14th International Conference on Data Engineering (ICDE'98), February 23-27, 1998, Orlando, Florida.

Fast Nearest Neighbor Search in High-dimensional Space

Stefan Berchtold†, Bernhard Ertl, Daniel A. Keim‡, Hans-Peter Kriegel, Thomas Seidl
Institute for Computer Science, University of Munich, Oettingenstr. 67, 80538 Munich, Germany
{berchtol, ertl, keim, kriegel, seidl}@dbs.informatik.uni-muenchen.de

† current address: AT&T Labs Research, berchtol@research.att.com
‡ current address: University of Halle-Wittenberg, keim@informatik.uni-halle.de

Abstract

Similarity search in multimedia databases requires an efficient support of nearest-neighbor search on a large set of high-dimensional points as a basic operation for query processing. As recent theoretical results show, state of the art approaches to nearest-neighbor search are not efficient in higher dimensions. In our new approach, we therefore precompute the result of any nearest-neighbor search which corresponds to a computation of the voronoi cell of each data point. In a second step, we store the voronoi cells in an index structure efficient for high-dimensional data spaces. As a result, nearest neighbor search corresponds to a simple point query on the index structure. Although our technique is based on a precomputation of the solution space, it is dynamic, i.e. it supports insertions of new data points. An extensive experimental evaluation of our technique demonstrates the high efficiency for uniformly distributed as well as real data. We obtained a significant reduction of the search time compared to nearest neighbor search in the X-tree (up to a factor of 4).

1 Introduction

An important research issue in the field of multimedia databases is the content based retrieval of similar multimedia objects such as images, text and videos [Alt+ 90] [Fal+ 94] [Jag 91] [MG 93] [SBK 92] [SH 94]. However, in contrast to searching data in a relational database, a content based retrieval requires the search of similar objects as a basic functionality of the database system. Most of the approaches addressing similarity search use a so-called feature transformation which transforms important properties of the multimedia objects into high-dimensional points (feature vectors). Thus, the similarity search corresponds to a search of points in the feature space which are close to a given query point and, therefore, corresponds to a nearest neighbor search. Up to now, a lot of research has been done in the field of nearest neighbor search in high-dimensional spaces [Ary 95] [Ber+ 97] [HS 95] [PM 97] [RKV 95].

Most of the existing approaches solving the nearest neighbor problem perform a search on a priori built index while expanding the neighborhood around the query point until the desired closest point is reached. However, as recent theoretical results [BBKK 97] show, such index-based approaches must access a large portion of the data points in higher dimensions. Therefore, searching an index by expanding the query region is, in general, inefficient in high dimensions. One way out of this dilemma is exploiting parallelism for an efficient nearest neighbor search as we did in [Ber+ 97].

In this paper, we suggest a new solution to sequential nearest neighbor search which is based on precalculating and indexing the solution space instead of indexing the data. The solution space may be characterized by a complete and overlap-free partitioning of the data space into cells, each containing exactly one data point. Each cell consists of all potential query points which have the corresponding data point as a nearest neighbor. The cells therefore correspond to the d-dimensional Voronoi cells [PS 85]. Determining the nearest neighbor of a query point now becomes equivalent to determining the Voronoi cell in which the query point is located. Since the Voronoi cells may be rather complex high-dimensional polyhedra which require too much disk space when stored explicitly, we approximate the cells by minimum bounding (hyper-)rectangles and store them in a multidimensional index structure such as the X-tree [BKK 96]. The nearest neighbor query now becomes a simple point query which can be processed efficiently using the multidimensional index. In order to obtain a good approximation quality for high-dimensional cells, we additionally introduce a new decomposition technique for high-dimensional spatial objects.

The paper is organized as follows: In section 2, we introduce our new solution to the nearest neighbor problem, which is based on approximating the solution space. We formally define the solution space as well as the necessary cell approximations and outline an efficient algorithm for determining the high-dimensional cell approximations. In section 3, we then discuss the problems related to indexing the high-dimensional cell approximations and introduce our solution which is based on a new decomposition of the approximations. In section 4, we present an experimental evaluation of our new approach using uniformly distributed as well as real data. The evaluation unveils significant speed-ups over the R*-tree- and X-tree-based nearest neighbor search.

2 Approximating the Solution Space

Our new approach to solving the nearest neighbor problem is based on precalculating the solution space. Precalculating the solution space means determining the Voronoi diagram (cf. figure 1a) of the data points in the database. In the following, we recall the definition of Voronoi cells as provided in [Roo 91].

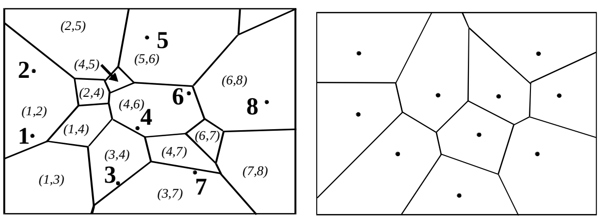
Definition 1. (Voronoi Cell, Voronoi Diagram)
Let *DB* be a database of points. For any subset $A \in DB$ of size $m := |A|$, $1 \le m < N$, and a given distance function $d: \Re^d \times \Re^d \to \Re_0^+$, the *order-m Voronoi Cell* of *A* is defined as

$$VoronoiCell(A) :=$$
$$\{x \in \Re^d \,|\, \forall (P_i \in A)\ \forall (P_j \in DB \backslash A){:}\ d(x, P_i) \le d(x, P_j)\}\ .$$

The *order-m Voronoi diagram* of *DB* is defined as

$$VoronoiDiagram_m(DB) :=$$
$$\{VoronoiCell(A) \,|\, A \subset DB \wedge |A| = m\}\ .$$

Note that we are primarily interested in the nearest neighbor of a query point, and that we assume the Voronoi cells to be bounded by the data space (*DS*). Therefore, in the following we only have to consider bounded Voronoi cells of order 1, which are also called NN-cells (cf. figure 1b).



a. Voroni Diagram of order 2 (cf [PS 85])   b. NN-Diagram

Figure 1: Voronoi diagram and NN-diagram

Definition 2. (NN-cell, NN-Diagram)
For any Point $P \in DB$ and a given distance function $d: \Re^d \times \Re^d \to \Re_0^+$, the *NN-cell* of *P* is defined as

$$NN\text{-}Cell(P) :=$$
$$\{x \in DS \,|\, \forall (P' \in DB \backslash \{P\}){:}\ d(x, P) \le d(x, P')\}\ .$$

The *NN-Diagram* of a database of points *DB* is defined as

$$NN\text{-}Diagram(DB) := \{NN\text{-}Cell(P) \,|\, P \in DB\}\ .$$

According to Definition 2, the sum of the volumes of all NN-cells is the volume of the data space (cf. figure 1b):

$$\sum_{i=1}^{N} Vol(NN\text{-}Cell_i) = Vol(DS)\ .$$

If we are able to efficiently determine the NN-cells, to explicitly store them, and to directly find the NN-cell which contains a given query point, then the costly nearest neighbor query could be executed by one access to the corresponding NN-cell. In general, however, determining the NN-cells is rather time consuming and requires (at least) $\Omega(N \log N)$ for $d \in \{1, 2\}$ and $\Omega(N^{\lceil d/2 \rceil})$ for $d \ge 3$ in the worst case [PS 85] for an Euclidean distance function. In addition, the space requirements for the NN-cells (number of *k* faces of the NN-diagram) are

$$O(N^{min(d+1-k, \lceil d/2 \rceil)}) \qquad for\ \ 0 \le k \le d-1$$

in the worst case [Ede 87], making it impossible to store them explicitly. For a practicable solution, it is therefore necessary to use approximations of the NN-cells, which is a well-known technique that has been successfully used for improving the query processing in the context of geographical databases [BKS 93]. In principle, any approximation such as (hyper-)rectangles, rotated (hyper-)rectangles, spheres, ellipsoids, etc. may be employed. In our application, we use an approximation of the NN-cells by minimum bounding (hyper-)rectangles and store them in a multidimensional index structure such as the X-tree [BKK 96]. The nearest neighbor query can then be executed by a simple and very efficient point query on this index. In the following, we define the approximation of the NN-cells.

Definition 3. (MBR-Approximation of NN-cells)
The MBR approximation $Appr_{MBR}$ of a nearest neighbor cell (*NNC*) is the minimum bounding (hyper-)rectangle $MBR = (l_1, h_1, \ldots, l_d, h_d)$ of *NNC*, i.e. for $i = 1, \ldots, d$:

$$l_i = min\{p_i | p \in NNC\} \text{ and } h_i = max\{p_i | p \in NNC\}\ .$$

Let us now consider examples of the NN-cells and their MBR-approximations for a number of different data distributions. Figure 2a and b show the NN-diagram and the corresponding approximation diagram for two independent uniformly distributed dimensions, figure 2c and d show the two diagrams for a regular multidimensional uniform distribution, and figure 2e and f show the diagrams for a sparse distribution. A uniform distribution is usually generated by using a random number generator to produce the data values for each of the dimensions independently. This generation process produces a data set which - projected onto each of the dimension axes - provides a uniform distribution. It does not mean, however, that the data is distributed uniformly in multidimensional space, i.e. for a partitioning of the data space into cells of equal size that each

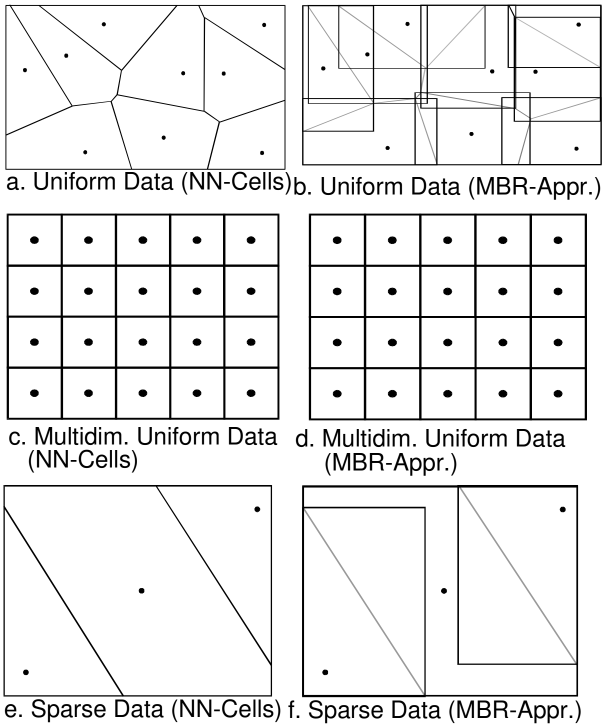


Figure 2: NN-cells and their MBR-approximations

of the cells contains an equal number of data points. A distribution which fulfills the latter requirement is called a multidimensional distribution.

Note that the regular multidimensional uniform distribution corresponds to the best case for our approach and the sparse distribution corresponds to the worst case: In case of the regular multidimensional uniform distribution, the MBR approximations are identical with the NN-cells, which means that the approximations do not have any overlap and therefore a point query on the index accesses only one page. In contrast, in case of the sparse distribution almost all approximations are identical with the *DS* which means that the approximations completely overlap and a point query on the index accesses most data pages.

Let us now consider the space and time complexity of our approximations: The MBR-approximation of a NN-cell requires $2 \cdot d$ space, which means that we need a total of twice the size of the database for all points $(2 \cdot d \cdot N)$. Determining the approximation of a Voronoi cell can be seen as a typical linear programming problem: We have a number of linear constraints (at most N-1) and try to find the maximum extension of the NN-cell in $2 \cdot d$ directions, which directly correspond to the borders of the bounding box. The time needed to determine the approximations is therefore the time of $2 \cdot d$ linear programming runs with at most $N-1$ linear constraint.

There are many well-known solutions to linear programming problems. The most widely used approach is the Simplex method [Dan 66]. The simplex method is based on the observation that - if a solution exists - it must be a corner point of the solution space. The basic idea of the simplex method is to start with a valid (but potentially sub-optimal) solution and then move along the boundary of the polyhedra of valid solutions to find the corner point which is optimal according to optimization function. A problem of the simplex method is the first step, namely finding a valid starting point. A revised version of the simplex method which avoids this problem is the algorithm of Best and Ritter [BR 85]. The complexity of the algorithm is $O(n^{\lceil d/2 \rceil})$ in the worst case where *n* is the number of points considered in determining the approximation [PS 85]. In the average case, the complexity is $O(n \cdot d!)$ [Sei 90].

One problem of all linear programming algorithms is that they usually need to consider the constraints resulting from all *N* points in the database (*n*=*N*), making the linear programming prohibitively expensive for large databases. An important observation, however, is that usually only a small number of points actually contributes to the maximum extension of the NN-cell. In general, we may therefore restrict the number of points while still obtaining the correct extension of the NN-cell. Since we are only interested in an approximation of the NN-cell, we may even omit data points which contribute to the NN-cell, thereby loosing the correct approximation of the NN-cell. This, however, is acceptable since the determined approximation may only become larger than the correct approximation which means that we do not get false dismissals and therefore do not compromise the correctness of our approach (cf. Lemma 1).

Based on this observation, our optimized algorithms for calculating the approximations accept slightly suboptimal approximations in exchange for reducing the number of points examined in the linear programming considerably. For determining the points which are used in the linear programming, we use an index-based search for a number of points which are close to the considered point. Closeness may be defined by a multidimensional point or sphere query (from experiments, we obtained $radius = 2 \cdot \sqrt[d]{1/n}$ as a good heuristic value) which can be executed efficiently using the multidimensional index. In our experiments on uniformly distributed data, the approach provides very good results, i.e. the determined approximations are close to the correct approximations. On real data, however, there is a much higher variation: The number of points resulting from the multidimensional point or sphere query varies in a wide range, which is a result of the high clustering of the real data. The largely differing numbers of points considered in the linear programming results in a high variance of the quality of the determined approximations and also of the time needed for determining the approximations. In the worst case, the number of data points resulting from a point or sphere query is in the order of *N*, which means that the complexity of the algorithm is similar to the complexity of the correct algorithm. For real data distributions, we therefore developed other heuristics for determining the relevant points. A heuristic which provides good results for real data is to use a constant number of points ($4 \cdot d$), namely the $2 \cdot d$ nearest neighbor points in all directions and the $2 \cdot d$ points which have the smallest deviation from the orthogonal axes in all directions. For this heuristic, the complexity of the linear programming algorithm becomes $O(d!)$ since the number of considered points is constant $(n = 4 \cdot d)$. In our experiments, the average performance of this algorithm turned out to be rather good. Figure 3 presents the insertion algorithm including our optimized algorithms for determining the NN-cells. We name the four possibilities for determining the data points which are used in the linear programming as follows:

Correct (all *N* points are considered)

Point (all points of which the rectangle in the index contains the point)

Sphere (all points of which the rectangle in the index intersects the sphere)

NN-Direction (*2d* NN-points in all directions and *2d* points with smallest deviation from the orthogonal axes)

```
Tree::insert (DataPoint DP, Dim D; OptAlg Alg)
{  SetOfPoints p_set;
   MBR mbr;
   switch(Alg)
   {  case Point:
              p_set = PointQuery(DP); break;
      case Sphere:
              p_set = SphereQuery(DP); break;
      case NN-Direction:
              for (i = 1; i != d; i++)
              { p_set->add(NNDimQuery(DP, i));
              p_set->add(NNAxesQuery(DP, i));
              };  break;
      default:
              p_set = AllPoints;
   }
   for (i = 1; i != d; i++)
   {  mbr[2*i-1] = LinOpt(DP, p_set, i, left);
      mbr[2*i] = LinOpt(DP, p_set, i, right);
   }
   insert(mbr);
}
```

Figure 3: Insertion algorithm

In figure 4, we show the results of the experimental comparison. Figure 4a shows the performance (time needed to calculate the approximations) which directly corresponds to the insertion time and figure 4b shows the quality

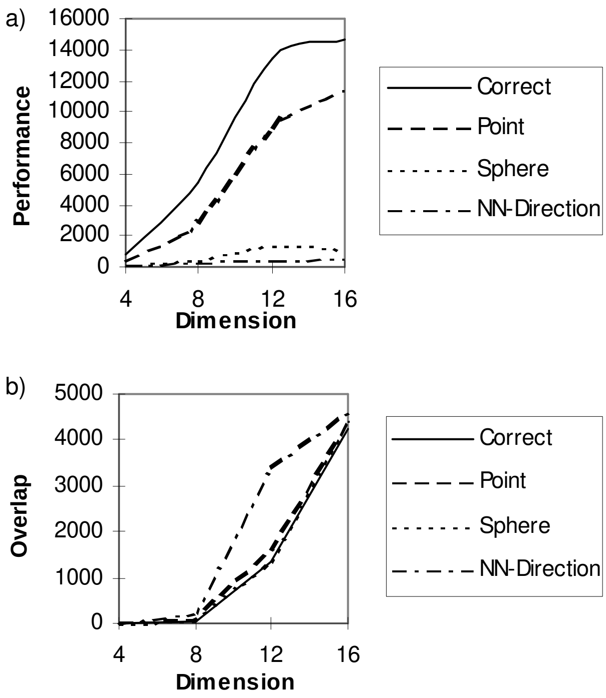


Figure 4: Comparison of the four algorithms

of the linear programming (overlap of the approximations) depending on the dimensionality *d* of the data.As expected, independent from the strategy used for determining the points, the time needed to compute the approximations increases with the dimension and the quality of the approximations decreases with the dimension (i.e., the overlap of the approximations increases). Note that the most accurate algorithm (Correct) has the poorest performance and that the least accurate algorithm (NN-Direction) has the best performance. Obviously, there is a trade-off between index creation and query execution time. One may choose to spend more time in creating the index and save time in executing the query or save time in creating the index and spend more time in executing the query. Note, however, that the determination of the approximations is only done once at index creation time whereas at query execution time, only a point query on the index has to be performed.

To obtain an evaluation criterion which takes both — accuracy and performance —into account, we may consider the quality-to-performance ratio. In figure 5, we present the four quality-to-performance curves. Note that for lower dimensions (4 and 8) the Sphere algorithm provides the best quality-to-performance ratio and for higher dimensions (12 and 16), the NN-Direction algorithm.

For the correctness of our approach, it is important that the use of the three optimized algorithms (Point, Sphere, NN-Direction) to determine the approximations do not in-

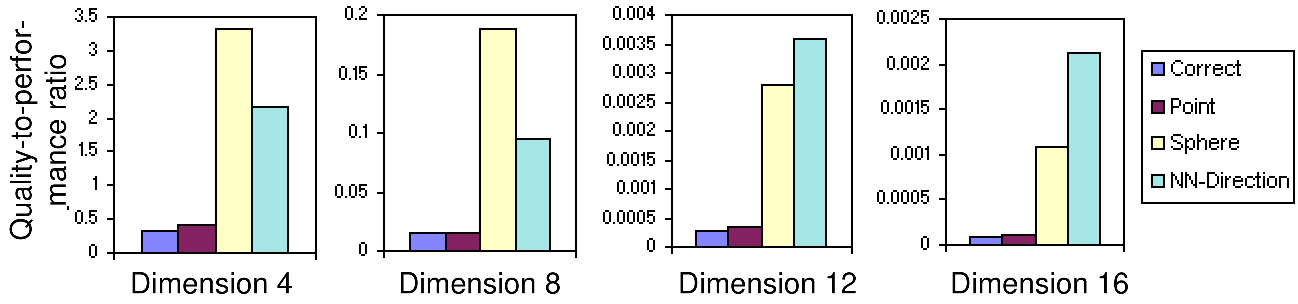


Figure 5: Quality-to-performance ratio of the four algorithms

duce false dismissals. In the following lemma, we show that the approximations determined by our algorithms may only be larger than the correct approximations, which implies that a point query on an index created by our three optimized algorithms provides a superset of the results obtained from the same point query on an index that is created using the correct approximations. Lemma 1 is used later in Lemma 2 to show the overall correctness of our approach.

Lemma 1 *(Correctness of Optimized Algorithms)*

For an arbitrary point set DB, $|DB| = N$, the following observation holds:

$$\forall P \in DB \;\; \forall Alg \in \{Point, Sphere, NN\text{-}Direction\} :$$

$$Appr^{Correct}(P) \subseteq Appr^{Alg}(P) .$$

Proof. The correct approximation $Appr^{Correct}(P)$ of a point P is an MBR approximation $(l_1, h_1, \ldots, l_d, h_d)$. Each of the l_i and h_i $(1 \le i \le d)$ results from a linear programming using $N-1$ constraints. To show Lemma 1, we have to show that the approximation $Appr^{Alg}(P) = (l_1', h_1', \ldots, l_d', h_d')$ resulting from any of the other algorithms provides a possibly larger MBR, i.e.

$$\forall i, 1 \le i \le d: \;\; l_i' \le l_i \;\wedge\; h_i' \ge h_i$$

Without loss of generality, we consider an arbitrary point *P* and an arbitrary dimension *j*. All following consideration analogously apply to all other points and dimensions. Let us now consider the simplex algorithm which is used to determine the l_j and h_j. The principle idea of the revised simplex method is to determine the corner of the polyhedra of valid solutions, which is optimal according to the optimization function. The only difference between the correct algorithm and the *Point*, *Sphere* and *NN-Direction* algorithms is that the number of constraints used by the optimized algorithms is a subset of the constraints of the correct algorithm. As a result, the polyhedra of valid solutions of the correct algorithm $PH^{Correct}$ is contained in the polyhedra of valid solution for any of the optimized algorithms PH^{Alg}. Due to the main theorem of linear programming theory, the l_j and h_j have to be corner points of the polyhedra and therefore, from $PH^{Correct} \subseteq PH^{Alg}$ follows that $l_j' \le l_j$ and $h_j' \ge h_j$. q.e.d.

Until now, we have only dealt with the case of a static database which is obviously not realistic for real databases. Our algorithms, however, also work for the dynamic case. We only have to dynamically update the approximations of the NN-cells which are affected by the update operation. In case of an insertion operation, we even do not need to find all NN-cells since the existing NN-cells may only become smaller by an insertion. We are therefore able to use a sphere query with the new point as center and update all NN-cells that are intersected by the sphere. For deletions and modifications, efficient algorithms to dynamically update the NN-cells have been proposed by Roos [Roo 91]. His dynamic algorithms to calculate the NN-cells may be applied to our approach to obtain sufficiently efficient solutions for the dynamic case.

3 Indexing the Solution Space

One problem of our approach as introduced so far is that the overlap of the approximations rapidly increases with the dimension even for the correct approximations (cf. figure 4b). A direct result of the increasing overlap is that the query processing time is also increasing with the dimension. In this section, we introduce a solution for this problem, which reduces the overlap and is based on a decomposition technique. The concept of decomposing objects to improve the query processing has originally been proposed in [KHS 91] [SK 93] for improving the query processing in geographical databases. The decompositions proposed in [KHS 91] are based on decomposing the objects into triangles, trapezoids, convex polyhedra, and combinations (heterogeneous decompositions). None of those decompositions, however, is directly applicable to the high-dimensional case. In the following, we are going to examine the specific properties of our high-dimensional

NN-cells and we then use those properties to define an adequate decomposition.

Let us first recall our example provided in figure 2c and d, showing a regular multidimensional uniform data distribution. As already mentioned, using our approach to index the resulting approximations is optimal since the NN-cells coincide with the MBRs. Consequently, our approach accesses only one data page during the nearest neighbor search because every query point is located in exactly one candidate approximation which is identical to the corresponding voronoi cell.

Unfortunately — with few exceptions — real multidimensional data do not correspond to a perfect multidimensional uniform distribution. In most cases, the data is closer to a high-dimensional sparse distribution. The reason is that the high-dimensional space cannot be filled completely and clusters are likely to occur. As indicated in figure 2e and f, the MBR approximations of the NN-cells induce a high degree of overlap for sparse data. The problem is that the volume of the approximations is by far larger than the volume of the NN-cells. For a sparsely populated high-dimensional space, the volume may even become as high as $N \cdot |DS|$ in the worst case, which means that we have to access all of the database even for a simple point query. What is required is a decomposition which minimizes the volume of the approximations. We are therefore looking for a decomposition which provides the minimum volume among all possible decompositions. In the following, we formally define the terms decomposition and optimal decomposition.

Definition 4. (Decomposition, Optimal Decomposition)
A decomposition of a NN-diagram consisting of NN-cells $NNC_j \ (j = 1 \ldots N)$ is a partitioning of each of the cells into k partitions $\{DC_{j1}, \ldots, DC_{jk}\}$ such that

$$\forall\, j \in \{1 \ldots N\}\colon\ NNC_j = \bigcup_{i=1}^{k} DC_{ji} \ \wedge$$

$$\wedge\ \ \forall\, i, \hat{\imath} \in \{1 \ldots k\}, i \neq \hat{\imath}\colon\ \ DC_{ji} \cap DC_{j\hat{\imath}} = \varnothing\,.$$

The decomposition is called *optimal* iff

$$\sum_{j=1}^{N} \sum_{i=1}^{k} Vol(Appr_{MBR}(DC_{ji})) \text{ is minimal.}$$

Determining the optimal decomposition requires examining all possible decompositions, which is prohibitively expensive. We therefore use a heuristic which provides good results especially for real high-dimensional data distributions. Our heuristic is based on two observations: The first observation is that for high-dimensional data, it is not possible to decompose the NN-cells in all dimensions since this would result in a high number of MBR approximations, which is exponential in the number of dimensions. The second observation is that we obtain the smallest volume if we decompose the NN-cells in those dimensions, in which the NN-cells are most oblique. Many algorithms could be used, and one possibility is to choose the maximum of all shortest diagonals. Figure 6 provides an example: We show the approximations of a NN-cell (cf.

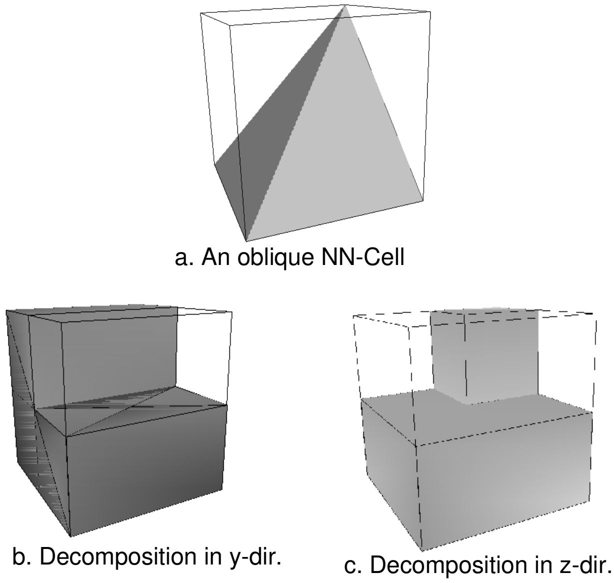


Figure 6: Decompositions of NN-cells and their approximations

figure 6a) after decomposing the NN-cell in y-direction (cf. figure 6b) and in z-direction (cf. figure 6c). In the example, we use a decomposition into two partitions ($k = 2$). The decomposition in the more oblique dimension (z-direction) provides a lower overall volume of the approximations and therefore also a much lower overlap. The idea of our decomposition approximations is to determine the most oblique dimensions and to use a simple linear decomposition in those dimension. In the following, we assume that we want to decompose the NN-cells in the d' most oblique dimensions $(d' \leq 7)$. Without loss of generality, we assume that the d' most oblique dimensions are the first d' dimensions $(d_1, \ldots, d_{d'})$. The NN-cells are decomposed in each of those dimensions. The number of partitions used for each of the dimensions is chosen depending on their obliqueness. If the dimensions $d_1, \ldots, d_{d'}$ are sorted with an decreasing obliqueness, the number of partitions used in each of the dimensions $(n_1, \ldots, n_{d'})$ is also decreasing $n_1 \geq \ldots \geq n_{d'}$. The resulting total number of partitions (k) is

$$k = \prod_{i=1}^{d'} n_i\,.$$

For practical purposes, k has to be rather small $(k \leq 100)$ since otherwise the number of index entries becomes very large. If the n_i are assumed to be constant, then for $d' = 5, 6, 7$ the n_i has to be chosen as 2, for $d' = 4$ the n_i has to be chosen as 3, for $d' = 3$ the n_i can be at most 4, and for $d' = 2$ the n_i may be less or equal to 10. Let us now formally define our MBR decomposition:

Definition 5. (MBR Decomposition)
Let the minimum bounding (hyper-)rectangle $MBR_j = (l_1, h_1, \ldots, l_d, h_d)$ be the MBR approximation $Appr_{MBR}(NNC_j)$ of the NN-cell NNC_j. Then, the decomposition the NN-cell NNC_j is defined as the set $\{DC_{j1}, \ldots, DC_{jk}\}$ with

$$DC_{ji} = NNC_j \cap MBR_{ji}$$

The MBR_{ji} $(i = 1 \ldots k)$ are defined as

$$MBR_{ji} = \Big(l_1 + i_1 \cdot \frac{h_1 - l_1}{n_1}, l_1 + (i_1 + 1) \cdot \frac{h_1 - l_1}{n_1}, \ldots,$$

$$l_{d'} + i_{d'} \cdot \frac{h_{d'} - l_{d'}}{n_{d'}}, l_{d'} + (i_{d'} + 1) \cdot \frac{h_{d'} - l_{d'}}{n_{d'}}, l_{d'+1}, h_{d'+1}, \ldots, l_d, h_d\Big)$$

where the i_j $(0 \le i_j \le n_j, \quad j = 1 \ldots d')$ are determined sequentially as the maximum values such that finally

$$\sum_{j=1}^{d'} i_j \cdot n_j^{(d'-j)} = i \,.$$

Our definition of an MBR decomposition fulfills the properties of decompositions as defined in Definition 4. The completeness property is guaranteed since

$$\forall\, j \in \{1 \ldots N\}: \bigcup_{i=1}^{k} MBR_{ji} = Appr_{MBR}(NNC_j)$$

and therefore,

$$\forall\, j \in \{1 \ldots N\}: \bigcup_{i=1}^{k} DC_{ji} = NNC_j \,.$$

The disjointedness of the DC_{ji} is guaranteed since the MBR_{ji} are defined as disjoint (hyper-)rectangles and therefore, the $DC_{ji} = NNC_j \cap MBR_{ji}$ are also disjoint.

In the following, we show the correctness of our approach, i.e. our approach does not induce false dismissals. The correctness includes the proof that the use of approximations and the use of decompositions do not exclude the correct solution.

Lemma 2 *(Correctness of our Approach)*
Our approach does not induce false dismissals, i.e. for an arbitrary point set *DB*, $|DB| = N$, a point query on an index containing the approximations of the decomposed NN-cells $\{DC_{j1}, \ldots, DC_{jk}\}, j = 1 \ldots N$ provides a result set RS containing the nearest neighbor *NN*.

Proof. To show the correctness of our approach, we have to show that the following three steps do not induce false dismissals:

1. Usage of approximations of the NN-cells.
2. Usage of the optimized algorithms for determining the NN-cells.
3. Usage of decomposition of the NN-cells.

The correctness of the first step is easy to show: Since $NNC_j \subseteq Appr_{MBR}(NNC_j)$, the result set *RS* of a point query contains at least the correct NN-cell and therefore the usage of approximations does not induce false dismissals. The correctness of the second step has already been shown in Lemma 1. The correctness of the third step can be shown as follows: According to Definition 4,

$$\forall\, j \in \{1 \ldots N\}: NNC_j = \bigcup_{i=1}^{k} DC_{ji}$$

and from $DC_{ji} \subseteq Appr_{MBR}(DC_{ji})$, it follows that

$$\bigcup_{i=1}^{k} DC_{ji} \subseteq \bigcup_{i=1}^{k} Appr_{MBR}(DC_{ji})$$

and therefore

$$NNC_j \subseteq \bigcup_{i=1}^{k} Appr_{MBR}(DC_{ji}) \,.$$

Since all approximations of the decomposed NN-cells $Appr_{MBR}(DC_{ji})$ are in the index, the result set *RS* of a point query contains at least the correct NN-cell and, therefore, the usage of our decomposition of the NN-cells does not induce false dismissals. q.e.d.

4 Experimental Evaluation

To show the practical relevance of our approach, we performed an extensive experimental evaluation of the NN-cell approach and compared it to the R*-tree [BKSS 90] as well to as the X-tree [BKK 96]. All experimental results presented in this sections are computed on an HP720 workstation with 48 MBytes of main memory and several GBytes of secondary storage. All programs have been implemented in C++. The test data used for the experiments are real point data consisting of Fourier points in high-dimensional space ($d = 8$) and synthetic data consisting of uniformly distributed points in high-dimensional space ($d = 4, 6, 8, 10, 12, 14, 16$). The block size used for our experiments is 4 KBytes, and all index structures were allowed to use the same amount of cache. For a realistic evaluation, we used very large amounts of data in our experiments.

First, we evaluated the NN-cell approach on synthetic databases with varying dimensionality. For the experiments presented in figure 7, we used 10,000 uniformly distributed data points of dimensionality $d = 4$ to $d = 16$. Figure 7 shows the total search time of the NN-cell approach and of a classic NN-search on the R*-tree and the X-tree. As expected, the search time increases with growing dimension. For lower dimensions, the total search time of NN-cell approach and R*-tree as well as X-tree is comparable but for higher dimensions, the NN-cell approach is much faster than both, the R*-tree and the X-tree. In figure 8, we show the speed-up of the NN-cell approach over the R*-tree which is increasing to more than 325% for $d = 16$. The high speed-up factors are caused by the fact that the R*-tree needs to access most of the directory and data pages, while the NN-cell approach takes advantage of the precalculation of the solution space. In figure 9, we show a more detailed comparison, namely the time needed for accessing the pages and the time needed for processing the queries. It turned out that the CPU-time and the page accesses of the NN-cell approach are both better than those of the R*-tree whereas in comparison with the X-tree, only the CPU-time is better. This effect may be explained by the fact that on the one side, the X-tree uses an overlap-free split strategy which minimizes the overlap in the directory and thereby reduces the number of page accesses. On the other side, however, the X-tree has to perform a more CPU-time consuming NN-query while the NN-cell approach only performs a simple point query. Note that in contrast to many other database operations, the total search time of NN-queries is not dominated by the number of page accesses since the nearest neighbor algorithm requires sorting the nodes according to the min-max distance; and therefore, the NN-cell approach provides a better total search time than the X-tree.

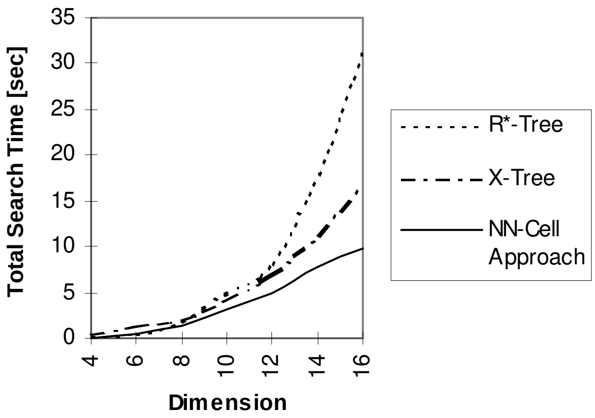


Figure 7: Comparison of R*-tree, X-tree, and NN-cell approach depending on the dimensionality

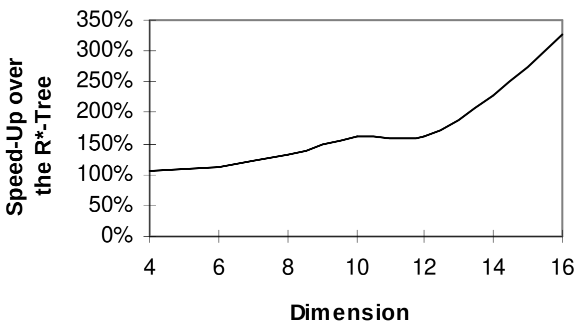


Figure 8: Speed-up of the NN-cell approach over the R*-tree depending on the dimensionality

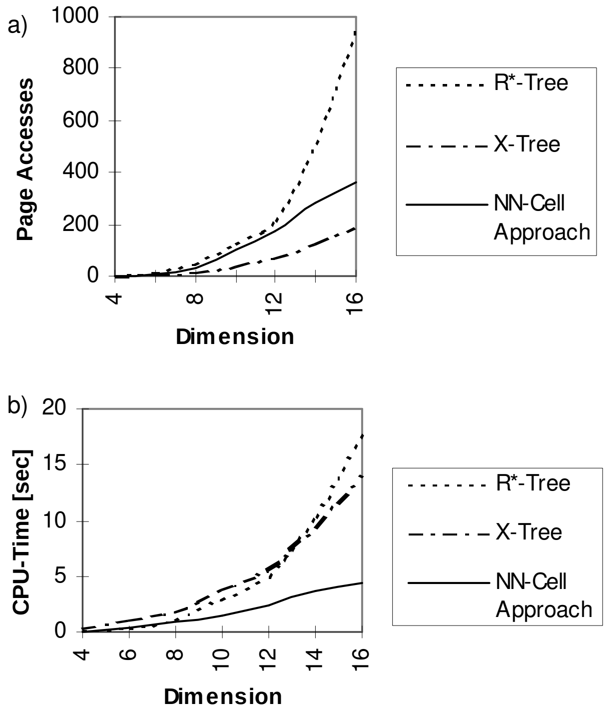


Figure 9: Number of page accesses versus CPU-time

In our next series of experiments, we compared the NN-cell approach with the R*-tree and the X-tree depending on the size of the database (i.e., the number of tuples). In figure 10, we show the total search time for d=10 depending on the size of the database which varied between N=5,000 and N=20,000. Again, the NN-cell approach performed significantly better than the R*-tree and the X-tree, and shows a logarithmic behavior in the number of database tuples.

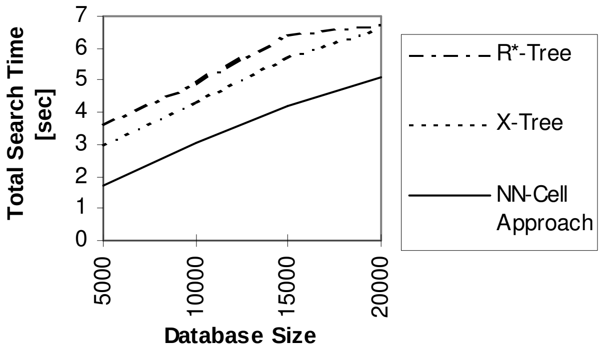


Figure 10: Comparison of NN-cell approach, R*-tree, and X-tree depending on the size of the database

Since one may argue that synthetic databases with uniformly distributed data are not realistic in high-dimensional space, we also used real data in our experiments. We had access to a databases containing high-dimensional Fourier points. Since the X-tree turned out to be consistently better than the R*-tree in our experiments, we only compared the NN-cell approach to the X-tree. The results of our experiments (cf. figure 11) show again a significant improvement

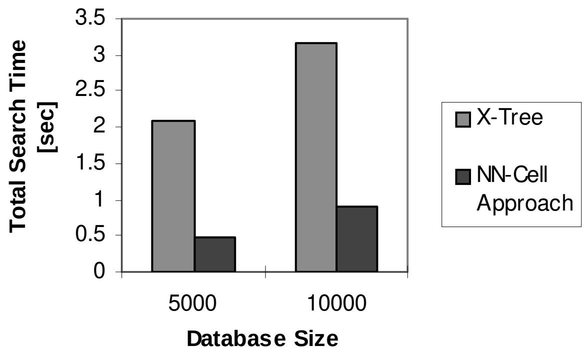


Figure 11: Comparison of NN-cell approach and X-tree on real data

of the NN-cell approach over the X-tree (speed-up of up to 425%). Comparing the page accesses and CPU-time needed for the Fourier database (cf. figure 12) revealed that the NN-cell approach now performed better in both categories, which is due to the fact that the approximations of the NN-cells turned out to be better than the approximations for the uniformly distributed data.

In a last experiment, we evaluated the impact of decomposing the approximations (cf. section 3). For our comparison, we used the most exact approximation algorithm for determining the approximations (*Correct*). As a measure for the quality of the approximations with and

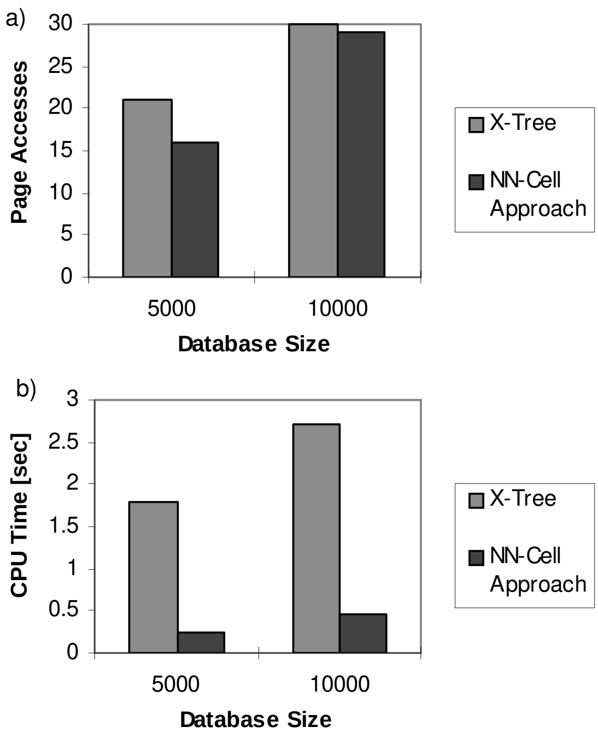


Figure 12: Number of page accesses versus CPU-time

without decomposition, again we used the average overlap of approximations which directly corresponds to the query performance. Figure 13 shows the results of our comparison which clearly reveals a significant improvement over the *Correct* algorithm for determining the approximations, which in addition increases with the dimensionality. Note that the quality improvement over the optimized algorithms (*Point*, *Sphere*, *NN-Direction*) is even higher.

5 Conclusions

In this paper, we proposed a new technique for efficient nearest neighbor search in a set of high-dimensional points. Our technique is based on the precomputation of the solution space of any arbitrary nearest-neighbor search. This corresponds to the computation of the voronoi cells of the data points. Since voronoi cells may become rather complex when going to higher dimensions, we presented a new algorithm for the approximation of high-dimensional voronoi cells using a set of minimum bounding (hyper-) rectangles. Although our technique is based on a precomputation of the solution space, it is dynamic, i.e. it supports insertions of new data points.

We finally showed in an experimental evaluation that our technique is efficient for various kinds of data and clearly outperforms the state of the art nearest-neighbor al-

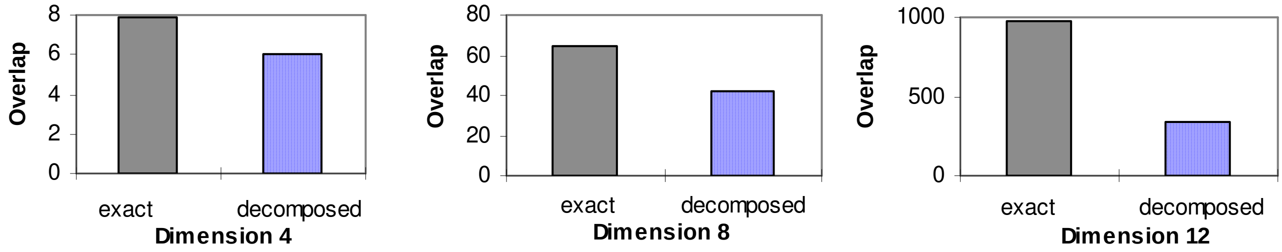


Figure 13: Effect of decomposing the NN-cells

gorithms. In our experiments, we obtained a significant reduction of the search time, up to a factor of 4. Our future research interests are focussed on the application of our technique to k-nearest neighbor search.

References

[Alt+ 90] Altschul S. F., Gish W., Miller W., Myers E. W., Lipman D. J.: *'A Basic Local Alignment Search Tool'*, Journal of Molecular Biology, Vol. 215, No. 3, 1990, pp. 403-410.

[Ary 95] Arya S.: *'Nearest Neighbor Searching and Applications'*, Ph.D. thesis, University of Maryland, College Park, MD, 1995.

[BBKK 97] Berchtold S., Böhm C., Keim D., Kriegel H.-P.: *'A Cost Model for Nearest Neighbor Search in High-Dimensional Data Spaces'*, Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS), Tucson, AZ, 1997, pp. 78-86.

[Ber+ 97] Berchtold S., Böhm C., Braunmüller B., Keim D., Kriegel H.-P.: *'Fast Parallel Similarity Search in Multimedia Databases'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Tucson, AZ, 1997, pp. 1-12. Best Paper Award.

[BKK 96] Berchtold S., Keim D., Kriegel H.-P.: *'The X-tree: An Index Structure for High-Dimensional Data'*, Proc. 22nd Conf. on Very Large Data Bases, Mumbai, India, 1996, pp. 28-39.

[BKS 93] Brinkhoff T., Kriegel H.-P., Schneider R.: *'Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems'*, Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 40-49.

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[BR 85] Best M. J., Ritter K.: *'Linear Programming. Active Set Analysis and Computer Programs'*, Englewood Cliffs, N.J., Prentice Hall, 1985.

[Dan 66] Dantzig G. B.: *'Linear Programming and Extensions'* (in German), Springer, Berlin, 1966.

[Ede 87] Edelsbrunner H.: *'Algorithms in Combinatorical Geometry'*, EATCS Monographs in Computer Science, Springer, Berlin, 1987.

[Fal+ 94] Faloutsos C., Barber R., Flickner M., Hafner J., et al.: *'Efficient and Effective Querying by Image Content'*, Journal of Intelligent Information Systems, 1994, Vol. 3, pp. 231-262.

[HS 95] Hjaltason G. R., Samet H.: *'Ranking in Spatial Databases'*, Proc. 4th Int. Symp. on Large Spatial Databases, Portland, ME, 1995, pp. 83-95.

[Jag 91] Jagadish H. V.: *'A Retrieval Technique for Similar Shapes'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 208-217.

[KHS 91] Kriegel H.-P., Horn H., Schiwietz M.: *'The Performance of Object Decomposition Techniques for Spatial Query Processing'*, Proc. 2nd Symp. on Large Spatial Databases, Zurich, Switzerland, 1991, in: Lecture Notes in Computer Science, Vol. 525, Springer, 1991, pp. 257-276.

[MG 93] Mehrotra R., Gary J. E.: *'Feature-Based Retrieval of Similar Shapes'*, Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 108-115.

[PM 97] Papadopoulos A., Manolopoulos Y.: *'Performance of Nearest Neighbor Queries in R-Trees'*, Proc. of the 6th International Conference on Database Theory, Delphi, Greece, 1997, LNCS 1186, pp. 394-408.

[PS 85] Preparata F. P., Shamos M. I.: *'Computational Geometry: An Introduction'*, Springer, 1985.

[RKV 95] Roussopoulos N., Kelley S., Vincent F.: *'Nearest Neighbor Queries'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 71-79.

[Roo 91] Roos T.: *'Dynamic Voronoi Diagrams'*, Ph.D. Thesis, University of Würzburg, Germany, 1991.

[SBK 92] Shoichet B. K., Bodian D. L., Kuntz I. D.: *'Molecular Docking Using Shape Descriptors'*, Journal of Computational Chemistry, Vol. 13, No. 3, 1992, pp. 380-397.

[Sei 90] Seidel R.: *'Linear Programming and Convex Hulls Made Easy'*, Proc. 6th Annual Symp. on Computational Geometry, Berkeley, CA, 1990, pp. 211-215.

[SH 94] Sawhney H., Hafner J.: *'Efficient Color Histogram Indexing'*, Proc. Int. Conf. on Image Processing, 1994, pp. 66-70.

[SK 93] Schiwietz M., Kriegel H.-P.: *'Query Processing of Spatial Objects: Complexity versus Redundancy'*, Proc. 3rd Int. Symp. on Large Spatial Databases, Singapore, 1993, in: Lecture Notes in Computer Science, Vol. 692, Springer, 1993, pp. 377-396.