The Pde-tree: an Access Method for Efficient Indexing of Points

Antigoni Manousaka

Department of Informatics, Aristotle University
54124 Thessaloniki, Greece

Abstract. Point indexing is used in several and important applications, for instance, in: spatial databases, GIS, image processing, medical image analysis and molecular biology. In such applications information is stored in the form of high-dimensional feature vectors. Similarity queries should be supported efficiently in these applications, which presume efficient dynamic indexing. In this paper, we describe an index-structure family, which is called Pde-tree. Pde-tree combines Data Partitioning and Space Partitioning characteristics with the use of a distance function and achieves two important characteristics: (i) disjointness of data regions within the index and (ii) the ability to use different distance functions during queries (i.e., different from the distance function that the index building was based upon). A member of this family, called Pde_r-tree, is implemented and evaluated experimentally. The detailed experiments examine several factors and illustrate the significant advantages of the proposed method.

1 Introduction

A number of scientific areas require the management of very large databases of high dimensional feature vectors. The high dimensional feature vectors represent points in a high dimensional space. These databases are not static and modifications, insertions and deletions must be supported. The most important use of current databases is the answer of various types of similarity queries. To support dynamic databases which efficiently answer similarity queries we have to equip the database with an appropriate index. In such an index the points are grouped in multidimensional rectangles or spheres in order for the neighboring points to be neighbors in the index independently of the insertion order.

A number of indexing approaches have been proposed to address the problem of time-efficient response to similarity queries (a summary is given in Section 2). One strategy is to split the multidimensional feature space to partitions (quadtree, K-D-B tree). In these structures, overlapping of feature space partitions is not allowed. The advantage of the no overlapping case is that during search queries, a single root-to-leaf path is always followed. Therefore, this case maintains the appealing characteristic of the B-tree structure. A different strategy is the partitioning of the data space using Minimum Bounding Rectangles (MBRs) or Minimum Bounding Spheres (MBSs) or a combination of the two. The advantage of this case is that unused space is not indexed; however, the property of no overlapping does not hold anymore.

In this paper, we describe an index-structure family, which is called Pde-tree. Pde-tree combines Data Partitioning and Space Partitioning characteristics with the use of a distance function and achieves two important characteristics: (i) disjointness of data regions within the index (i.e., no overlapping) and (ii) the ability to use different distance functions during queries (i.e., different from the distance function that the index building was based upon). Both these characteristics provide significant advantages during query execution. More importantly, the proposed approach defines an index family that, by following specific options, leads to efficient index structures for points. To examine the aforementioned argument, we describe the Pde_r-tree index, which resembles the R-tree but follows the options of the Pde index family.

The contributions of this paper are summarized as follows:

- a novel index family for point data, which combines the advantages of existing indexes and avoids their shortcomings,
- the description of the Pde_r-tree, which results from following specific options of the proposed index family, and
- detailed experimental results, which examine several factors of the Pde_r-tree and illustrate its good performance.

The rest of this paper is organized as follows. Section 2 summarizes the related work. In Section 3 we describe the outline of the proposed approach and the basic options of the Pde-tree family. The details on the algorithms (insertion, deletion, search) are given in Section 4, whereas Section 5 contains the specific options followed by the Pde_r-tree. Experimental results are given in Section 6, and finally, Section 7 draws the conclusion.

2 Related Work

One of the earliest proposed multidimensional index structures for rectangle data and points is the R-tree [1]. The R-tree is a height-balanced tree corresponding to a hierarchy of nested rectangles. A leaf is the minimum bounding rectangle that includes the data entries contained in it. A node is the rectangle determined by the minimum bounding rectangle of its children rectangles and coincides with the minimum bounding rectangle of the data entries contained in its lower leaves. The root node corresponds to the minimum bounding rectangle of the whole data entries. Several variants of the R-tree have been proposed [2, 3]. The R*-tree [3] is the most successful one. It introduced the forced reinsertion mechanism and is different from the R-tree in the insertion and the split algorithm. The R*-tree is not overlap-free. Therefore, the search time of a point query depends on the amount of overlap and is not determined by the height of the tree. The R+-tree [2] achieves the no overlapping property, however at the expense of storage efficiency, due to the replication it uses. The K-D-B-tree [4] is an index structure for multidimensional point data. It is similar to the B-tree and it is height-balanced. It is constructed by recursively dividing the feature space into subregions using coordinate planes. Nodes and leaves correspond to subregions. The K-D-B-tree is characterized by the disjointness among the nodes on the same tree level. A disadvantage of the K-D-B-tree is that the forced split, which occurs when a region of an intermediate node is divided, can cause the creation of nearly empty leaves and

nodes. This reduces the performance of the K-D-B-tree on range queries and nearest neighbor queries. The K-D-B-tree cannot ensure the minimum storage utilization. An advantage of the K-D-B-tree is the overlap-free structure. Therefore, the search time of a point query is determined by the height of the tree. The SS-tree [5] is an index structure designed for similarity indexing of multidimensional point data. Bounding spheres are used rather than bounding rectangles for the region shape. An advantage of using bounding spheres is that nearly half the storage is required, compared to bounding rectangles. This advantage permits almost doubling the fanout of nodes and reduces the height of trees. The SR-tree (Sphere/Rectangle-tree) [6] uses a combination of bounding spheres and bounding rectangles. The structure of the SR-tree is based on that of the R*-tree and the SS-tree, and corresponds to the nested hierarchy of regions. However, the distinctive feature of the SR-tree is that it specifies a region by the intersection of the bounding sphere and the bounding rectangle of underlying points.

3 Outline of the Proposed Approach

3.1 Options and Design Implementation

To organize the points in regions we have to choose between using rectangles, spheres, or partitions of other shape, such as arc partitions etc. In the case of uniformly distributed data, the use of spheres will always result in region overlapping. Overlapping is undesirable, as it increases significantly with high dimensionality, resulting to performance degradation.

In order to avoid the possible performance degradation due to region overlapping, the data have to be grouped in regions of such shape that would make possible the occupation of the entire feature space without region overlapping. The insertion of a new point, which is out of any existing data region leads to the expansion of one of them in order to include the new point. To prevent region overlapping as a result of the expansion, an *expansion area* corresponding to each region is employed, to define its maximum allowable expansion.

A particularly useful characteristic of structures like M-tree [7], is the employment of pre-calculated distances to prune sub-trees when range queries are executed. Structures employing such distance metrics require a small number of comparisons for excluding from further consideration nodes and leaves, which according to their corresponding distance value do not include data points that are potentially of interest to the given query. To endow the proposed structure with such pruning capabilities, the use of a distance metric for pruning is introduced. We propose the calculation of distances from a *static reference poin* t, which has the advantage that the distance for each data point is calculated once during insertion and remains unaltered, regardless of any future chances to the tree (e.g. leaf split, parent nodes split, etc.).

The proposed structure is designed for indexing points *(P)* in a multidimensional space, employs a distance metric *(d)*, and introduces the notion of expansion area *(e)*. Based on these characteristics, the proposed structure was named Pde-tree. The goal of the Pde-tree is to combine the advantages of the K-D-B tree, R*tree and SS-tree structures.

3.2 The Distance Function

The distance function employed in the Pde-tree does not affect the grouping of data point to leaves or the grouping of leaves to nodes. It is used only for ordering the data points assigned to a given leaf and the leaves or nodes assigned to a given node, for the purpose of facilitating the pruning procedure in range queries.

In the two-dimensional feature space and in particular in geographical information systems (GIS), the Euclidean distance is an obvious choice; however, in high-dimensional feature spaces the choice of the distance metric is not obvious, and the notion of similarity calculation is heuristic [8].

It the case where prior knowledge regarding the distance function that will be required by the users is available (as a result of the nature of the particular application), the distance function that should be calculated during insertions clearly coincides with the user-required one. On the other hand, if no such prior knowledge is available, the choice of a distance function will be based on the satisfaction of certain desirable properties of distance functions for high-dimensional spaces; both necessary and desirable properties of distance functions are discussed in the sequel.

A distance function $d(a, b)$ between two points a and b, must possess the following properties:

- symmetry $d(a, b) = d(b, a)$,
- non negativity $d(a, b) > 0$ for every pair of discrete a, b and $d(a, a) = 0$, and
- triangle inequality $d(a, b) \leq d(a, c) + d(c, b)$.

Minkowski functions have these properties and are described as:

$$d_p(a, b) = \left[\sum_{i=1,k} |a_i - b_i|^p \right]^{\frac{1}{p}}$$

where a, b are points of the k-dimensional space. If p=1, the Minkowski Metric is called the *first order Minkowski Metric* or *Manhattan distance* metric or *city block distance*, if p=2 the Minkowski Metric is the familiar Euclidean distance. Generally the following relation is true: $d_m(a, b) \geq d_n(a, b)$, where $m < n$.

A criterion to choose a distance function is the range of its possible return values. The chosen criterion is the *maximum distance* in the multidimensional unit space: the less the maximum distance is, the less the distance between neighboring points is and, consequently, the selection process in nearest neighbor queries is more difficult. For a unit k-dimensional space the maximum value of the first order Minkowski Metric is k, the maximum value of the Euclidean distance is $\sqrt{k}$, the maximum value of the p order Minkowski Metric is $\sqrt[p]{k}$. It is obvious that if the selection of the function is not imposed by the nature of the problem, taking into consideration the simplicity of the calculations and the wide range of return values, the first order Minkowski Metric is the most suitable one. The choice of distance function clearly influences the result of a query. The issue of executing a range query using a distance function different than the one used to create the tree will be discussed in the sequel.

3.3 Node Structure

The information contained in a node of the Pde-tree can be viewed as consisting of two parts. The first part contains the information referring to data regions and the distance from the reference point, while the second part contains information pertaining to the expansion regions. During insertion, deletion and exact search both parts are used. During range queries and nearest neighbor queries only the first part is used. The splitting of the index in two parts so that the information retrieval during similarity queries is not overburdened by the second part proved useful.

Evidently, the use of additional information within the nodes of the Pde-tree corresponds to a reduction of the available fanout. Nevertheless, an analogous approach has been followed by other index structures, like the SR-tree. Experimental results in the sequel show that the Pde-tree requires less node-storage overhead compared to existing approaches, like the SR-tree.

4 Algorithms of the Pde-tree

4.1 Insertion

The expansion area of the root is the whole multidimensional normalized feature space. The expansion area of the first leaf created, upon the insertion of the first point to the tree, is also the whole feature space. A point is inserted to the leaf, to the expansion area of which the point belongs. Starting from the root, the node [1] with the expansion area including the point is selected proceeding to the leaf.

If the leaf is not full, the new point is inserted to it. If the point is out of the data area of the leaf, the data area is expanded to include the point. If necessary, the data area of the parent node is also expanded recursively. If the leaf is full, the split algorithm is applied resulting in two new leaves. The split algorithm takes into consideration only the data region of the leaf. The plane that divides the data region is expanded to divide also the expansion area. The information of the replacement of the leaf by two new ones and the information of the plane dividing the expansion area are forwarded to the parent node.

When a child (leaf or node) of a node is substituted by two new children after a split, the node is updated as follows. If the node is not full the information pertaining to the child is deleted and the information of the new children is inserted. Subsequently the node is updated with the new information of the plane dividing its expansion area to the expansion areas of its children. If the data area of the node is modified the parent node is updated recursively.

If the node is full a split algorithm is applied. The first created plane of those dividing its expansion area to the expansion areas of the children of the node is used as the splitting plane. The algorithm described above for the partially full node is used to update the one of the two nodes that includes the child, which is being replaced. The procedure is applied recursively.

[1] Henceforth, by the term node we mean an internal node of the tree, whereas a leaf node is denoted as leaf.

In the case where the insertion is done in a random way, i.e., the points are not inserted ordered in any specific dimension (this case is the most expected), then the use of space is very efficient. In particular, when the insertion of points follows the same distribution (regardless of which this distribution is) during the lifetime of the database, no problem will be encountered with respect to the efficiency of space used. Since a plane partitions the database into two parts with equal number of points and the distribution remains the same, then at every instance the same plane will partition the feature space into two approximately equally populated parts and the tree will remain balanced with respect to this factor. However, when the points are inserted ordered in any dimension or ordered by their distance from a reference point, then space efficiency may be violated. We are going to examine the sensitivity of the structure to this in the sequel.

4.2 Deletion

A point is deleted by deleting it from the leaf. If the number of points falls below a predefined number of points (threshold), the leaf is deleted and its expansion area is assigned to the neighboring leaves as the result of the removal of the plane used to create it. The points of the deleted leaf are reinserted and the parent nodes are updated recursively.

When a child (leaf or node) of a node is deleted, the node is updated as follows. If the number of the children after deletion remains above the lower threshold the child is deleted, the plane used to create its expansion area is removed and its expansion area is assigned to the neighboring children.

If the number of the children after deletion falls below the lower threshold, the node is deleted, the parent nodes are recursively updated, and the points of the corresponding leaves are reinserted.

4.3 Exact Search

To perform exact search for a point, both the expansion and data areas are used. The expansion area including the searched point is determined and it is checked whether the point is included in the corresponding data area. This is performed recursively towards the leaves. If a point is outside the data area, it is safely concluded that it is not in the database. The height of the tree is the upper bound of the number of nodes to be searched in order to determine whether the point is or not in our database.

4.4 Similarity Queries

Range query In order to determine the points, which are located within a distance x from point A, only the information of the data area and the distance of the encapsulated points from the reference point are used. The information pertaining the expansion area is ignored.

The children of a node are ordered according to the minimum distance of the points enclosed in the leaves they point, from the reference point. This permits us to exclude

the children (nodes or leaves) whose minimum distance from the reference point is larger than the sum of the distance of point A from the reference point and x. Children (nodes or leaves) including points whose maximum distance from the reference point is less than the distance of A reduced by x are also excluded. By this procedure the number of distance calculations that will subsequently be required is significantly reduced.

The next step is to exclude children (nodes or leaves), which are located in larger than x distance from point A. For that, the minimum distance of point A from every child is calculated and those whose distance is greater than x are excluded.

A similar procedure is applied to leaves. The points included in the leaf are indexed by their distance from the reference point and those whose distance is less than that of A reduced by x or greater than the distance of A increased by x are excluded. This reduces the number of subsequent calls of the distance function. The distance of the remaining points from A is then calculated and evaluated.

Using the above referenced pruning criteria in combination with the overlap-free feature of Pde-tree, much fewer node and leaf read operations as well as much fewer calls of the distance function are expected to be required in range queries, compared to previous structures.

Search of k Nearest Points The algorithm of k-nearest points query is the one described in [9] adjusted to the Pde-tree. To find the k-nearest to a given point A, an array of nodes/leaves ordered by decreasing distance from point A is constructed. Root is the first node inserted into the array. If the last member of the array (the closest to point A) is a node, it is deleted from the array and every one of its children is inserted in the sorted array. If the last member of the array is a leaf it is deleted from the array and the enclosed points are checked to select the k-nearest points. This procedure is repeated recursively until k points are determined. After determining k points this procedure is repeated until the distance of the k-point of the nearest neighbors is less than the distance of the last member of the array. In this procedure the expansion area is not used.

Using the pre-calculated distance of the points of a leaf from the reference point, the points whose distance is smaller than the distance of point A reduced by the distance of up to that moment k-point as well as the points whose distance is bigger than the sum of these two distances are excluded. This way the number of calls of the distance function is greatly reduced. The no overlapping feature of Pde-tree leads to less read operations than other methods.

Range Queries with Different Distance Function To determine the set of points B satisfying the condition $\varphi(A, B) < d$, where $\varphi(A, B)$ is the distance of points A,B calculated with function $\varphi(x, y)$, which is *different* from the distance function that has been used to construct the tree, we do the following: The maximum distance that a point satisfying the condition could have in every dimension of the multidimensional space is calculated. This way a region containing potential answers to the query is determined and subsequently the maximum of the distances of the points of this region from the query point is calculated using the distance function employed for the creation of the tree. This distance is used for pruning of nodes and leaves or points by comparing

it with the pre-calculated distance from the reference point. The procedure is the same in every other respect with that applied before, except that the distance function $\varphi(x, y)$ is used.

5 The Pde_r-tree

The Pde_r-tree is a member of the Pde-tree family. The above description and algorithms are valid for the Pde_r-tree. Assuming a unit normalized feature space, the Pde_r-tree is implemented as follows:

1. *The data and the expansion area shape:* A simple shape that would make possible the occupation of the entire feature space without region overlapping is the rectangle. This selection gives the letter r to the name of the tree structure.
2. *The leaf split strategy:* The dimension along which the length of the data region is maximum is determined. The split is made by a plane crossing the above referenced axis at such a position that an equal number of points is left in the two resulting data regions. If this is not possible due to the concentration of points on the plane, then the axis with the next lower length is selected recursively until the condition is met. If this is not possible, the dimension along which the most balanced split is achieved is selected.
3. *The distance function:* Any function that meets the necessary properties can be used. In the current implementation of the Pde_r-tree the first order Minkowski Metric was selected.

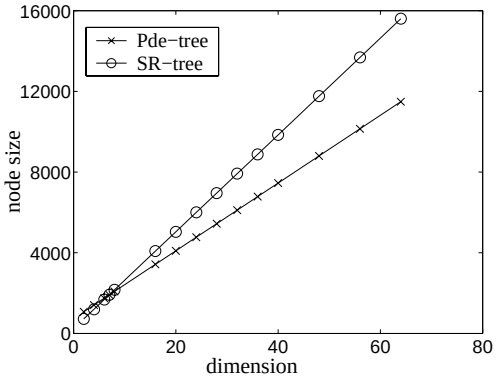

Fig. 1. SR-tree and Pde_r-tree node size requirements.

A leaf of the Pde_r-tree consists of entries $E_1, ..., E_n$ where n is between or equal to the minimum and the maximum number of entries in a leaf. Each entry contains a point, the distance of the point from the reference point and a pointer to additional information of the point.

A node consists of two parts: (i) The first part consists of entries $C_1, ..., C_q$, where q is between or equal to the minimum and the maximum allowable number of entries in a node. Each entry corresponds to a child data region and includes a child pointer, the data region rectangle, the minimum and the maximum distance of the enclosed points from the reference point. The entries are sorted according to the minimum distance of the enclosed points from the reference point. (ii) The second part consists of the expansion area of the current node and an array containing the split information used to divide the expansion area to the expansion areas of the children of the node. As described, an important aspect of tree structures is the amount of information that has to be stored for every node. Figure 1 illustrates that the proposed structure compares favorably to the SR-tree in high-dimensional feature spaces, namely when the dimensionality of the data exceeds 7 for the case of a maximum of 20 children per node. Although structures like the R-tree and the SS-tree require less information to be stored, they lack important features of the Pde_r-tree, namely the disjointness and the ability to use various distance functions during query execution respectively.

6 Performance Evaluation

In this section we present experimental results on the performance of the Pde_r-tree. We focus on studying the factors that affect its performance so as to understand its advantages. A comparative performance evaluation against other structures is currently an on-going research direction that we follow.

Artificial and real data were used through our experiments. Independent distributions in every dimension were used as artificial data. The distributions used for 2-dimensional spaces were the normal, the uniform as well as the combination of normal distributions in each dimension. For spaces of dimension 4 and 8, the normal and uniform distributions were used. Data from image processing in a 7-dimensional feature space corresponding to the features describing the objects of images were used as real data [10]. They were organized according to the subject of the image (car, tiger, etc) and were inserted in the database without changing the existing order.

The tree is implemented with a suitable size of node to point to 20 children and a suitable size of leaf to include 12 points. The capacity of the node and the leaf remains constant throughout all the case studies in order to expose the characteristics of the Pde_r-tree structure without restricting performance evaluation to the case where a specific hardware of operating system is employed.

6.1 A 2-dimensional Case Study

Initially, in order to have an intuitive representation (through the illustration of the outcome in 2-d charts) of the inter workings of the Pde_r-tree and the overall manipulation of data, a 2-dimensional case study is implemented.

Figure 2(a) illustrates a set of points that are organized in two highly concentrated clusters. Figures 2(b) and 2(c) show the resulting leaf and internal nodes of the Pde_r-tree. The disjointness property is evident in these figures. An analogous conclusion is drawn for the data set of Fig. 2(d) (points following uniform distribution), where Figs. 2(e) and 2(f) illustrate the resulting leaves and nodes.

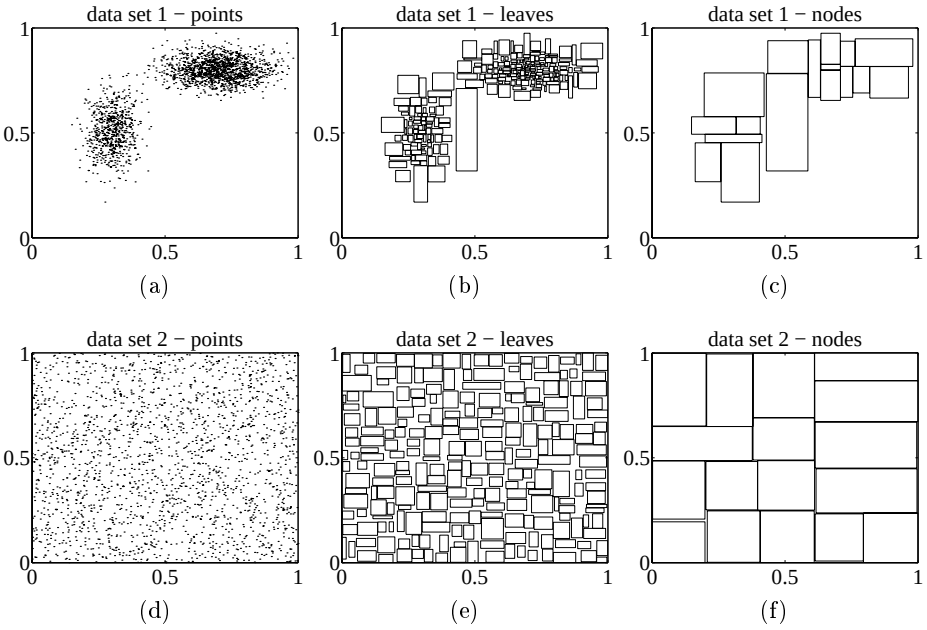

Fig. 2. Two-dimensional case studies. The points in (a) follow a combination of independent in each dimension normal distributions with mean values (0.3, 0.5) for 34% of the points and (0.7, 0.8) for the remaining 66%. The points in (d) follow independent in each dimension uniform distributions in the range [0, 1].

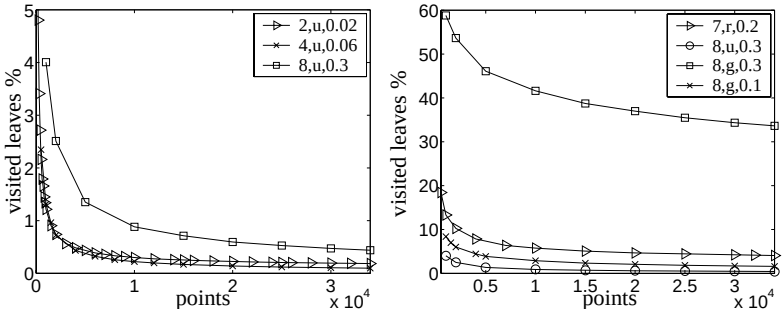

Fig. 3. The mean number of leaves visited in order to answer range queries, as a percentage of the total number of leaves. The three attributes recorded in the legend for every curve are the feature space dimension, the data distribution (u: uniform, g: normal, r: real data), and the distance defining the range of the query.

6.2 Range Queries

During a query, the number of visited leaves depends on the position of the query point, thus performance evaluation results depend heavily upon the employed query-point set. While various such sets can be used, the points for which queries are executed are in practice expected to approximately follow the distribution of the actual data set. To thoroughly evaluate the efficiency of the Pde_r-tree, a query will be performed for every point of the tree and the average of the results (number of visited leaves and number of reads, i.e. leaves and nodes visited) will be considered. Queries resulting to zero point answer are not included in the evaluation procedure. A query with such a result is possible to be answered even with a read of a single node without visiting any leaf. Consequently, our test procedure considers the worst case; queries in real applications are expected to have better results.

Figure 3 presents the mean number of leaves visited in order to answer range queries, as a percentage of the total number of leaves. Figure 4 presents the mean number of reads required in order to answer range queries, as a percentage of the number of reads in the case where the data are stored and read sequentially.

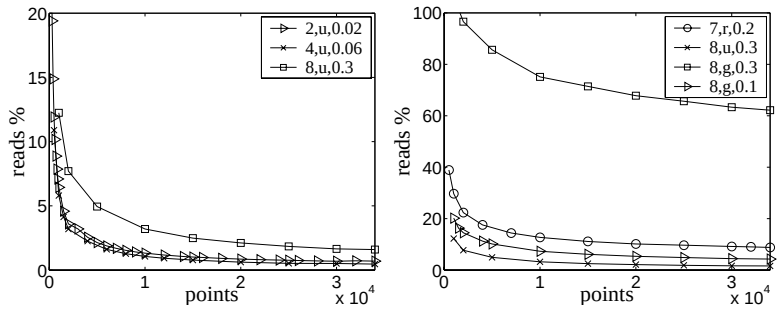

Fig. 4. The mean number of reads required in order to answer range queries, as a percentage of the number of reads in the sequential case. The three attributes recorded in the legend for every curve are the feature space dimension, the data distribution (u: uniform, g: normal, r: real data), and the distance defining the range of the query.

We observe that the curves are decreasing, that is the algorithms of the tree respond better as the data set gets larger. It is also observed that if we try to evaluate the tree with a small data set (small depending on the dimension) the results may not be reliable.

6.3 K-nearest Neighbor Queries

The assumption about the query-point set used in range queries in the previous section applies to k-nearest neighbor queries as well. The case studies include queries with 2 and 21 points. The result in the case of 2-nearest neighbors query is the query-point itself

and its nearest neighbor. The case of 2-nearest neighbors, with the above mentioned query points set, is a worse case compared to a usual query for 1-nearest neighbor for query points that may not be included in the dataset. In the former case, a leaf enclosing the query point exists and thus is visited during query execution, whereas in the latter case it is possible that no leaf that could potentially enclose the query point in its data area exists.

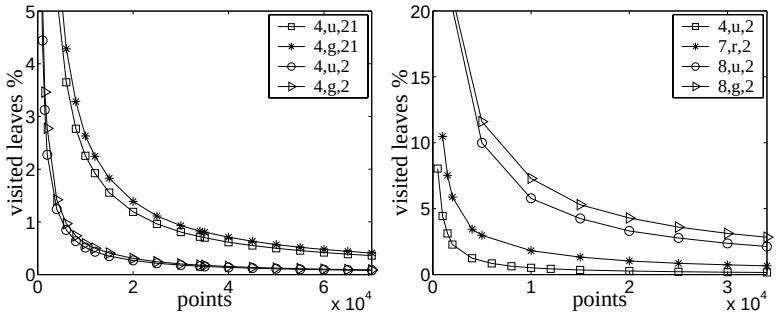

Fig. 5. The mean number of leaves visited in order to answer 2- and 21-nearest neighbor queries, as a percentage of the total number of leaves. The three attributes recorded in the legend for every curve are the feature space dimension, the data distribution (u: uniform, g: normal, r: real data), and the number of neighbors of the query.

Figure 5 presents the mean number of leaves visited in order to answer 2- and 21-nearest neighbor queries, as a percentage of the total number of leaves. Figure 6 presents the mean number of reads required in order to answer nearest neighbor queries, as a

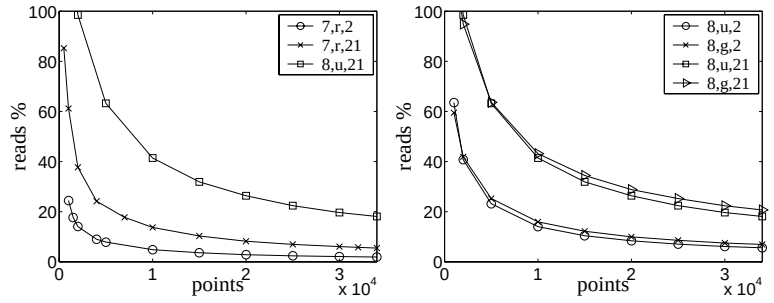

Fig. 6. The mean number of reads required in order to answer nearest neighbor queries, as a percentage of the number of reads in the sequential case. The three attributes recorded in the legend for every curve are the feature space dimension, the data distribution (u: uniform, g: normal, r: real data), and the number of neighbors of the query.

percentage of the number of reads in the case where the data are stored and read sequentially.

Although the Pde-tree is not recommended for inserting sorted data, in one of our case studies we experimented with the sensitivity of the insertion algorithm to the insertion of sorted points. The experiment was as follows: 2000 points of dimension 4 randomly distributed (uniform distribution) were inserted, 2000 points sorted along one axis were subsequently inserted and then points were inserted randomly to the number of 34000 points. As made evident from Fig. 7 (curve '4,up,2') and Fig. 8 (curve '4,up'), no performance degradation is observed.

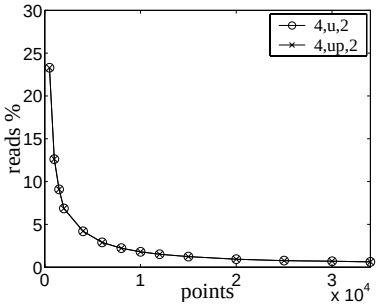

Fig. 7. The mean number of reads required in order to answer nearest neighbor queries, as a percentage of the number of reads in the sequential case, for random and partially sorted insertions.

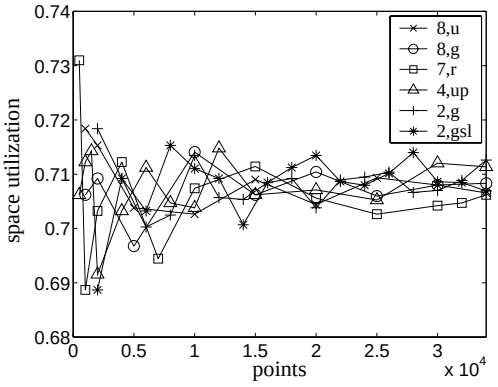

Fig. 8. Average space utilization of leaves.

6.4 Leaves Space Utilization

Figure 8 presents the average space utilization of leaves. For every curve two attributes are described, the feature space dimension (2, 4, 7, or 8) and the data distribution (u: uniform, g: normal, r: real data, up: uniform partially sorted, gsl: normal slided). The space utilization is around 71%.

The curve '2,g' corresponds to the data set of Fig. 2(a). The data set corresponding to the curve '2,gsl' is quite similar to that of '2,g': the first 10 thousands points follow the distribution of the previous data set, while for the creation of the next 10 thousands points the mean values of the normal distributions were moved from (0.3, 0.5) and (0.7, 0.8) (Fig. 2) to (0.32, 0.52) and (0.72, 0.82) respectively. For the remaining data (14 thousands points), the mean values were moved to (0.34, 0.54) and (0.74, 0.84). Although the distribution did not remain the same during the database creation, no effect is observed. This is a clear indication that Pde_r-tree is not sensitive to small modification of the data distribution during the lifetime of the database.

7 Conclusions

In this paper the Pde-tree index family for the efficient indexing of point data is proposed. Pde-trees ensure the disjointness of data regions and the non-indexing of unused space. The implementation of the Pde_r-tree proved that the efficiency of the algorithms increases with the number of points in the tree. It behaves reliably, regardless of the data distribution.

Detailed experimental results illustrate the advantages of the proposed method and its sensitivity against several factors. Although Pde-trees are not designed for efficient insertion of sorted data, the experiments showed that it is not sensitive and behaves very well when a part of the data has been sorted before insertion. Another advantage of Pde-trees is that various distance functions can be used in similarity queries, which may be different from the distance function used to create the database.

As future work we consider the comparative examination of Pde-trees against other structures. This is an ongoing research task that we are currently following, searching for the impact of the trade-offs of different index structures. For instance, Pde-trees trade the fanout (by requiring additional information to be stored within nodes), whereas R+-trees (which also achieve disjointness) trade storage space (due to replication).

References

1. A. Guttman: "R-trees: a Dynamic Index Structure for Spatial Searching", *Proceedings ACM SIGMOD Conference*, Boston, MA, (1984) 47-57
2. T. Sellis, N. Roussopoulos, C. Faloutsos: "The R+-tree: a Dynamic Index for Multi-Dimensional Objects", *Proceedings 13th VLDB Conference*, Brighton, England, (1987) 507-518
3. N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger: "The R*-tree: an Efficient and Robust Access Method for Points and Rectangles", *Proceedings ACM SIGMOD Conference*, Atlantic City, NJ, (1990) 322-331

4. J.T. Robinson: "The K-D-B-tree: a Search Structure for Large Multidimensional Dynamic Indexes", *Proceedings ACM SIGMOD Conference*, Ann Arbor, MI, (1981) 10-18
5. D.A. White, R. Jain: "Similarity Indexing with the SS-tree", *Proceedings 12th IEEE ICDE Conference*, New Orleans, LO, (1996) 516-523
6. N. Katayama, S. Satoh: "The SR-tree: an Index Structure for High-Dimensional Nearest Neighbor Queries", *Proceedings ACM SIGMOD Conference*, Tucson, AZ, (1997) 369-380
7. P.Ciaccia, M.Patella, P. Zezula: "M-tree: an Efficient Access Method for Similarity Search in Metric Spaces", *Proceedings 23rd VLDB Conference*, Athens, Greece, (1997) 426-435
8. C. Aggarwal, A. Hinneburg, D. Keim: "On the Surprising Behavior of Distance Metrics in High Dimensional Space", *Proceedings ICDT Conference*, London, UK, (2001) 420-434
9. G. Hjaltason, H. Samet: "Distance Browsing in Spatial Databases", *ACM Transactions on Database Systems*, 24(2):265-318, (1999)
10. V. Mezaris, I. Kompatsiaris, M.G. Strintzis "Ontologies for Object-based Image Retrieval", *Proceedings Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, London, UK, (2003) 96-101