearest neighbors and range queries. In the following we discuss specifically the limits of the other techniques.

The technique for nearest neighbor searching using BBD trees (see Section 5.3.3 and [AMN+98]) can only be used with data represented in vector spaces and distances measured by using functions of the Minkowski family. Its performance depends exponentially on *dim*. As a consequence the algorithm can be practically used when the vector space has number of dimensions in the range from 2 to 20. For vector spaces

with higher dimension this approach suffer from the dimensionality curse.

The technique for approximate range queries using BBD trees (see Section 5.3.4 and [AM95]), solves the counting version of the range search problem in vector spaces when distances are Minkowski distances. Specifically it can only be used to count or compute the weight of objects contained in the result set of an approximate range query. It does not retrieve the set of objects qualifying for the approximate range query. The counting problem is generally easier to be addressed than the problem of obtaining the set of objects qualifying for a range query, since it might be solved without accessing the, otherwise needed, leaf nodes of the tree, where pointers to real objects are stored. Performance of this algorithm is moderate.

The techniques based on the angle property (see Section 5.3.5 and [PAL99, PL99]) are only able to answer to nearest neighbors search in vector spaces and Minkowski distances. The approach described in [PAL99], in its original definition, cannot be tuned to trade performance with quality of results, in fact it does not use any approximation parameter that can be used by the user to chose the desired degree of approximation. The performance obtained by these algorithms is generally low compared to ours.

The PAC nearest neighbor algorithm (see Section 5.3.6 and [CP00]) can be used in generic metric spaces. However in its original definition it is limited to the case of just a single nearest neighbor search and fails to handle both generic nearest neighbors and range queries. It offers moderate performance, limited to the case of one nearest neighbor search.

Summarizing, with the exception of the PAC nearest neighbor searching technique all other techniques can only be used in vector spaces and distances should be

measured using functions of the Minkowski family. In addition all of them, with the exception of the technique for approximate range queries using BBD trees, can only be used for nearest neighbors queries and in some cases just for one nearest neighbor queries. To the best of our knowledge, there are no techniques that support at the same time range and nearest neighbor queries in generic metric spaces.