IOS Press

Center-Based Indexing in Vector and Metric Spaces

Arkadiusz Wojna

Institute of Informatics, Warsaw University

ul. Banacha 2, 02-097 Warsaw, Poland

wojna@mimuw.edu.pl

Abstract. The paper addresses the problem of indexing data for k nearest neighbors search. Given a collection of data objects and a similarity measure the searching goal is to find quickly the k most similar objects to a given query object. We present a top-down indexing method that employs a widely used scheme of indexing algorithms. It starts with the whole set of objects at the root of an indexing tree and iteratively splits data at each level of indexing hierarchy. In the paper two different data models are considered. In the first, data objects are represented by vectors from a multi-dimensional vector space. The second, more general, is based on an assumption that data objects satisfy only the axioms of a metric space. We propose an iterative k-means algorithm for tree node splitting in case of a vector space and an iterative k-approximate-centers algorithm in case when only a metric space is provided. The experiments show that the iterative k-means splitting procedure accelerates significantly the k-nn searching over the one-step procedure used in other indexing structures such as GNAT, SS-tree and M-tree and that the representation of a tree node is an important issue for the performance of the search process. We also combine different search pruning criteria used in BST, GHT nad GNAT structures into one and show that such a combination outperforms significantly each single pruning criterion. The experiments are performed for a large number of benchmark data sets of the size up to several hundreds of thousands of data objects. The indexing tree with the k-means splitting procedure and the complex search criteria is particularly effective for the largest data sets for which this tree accelerates searching up to several thousands times.

1. Introduction

The problem of similarity based searching is an extension of the exact searching widely used in text and database applications. It is assumed that a distance measure is defined on objects and the problem is to find k objects from a set of data objects that are nearest to a given query object. The problem plays an important role for multimedia, machine learning and data mining applications, in particular the method of k nearest neighbors is one of the most popular classification models [13], [15] used by researchers and practitioners. For a long time the method was not used for practical applications because of the

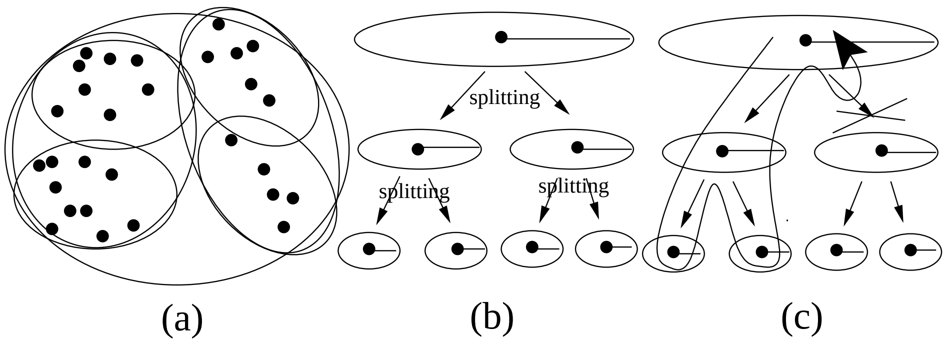


Figure 1. Indexing and searching in a data set: (a) a hierarchical structure of data clusters, (b) an indexing tree with nodes corresponding to data clusters, (c) search pruning in an indexing tree.

large computational complexity of the problem but the technology advance in recent decade allowed to apply the method to numerous domains like spatial databases, text information retrieval, image, audio and video recognition, DNA and protein sequence matching, planning and time series matching (e.g., in stock market prognosis and weather forecasting) [2, 32].

The basic approach to searching for nearest neighbors in a collection of data objects is to compute the distance from a query object to each data object in the collection and select the objects with the smallest distance. If the size of the collection is n and the number of queries is m the computational cost of finding the nearest neighbors to all queries is $O(mn)$. In many applications the size of a database is large (e.g., hundreds of thousands of objects) and then the cost $O(mn)$ is not acceptable. To reduce the cost of searching one can construct an indexing structure. The general idea of indexing is that it splits the whole collection of data objects into clusters in such a way that each cluster contains objects from the same region of a data space (Figure 1a). Each cluster has a compact representation that allows to check quickly whether the cluster can contain the nearest neighbors of a query object (Figure 1b). Instead of comparing a query object directly with each data object first it is compared against the whole regions. If a region is proved not to contain the nearest neighbors it is discarded from searching and in this way a large number of distance computations is saved (Figure 1c).

An important issue for indexing methods is what are the initial assumptions made about a data space. Two models are used. The first one assumes that data objects are represented by vectors from a vector space. This model is applicable to structural databases and complex multimedia objects transformable to feature vectors. However, not all databases fit to this model. In particular, there are spaces for which a distance function is provided (so called metric spaces) not based on feature vectors, e.g., texts with the editing distance, DNA or time dependent sequences or plans. This model is based on an assumption that only a positive distance function is defined for all pairs of data objects and structural properties of data are not used. In the paper, we present two variants of the same tree-based indexing method: for the case of a vector space and for the case of a metric space.

We restrict our consideration to application of the k-nn method mainly for object classification. It requires fast access to data therefore our effort has been concentrated on the case when data are kept in the main memory. With growing size of the main memory in data servers this case attracts more and more attention of people working in research and application areas.

The tree structure considered in the paper is similar to tree structures such as BST [21], GHT [31],

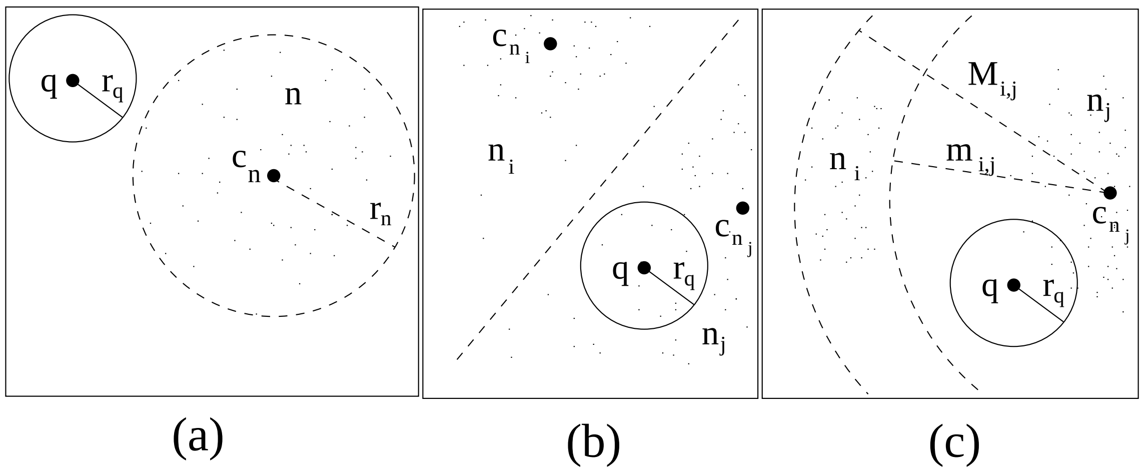


Figure 2. Three search pruning criteria: (a) the covering radius from BST, (b) the hyperplane cut from GHT, (c) the rings based from GNAT.

GNAT [8], SS-tree [37] and M-tree [11]. It is constructed with a top-down method that starts with the whole data set at the root of a tree and recursively at each node splits data objects into a fixed k smaller clusters (Figures 1a,1b). The main features that differentiate our tree structure from the above indexing structures are the node splitting procedure (Figure 1b) and the search pruning criteria (Figure 1c).

All the above mentioned methods from the literature use a one-step clustering procedure to split a node in a tree. Such procedure selects a number of cluster centers among objects at the node and assigns each data objects from the node to the nearest center. We propose a similar procedure but at every splitting operation the clustering procedure is repeated iteratively until the cluster centers are becoming stable. The experiments with real-life data sets described in Section 11 show that our iterative splitting procedure improves the effectiveness of the indexing tree. This empirical fact is supported by the theoretical result of Savaresi and Boley [28] who have proved that for an infinite elliptical model the iterative 2-means clustering procedure converges to a partition across the principal direction of data distribution.

As the search pruning criterion we propose the combination of three criteria from BST [21], GHT [31] and GNAT [8]. All three structures use the same clustering procedure that selects a number of centers and assigns each data object to the nearest center. However, each of the methods uses a different pruning criterion. BST keeps the center c_n and the covering radius r_n for each tree node n and uses the covering radius criterion to prune branches of the tree (see Figure 2a). The branch is discarded when the distance $\rho(c_n, q)$ between the center c_n and the query object q is larger than the sum $r_n + r_q$ of the covering radius r_n and the distance $r_q := \rho(q, x_{nearest})$ between the query object q and the previous nearest neighbor $x_{nearest}$. GHT is constructed identically as BST but it uses the hyperplanes separating subnodes of the same parent as the pruning criterion instead of the covering radius (see Figure 2b). Assume that c_{n_i} and c_{n_j} are the centers of two nodes n_i and n_j. Then the node n_i is discarded if $\rho(c_{n_i}, q) - r_q > \rho(c_{n_j}, q) + r_q$. GNAT pruning criterion is also based on mutual relation between clusters but it is more complex (see Figure 2c). If the degree of a tree node is k then each child node n_i keeps the minimal $m_{i,1}, \ldots, m_{i,k}$ and the maximal $M_{i,1}, \ldots, M_{i,k}$ distances from its elements to the centers of the remaining child nodes. Then the node n_i is discarded if there is a node n_j centered in c_{n_j} such that either $\rho(q, c_{n_j}) + r_q < m_{i,j}$ or

$\rho(q, c_{n_j}) - r_q > M_{i,j}$. Since all the above methods are based on the nearest center clustering procedure the search method presented in the paper combines all three pruning criteria into one. The experiments with real-life data sets described in Section 10 show that such a combination is more effective than any single criterion.

As the final result we propose the indexing tree with the k-means based splitting procedure and the complex search pruning criteria as the new indexing structure. In Section 11 we show that the method presented in the paper is several times faster than the tree with a one-step splitting procedure and a single criterion. Hence, in case of large databases it can accelerate searching even up to several thousands times in comparison to the linear search.

The paper is organized as follows. Section 2 describes a number of works related to the subject. Section 3 outlines the distance function used for similarity measure. Section 4 presents the general model of the k nearest neighbor search. Section 5 describes the data sets used in tests. The general schemes of the indexing and the searching methods are presented in Section 6. Section 7 defines the k-means as the method for splitting tree nodes in vector spaces and generalizes it to the k-approximate-centers for metric spaces. Section 8 discusses the problem of the initial centers selection in the k-means splitting procedure. Section 9 contains experimental results and discussion related to selection of the degree of an indexing tree node. Section 10 presents three criteria for search pruning and provides experimental results and analysis of significance of particular criteria. Section 11 provides experimental results comparing the performance of the iterative splitting procedures to a non-iterative one and to other methods known from the literature. In Section 12 the costs of indexing and searching are compared and discussed. Section 13 concludes the paper with a brief summary and discussion of possible directions for future research.

2. Related Work

The fundamental notion for the k-nn method is a distance measure. As a distance we use a weighted version of the Value Difference Metric (VDM) introduced by Stanfill and Waltz [30]. The VDM metric is an instance of a metric induced from the L_p norm. Originally Stanfill and Waltz used the Euclidean distance. Cost and Salzberg [12] used a simplified version of the VDM metric with the city block distance instead of the Euclidean. Aggarwal et al. [1] examine the meaningfulness of the concept of similarity in high-dimensional spaces with the L_p norm based metric and show that the smaller p the more effective metric is induced from the norm L_p. Our experiments confirmed this theoretical result and in the paper we use the metric with the city block distance definition. The experimental evaluation of the non-weighted version of the VDM metric is contained in [18, 19]. As an attribute weighting method we use an original algorithm described in Section 3. An exhaustive overview of other weighting methods is provided in [36].

In many applications the k nearest neighbors method provides a classification model for a decision system. The foundations of this paradigm are described in [24]. Wettschereck in his PhD thesis [35] presents an exhaustive analysis of different versions of the k-nn method with empirical evaluation of particular algorithms. In our work we use the majority voting as the method for decision selection.

With the increasing interest in the similarity based searching strategies a considerable effort has been made to accelerate searching techniques and a number of indexing methods both for general and for particular applications have been developed. It was recognized that bottom-up constructions such as Ward's clustering [33] lead to a very good performance but the $O(n^2)$ complexity of such constructions has made this approach too expensive for most of applications and the top-down scheme has remained more popular in practice. As a future work Brin suggested to consider a scheme with a top-down construction that is iteratively improved until it converges to a bottom-up type construction [8].

Since a great part of applications is associated with structural databases and multimedia objects transformed to feature vectors a number of indexing techniques have been developed for vector spaces (e.g., quad-trees [16] and k-d trees [4]). The cost of a distance computing operation between two vectors is usually low so the methods such as grid-files [25], k-d-b tree [26], R-tree [20] and its variants R^+-tree [29] and $R^\star$-tree [3] were set on optimizing the number of I/O operations. The above techniques generally work well for low dimensional problems, but the performance degrades rapidly with increasing dimensionality. This phenomenon called the dimensional curse have been theoretically substantiated by Beyer et al. [6]. They proved that under certain reasonable assumptions the ratio of the nearest and the farthest neighbors converges to 1 while increasing dimensionality. In order to avoid the problem some specialized methods for high-dimensional spaces have been proposed: X-trees [5], SR-trees [22], TV-trees [23] and VA-files [34].

All the above tree based methods are based on regions that are hypercubes so the application of these methods is strictly limited to vector spaces. However, a large number of databases with other kinds of distance measures have caused an increase of interest in general distance-based indexing methods and in our work we have focused on this more general case. An exhaustive overview of indexing methods for metric spaces is contained in [10]. SS-tree [37] uses a more general clustering scheme with spheres instead of rectangles as bounding regions but it is still limited because it uses the mean as the center of a cluster. A general distance-based indexing scheme is used in BST [21] and GHT [31]. In both cases trees have the same construction but different pruning criteria are used. GNAT [8], SS-tree [37] and M-tree [11] are specialized versions of the BST/GHT tree. To balance the tree GNAT computes the number of child nodes for each node separately. As the splitting procedure GNAT uses an algorithm that selects from a sample a previously computed number of centers and assigns objects from the parent node to the nearest centers. SS-tree and M-tree are focused on optimizing the number of I/O operations so they maintain a structure of nodes analogical to B-trees and allow a dynamic growth of the database. Clustering in M-tree is similar to clustering algorithm in SS-tree but M-tree uses either a random or a sampled set of centers instead of the means. Hence, it uses only a distance function and is applicable to any metric space. All the above structure use a one-step procedure for splitting tree nodes. In our method we propose an iterative procedure instead of the one-step and combine three search pruning criteria from BST, GHT and GNAT into one.

3. Distance Function

We assume that data objects are given from a pseudometric space $\mathbb{X}$ with a distance function $\rho : \mathbb{X}^2 \to \mathbb{R}$ such that $\forall x, y, z \in \mathbb{X}$:

1. $\rho(x, y) \geq 0$ (positivity)
2. $\rho(x, y) = \rho(y, x)$ (symmetry)
3. $\rho(x, x) = 0$ (reflexivity)
4. $\rho(x, y) + \rho(y, z) \geq \rho(x, z)$ (triangular inequality)

The triangular inequality is important for the performance of a searching process. All search pruning criteria are based on this property. In many papers the definition of a distance function satisfies the strict positivity: $x \neq y \Rightarrow \rho(x,y) > 0$. However, a number of important distance measures as the VDM metric [30] does not satisfy the strict positivity and all the indexing methods mentioned in the previous sections do not require this property to be satisfied.

The performance of the k-nn based classification method depends critically on the distance function used to measure similarity among data objects. All the data sets used in our experiments have data objects represented as vectors of attribute values $x = (x_1, \ldots, x_d)$ so we have assumed a d-dimensional vector space with both numerical and symbolic attributes. As a general scheme of a distance measure between two data objects $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$ we have used the L_1-norm based distance definition:

$$\rho(x,y) = \sum_{i=1}^{d} \rho_i(x_i, y_i)$$

where $\rho_i(\cdot,\cdot)$ is a measure of attribute value similarity. Aggarwal et al. [1] have examined the meaningfulness of the concept of similarity in high-dimensional spaces with the L_p-norm based metric and show that the smaller p the more effective metric is induced from the norm L_p. In the context of this result the L_1-norm based distance is the optimal trade-off between the quality of the measure and the indexing capabilities: the $p = 1$ is the minimal index among the L_p norms that preserves the triangular inequality. The fractional distance measures (with $p < 1$) do not have this property.

In the paper we use a metric specialized for decision systems, i.e., we assume that a training data set $\mathbb{U}$ is provided to induce the metric and each data object $x \in \mathbb{U}$ is labeled with a decision $dec(x)$ from a finite set V_d. The metric combines the distances specialized for numerical and for symbolic attributes [14]. For numerical attributes we used a widely used approach where the difference between attribute values is normalized by its largest observed value difference [27]:

$$\rho_i(x_i, y_i) = w_i \left| \frac{x_i - y_i}{\max_i - min_i} \right| \qquad (1)$$

where $\max_i$ and min_i are the maximal and the minimal value observed for the attribute i in the training set $\mathbb{U}$. The values w_i are the weights computed in the second phase of the metric induction process. For symbolic attributes we have used the VDM metric [12, 30] that considers two symbolic values to be similar if they have similar decision distribution, i.e. if they correlate similarly with the decision:

$$\rho_i(x_i, y_i) = w_i \sum_{v \in V_d} |P(Class(v)|x_i) - P(Class(v)|y_i)| \qquad (2)$$

where $Class(v) = \{x \in \mathbb{U} : dec(x) = v\}$ is the decision class of v. Different variants of this metric have been successfully used previously (see e.g. [30], [12], [7]). An experimental analysis of the quality of the non-weighted version of this metric for a classification task is provided in [18, 19]. The results in [18, 19] show that this metric with the k-nn algorithm gives classification accuracy at least as good as other widely used methods such as C5.0.

As a weighting procedure we have used Algorithm 1. At the beginning it assigns the equal weights $w_i := 1.0$ to all attributes and iteratively improves the weights. At each iteration the algorithm selects a random training and a random test samples $\mathbb{U}_{trn}$ and $\mathbb{U}_{tst}$, classifies each test object x from $\mathbb{U}_{tst}$ with

Algorithm 1 Weighting procedure

```
for each attribute $w_i := 1.0$
$modifier := 0.9$
$convergence := 0.9$
repeat $l$ times
   $\mathbb{U}_{trn}$ := a random training sample from $\mathbb{U}$
   $\mathbb{U}_{tst}$ := a random test sample from $\mathbb{U}$
   $QR = \frac{\sum_{x \in \mathbb{U}_{tst}:\, dec(x)=dec(nearest_{trn}(x))} \rho(x, nearest_{trn}(x))}{\sum_{x \in \mathbb{U}_{tst}} \rho(x, nearest_{trn}(x))}$
   for each attribute $i$
      $QR(i) = \frac{\sum_{x \in \mathbb{U}_{tst}:\, dec(x)=dec(nearest_{trn}(x))} \rho_i(x_i, nearest_{trn}(x)_i)}{\sum_{x \in \mathbb{U}_{tst}} \rho_i(x_i, nearest_{trn}(x)_i)}$
      if $QR(i) > QR$ then $w_i := w_i + modifier$
   $modifier := modifier * convergence$
```

its nearest neighbor in $\mathbb{U}_{trn}$ and computes the global quality ratio QR and the quality ratio $QR(i)$ for each attribute i. The quality ratio is the ratio between the sums of the distances to the nearest neighbors $\rho(x, nearest_{trn}(x))$ for the correctly classified objects and for all training objects. All attributes i that have the quality ratio $QR(i)$ higher than the global quality ratio QR have the weights w_i increased. In order to make the procedure convergable the coefficient $modifier$ used to modify weights is decreased at each iteration of the algorithm.

4. K Nearest Neighbors Method

The k nearest neighbors method is a widely used classification model that for a given test object x assigns a decision that is inferred from the decisions of the set $S(x,k)$ of the k nearest training data objects from $\mathbb{U}$ according to similarity measure ρ. In the paper we used one of the most popular procedures to determine a decision for a test object x on the ground of the neighborhood $S(x,k)$. For each decision value v the $Strength$ measure counts the number of training examples from $S(x,k)$ with the decision v:

$$Strength(x,v) = |\{y \in S(x,k) \,:\, dec(y) = v\}|$$

As a decision for a test object x the algorithm assigns the most frequent decision in the set of the k nearest neighbors $S(x,k)$:

$$decision_{knn}(x) = \arg\max_{v \in V_d} Strength(x,v)$$

In [18, 19] we have reported that classification accuracy can significantly depend on the value of k used to test objects and a different k is appropriate for different problem domains. Therefore, in terms of accuracy of the algorithm, it is important to find the optimal neighborhood size k. To estimate the optimal value of k one may use the leave-one-out method applied to the training set $\mathbb{U}$. The method estimates the classification accuracy of the classifier for different values of k ($1 \leq k \leq k_{max}$) and the value k for which the estimation is the greatest is selected. Applying it directly would require repeating leave-one-out estimation k_{max} times. However, this process may be emulated in time comparable to the

Table 1. Classification error of k-nn with the best k selected from the range $1 \leq k \leq 100$.

Dataset	Trn size	Test size	1-nn	Best k	Best k-nn
segment	1 540	770	2,34%	1	2,34%
splice (DNA)	2 000	1 186	6,83%	18	5,65%
chess	2 131	1 065	1,98%	1	1,98%
satimage	4 435	2 000	10,85%	10	9,50%
mushroom	5 416	2 708	0%	1	0%
pendigits	7 494	3 498	2,69%	4	2,41%
nursery	8 640	4 320	1,14%	1	1,14%
letter	15 000	5 000	3,06%	1	3,06%
census94	30 162	15 060	20,50%	47	15,74%
shuttle	43 500	14 500	0,05%	1	0,05%
census94-95	199 523	99 762	6,79%	17	4,92%
covertype	387 308	193 704	3,64%	3	3,57%

single leave-one-out test for k equal to the maximal possible value $k = k_{max}$. The detailed description of the procedure is provided in [18, 19].

5. Data Sets

We have performed experiments with different indexing and search methods for 12 benchmark data sets from the UCI repository [9]. The size of data sets ranges from a few thousands to several hundreds of thousands. The data sets provided as a single file (*segment, chess, mushroom, nursery, covertype*) have been randomly split into a training and a test part with the ratio 2 to 1. The remaining data sets (*splice, satimage, pendigits, letter, census94, shuttle, census94*-95) have been tested with the originally provided partition. All experiments were performed on an AMD Athlon XP 2100+, with 1024Mb of RAM. Each of the next sections presents results for the same partition of data.

Table 1 presents an exemplary classification accuracy obtained with the k-nn method from a range of performed tests. Classification accuracy does not depend on the selection of an indexing and a searching methods to be used but there are other factors. For all data sets originally split into a training and a test parts the only factor that causes different accuracy at different experiments is the weighting procedure from Section 3 where the final weight values depend on a random selection of a training and a test samples at each weighting iteration. For data sets to be randomly split the accuracy depends also on a partition used. To make the results comparable for each issue discussed in the next sections we have performed the experiments with the same distance weights and the same partition for each data set. To provide some information about significance of the accuracy results we have compared classification error from 5 experiments and noticed that for 1-nn the differences in the classification error were up to 1% and for the best k-nn method the differences were up to 0,5%. The goal of our work is not to analyze

Algorithm 2 Indexing schema

```
$deg$ - the splitting degree of tree nodes
$root$ - the top node with all training data objects from $\mathbb{U}$
$priorityQueue$ - the priority queue of leaf nodes used
                  for selection of the next node to be split

$priorityQueue := \{root\}$
repeat
   $parent :=$ the next node from $priorityQueue$ to be split
   $splitCluster(parent, deg)$
   add child nodes of $parent$ to $priorityQueue$
until the ratio between
   the number of nodes in $priorityQueue$ and $|\mathbb{U}|$ is $\geq \frac{1}{5}$
```

the quality of the used metric. If one is interested in this issue a statistically significant analysis of the used distance function is provided in [18, 19].

For each data set the classification error was measured for different values of k and in Table 1 the highest accuracy among all tested k is given. It is impossible to know the best k before tests but one can use the estimation procedure described in Section 4 in order to provide a good estimation of the best k [18, 19].

6. Indexing and Searching

Most of the distance based indexing methods reported in the literature [16, 4, 26, 20, 29, 3, 5, 22, 23, 21, 31, 8, 37, 11] and all the methods presented in the paper are based on a tree data structure. Algorithm 2 presents a general indexing scheme introduced by Fukunaga and Narendra [17]. All indexing algorithms presented in the paper fit to this scheme. It starts with the whole training data set $\mathbb{U}$ and recursively splits data objects into a fixed number of smaller clusters. The main features that distinguish different indexing trees are the splitting degree of tree nodes deg, the splitting procedure $splitCluster$ and the pruning criteria used in a search process.

Algorithm 2 assumes that the splitting degree is the same for all nodes in the tree. An exception to this assumption is Brin's GNAT structure that balances the tree by selecting the degree of a node proportional to the number of data objects it contains [8]. However, on the ground of experiments Brin has concluded that good balance was not crucial for the performance of the structure. In Section 9 we present the results that have confirmed this conclusion.

We have also assumed that the algorithm stops when the number of leaf nodes exceeds $\frac{1}{5}$ of the size of the training set $|\mathbb{U}|$, in other words when the average size of the leaf nodes is 5. It reflects the trade-off between the optimality of a search process and the memory requirements. In order to make a search process effective the splitting procedure $splitCluster$ has a natural property that data objects that are close each to other are assigned to the same child node. It causes that small nodes at the bottom layer of a tree have usually very near data objects and splitting such nodes until singletons and applying search

Algorithm 3 Searching schema

```
$q$ - query data object
$root$ - the root node of an indexing tree to be searched
$nodeStack$ - the stack of nodes to be searched
$nearestQueue$ - the queue of the data objects nearest to $q$
                 sorted according to the distance $\rho$
$discard(n : node, q : query, r_q : range)$ - the procedure
               checks whether pruning criteria apply to a node $n$
               while searching neighbors of $q$ in the range $r_q$

$nodeStack := \{root\}$
repeat
   $n :=$ pull the top node from $nodeStack$
   $r_q := \max_{x \in nearestQueue} \rho(q, x)$
   if $nearestQueue$ is not full or not $discard(n, q, r_q)$
      if $n$ is a leaf
         for each data object $x \in n$
            check $x$ against the farthest
            object $y \in nearestQueue$
            and replace $y$ with $x$ if $\rho(q, x) < \rho(q, y)$
      else
         sort child nodes of $n$ according to the relevance
         to the query $q$ and push them to $nodeStack$
         in the increasing order according to the relevance
until $nodeStack$ is empty
return $nearestQueue$
```

pruning criteria to such small nodes do not save many distance comparisons. On the other hand, in our implementation the memory usage for the node representation is 2-3 times larger than for the data object representation so the model with the number of leaf nodes equal to $\frac{1}{5}$ of the number of data objects does not increase memory requirements as significantly as the model where nodes are split until the leafs are singletons and the number of all tree nodes is almost twice as the size of the training data set $\mathbb{U}$.

Algorithm 3 is a searching schema [17] assumed to find a fixed number k of data objects nearest to the query q. It traverses an indexing tree rooted at $root$ in the depth-first order and applies a heuristic procedure to determine the order of visiting child nodes. An important issue is that the heuristic procedure guides the algorithm first to child nodes that are more probable to have data object close to the query q. In $nearestQueue$ it stores the nearest data objects, maximally k, from already visited nodes. At each tree node n the algorithm checks with pruning criteria whether n is to be visited, i.e., whether n is possible to contain an object that is closer to the query q than any previously found nearest neighbor from $nearestQueue$. If so and the node n is a leaf, it compares each data object $x \in n$ against data objects in $nearestQueue$ and replaces the farthest object y from $nearestQueue$ if x is closer to the query q than y. In case when the node n is an inner node it adds the child nodes to $nodeStack$ as to be

Algorithm 4 The iterative k-centers splitting procedure $splitCluster(objects, k)$

```
$objects$ - a collection of data objects to be split
          into $k$ clusters
$Cl_1, \ldots, Cl_k$ - partition of data objects from $objects$
          into a set of clusters
$centers$ - the centers of clusters
$prevCenters$ - the centers of clusters
          from the last but one iteration
$getCenter(Cl_i)$ - the procedure computes
          the center of the $i$-th cluster

repeat
   $centers$ :=select $k$ initial seeds from $objects$
   for each $x \in objects$
      assign $x$ to the cluster $Cl_i$
      with the nearest center $c_i \in centers$
   $prevCenters := centers$
   $centers := \emptyset$
   for each cluster $i$
      $c_i := getCenter(Cl_i)$
      add $c_i$ to $centers$
until $prevCenters = centers$
```

visited in the future.

7. K-means and K-approximate-centers

Algorithm 4 presents the iterative splitting procedure $splitCluster(objects, k)$ that is a generalization of the k-means algorithm. Initially it selects k objects as the centers $c_1, \ldots, c_k$ of clusters. Then it assigns each object x to the cluster with the nearest center and computes the new centers $c_1, \ldots, c_k$. It iterates the assignment procedure until the same set of centers is obtained in two subsequent iterations.

The procedure $getCenter(\cdot)$ computes the center of a cluster of objects and we propose different solutions depending on the space type. In the case of a vector space we propose the means as the centers of clusters. In this case Algorithm 4 becomes the well known k-means procedure. Boley and Savaresi have proved the following property of the k-means algorithm:

Theorem 1. [28] If a data set is an infinite set of data points uniformly distributed in a 2-dimensional ellipsoid with the semi-axes of the length 1 and a ($0 < a < 1$) the 2-means iterative procedure with random selection of initial centers has 2 convergence points: one is locally stable and one is locally unstable. The splitting hyperplanes corresponding to the convergence points pass through the center of the ellipsoid and are orthogonal to the main axes of the ellipsoid. The splitting hyperplane corresponding to the stable convergence point is orthogonal to the largest axis of the ellipsoid (see Figure 3).

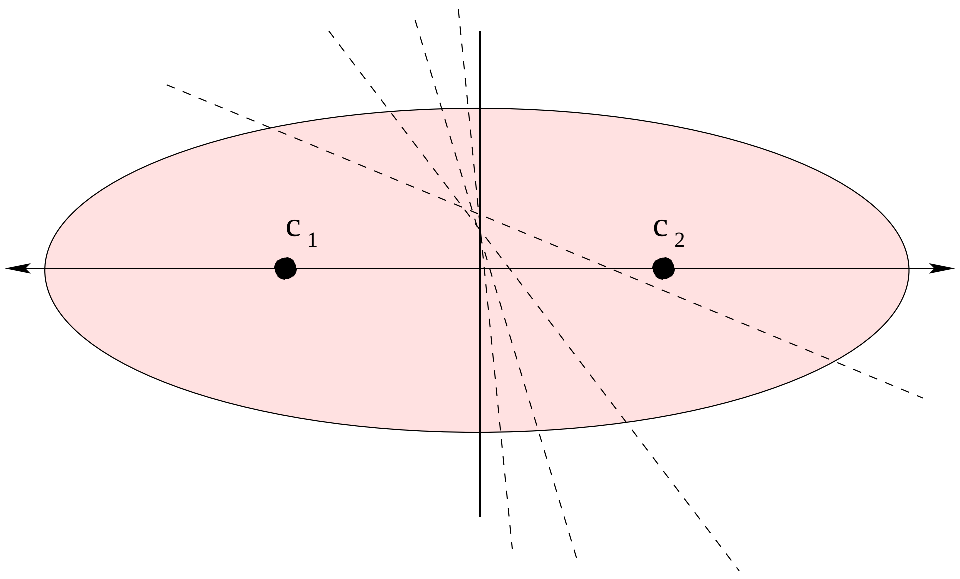


Figure 3. The convergence of the 2-means procedure to the locally stable partition for data uniformly distributed in an ellipse. The splitting line is orthogonal to the largest axis of the ellipse.

This theorem shows that in an infinite theoretical model the 2-means procedure with random selection of initial centers converges in a sense to the optimal partition of data what may indicate good splitting properties of this procedure in practice and explain good experimental performance of a tree based on the 2-means splitting procedure as presented in Section 11.

In general case of a metric space we propose the following approximation of the mean as the center of a cluster. When a cluster Cl_i contains one or two data objects it selects any of them as the center of Cl_i. Otherwise the algorithm constructs a sample S_i that contains the center c_i from the previous iteration used to assign objects and randomly selected $\max(3, \left\lfloor\sqrt{|Cl_i|}\right\rfloor)$ other objects from Cl_i. Then it computes the distances among all pairs of objects from S_i and as the new center of Cl_i it selects the object $c_i \in S_i$ that minimizes the second moment of the distance ρ in S_i:

$$c_i := \arg\min_{x \in S_i} E\left(\rho(x,y)^2\right)$$

In this way it selects the center from S_i that minimizes the variance of S_i. The assumption that the center from the previous iteration is included into the sample S_i in the next iteration makes it possible to use the previous center in the next center selection. It provides a chance for the stopping condition to be satisfied at each iteration and saves a significant number of unnecessary iterations. The computational cost of the center selection remains linear and thus it is comparable to the case of the k-means procedure used for vector spaces.

The discussion and experimental analysis related to selection of initial centers and the degree of nodes and experimental comparison of the indexing trees based on the k-means and the k-approximate centers algorithms with other tree structures are presented in the next sections.

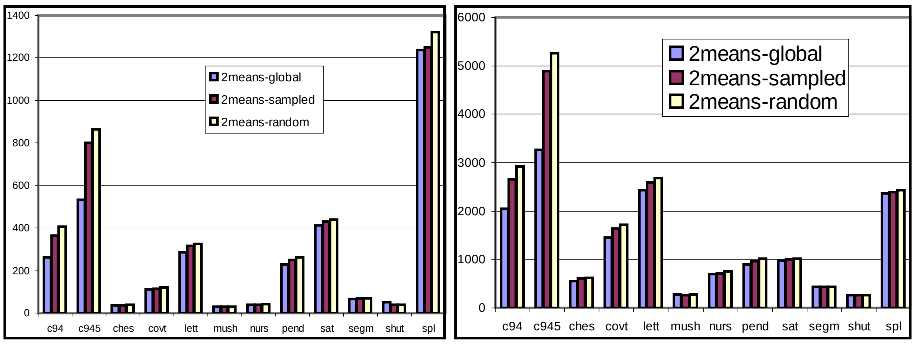


Figure 4. The average number of distance computations per single object of 1-nn (the left graph) and 100-nn (the right graph) search algorithms with the use of the 2-means based indexing trees with three different methods of selecting initial centers: the globally farthest, the sampled farthest and the random.

8. Initial Seeding

One can consider three general approaches in selecting initial centers for clusters in Algorithm 4: random, sampled [8] and exhaustive. The description of BST and GHT is quite general and it either does not specify a particular selection of initial centers or assumes a simple random model [21, 31]. M- [11] and SS-trees [37] are the dynamic structures and the splitting procedures assume that they operate on an existing inner node of a tree and they have access only to the information contained in a node to be split, in particular it does not have access to all data objects in case of splitting a non-leaf node so the splitting procedures from M- and SS-trees are incomparable to the presented iterative procedures.

In GNAT [8] a random sample of the size $3k$ is drown from a set of data objects to be clustered and the initial k centers are picked from this sample. First the algorithm picks one of the sample data objects at random. Then it picks the sample point that is farthest away from this one. Then it picks the sample point that is farthest from these two, i.e., that has the minimum distance from the two the greatest one. Then it picks the one farthest from these three and so on until there are k data points picked.

In the paper we propose yet another method for selecting initial k centers. It is similar to GNAT's method but it selects the first center more carefully and for selection of the others it uses the whole set to be clustered instead of a sample. First the algorithm computes the center of the whole set to be clustered (as the mean in case of a vector space and as the approximate center in case of a metric space). As the first seed it picks the object that is the farthest from the center of the whole data set. Then it repeats selection of the farthest objects as in GNAT but from among the whole set not a sample. The computational cost is $O(nk^2)$ where n is the size of the set to be clustered and for small values of k this cost is still acceptable.

One can consider the exhaustive procedure that checks all k-sets among objects to be clustered as the sets of k centers and selects the best one according to a predefined quality measure but the computational cost of this method does not allow us to use it in practice.

Figure 4 presents the performance of the search algorithm for three different seeding procedures used in the 2-means based indexing trees: a simple random procedure, GNAT's sampled selection of the farthest objects and the global selection of the farthest objects described above. The graphs indicate that the indexing trees with all three methods have the comparable performance what may be explained with the good convergence property of the 2-means algorithm. However, for a few larger data sets: *census94,*

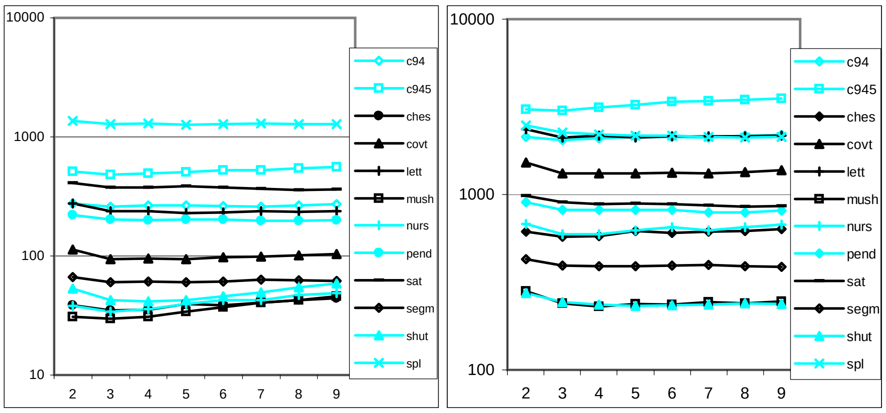


Figure 5. The average number of distance computations per single object of 1-nn (the left graph) and 100-nn (the right graph) search algorithms with the use of the k-means based indexing trees for $2 \leq k \leq 9$.

census94-95, letter and *pendigits* the difference between the global and the two other selection methods is noticeable (50% in comparison to the sampled and 60% in comparison to the random methods in case of the data set *census94-95*).

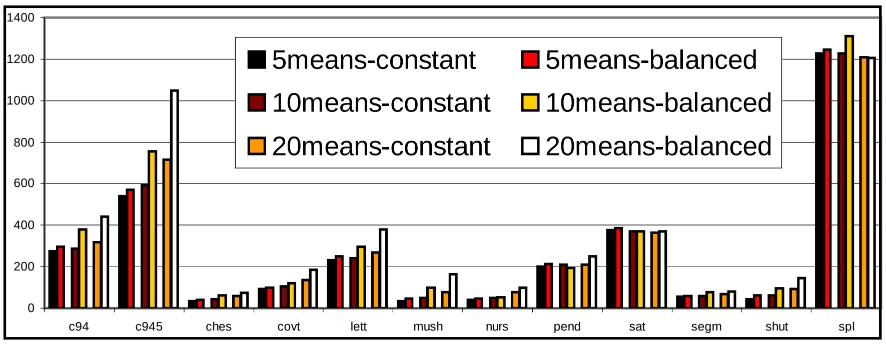
In the experiment the city block metric described in Section 3 was used. For such a metric the k-means algorithm does not have the convergence property: there is a configuration of data points in a vector space that the k-means algorithm "flickers" for, i.e., it never reaches the point when the clusters are the same at two subsequent iterations. Therefore the number of iterations at the splitting procedure has been limited to 1000. We have checked whether this threshold is reached while constructing the indexing trees. It has happened for the sampled and the random methods (for the case of the data sets *census94*, *census94-95* and *covertype*) but never for the global method.

Summing up, the global method seems to have a little advantage over the others and we decided to use this one in farther experiments described in the next sections.

9. Degree of Tree Nodes

In order to analyze the performance of the k-means indexing trees (as a function of the degree of nodes k) we have performed experiments for 8 successive values of k ranging from 2 to 9. Figure 5 presents the performance graphs for particular data sets. As it is shown they are quite stable in the range of tested values except for the value 2 and different values of k have the best performance for particular data sets. At 1-nn search 7 data sets have the best performance at $k = 3$, 1 at $k = 4$ and 2 at $k = 5$ and $k = 8$. At 100-nn search 4 data sets have the best performance at $k = 3$, 2 at $k = 4, 5$ and 8 and 1 at $k = 7$ and 9. These statistics indicate that the best performance is for small values of k (but greater than 2). Assuming k equal to 3, 4 or 5 one may have the confidence that they get almost optimal performance.

In the literature the splitting degree of tree nodes is usually assumed to be constant over all nodes in a tree. The exception to this rule is the GNAT structure [8] that tries to balance the size of branches by choosing different splitting degrees for nodes. It assumes a fixed k to be the average splitting degree of nodes and applies the following procedure to construct a tree. The top node is allocated the degree k.



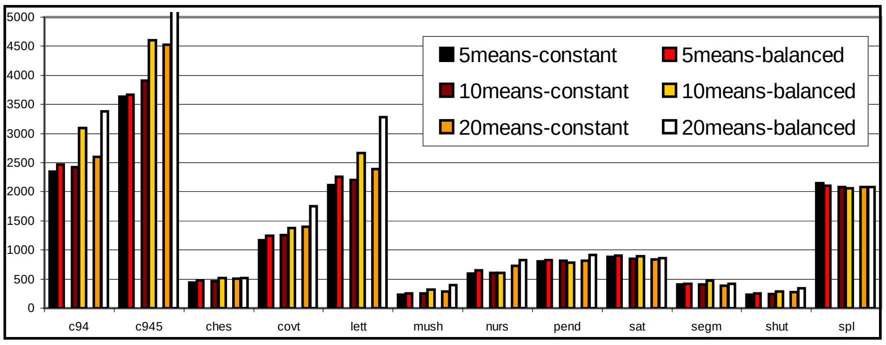


Figure 6. The average number of distance computations per single object of 1-nn (the upper graph) and 100-nn (the lower graph) search algorithms with the use of the k-means based indexing trees with the constant and the balanced degree of tree nodes. For each data set the first pair of columns represents the performance for the constant and the balanced degrees with $k = 5$, the second pair represents the performance for $k = 10$ and the third one for $k = 20$.

Then each of its children is allocated a degree proportional to the number of data points it contains (with a certain minimum and maximum) so that the average is equal to the global degree k. This process works recursively so that the children of each node have the average degree equal to k. In experiments Brin set the minimum of the degree to 2 and the maximum to $\min(5k, 200)$. On the ground of experiments he has reported that good balance was not crucial to the performance of the structure.

We have implemented this balancing procedure too. In case of $k = 2$ the value 2 is both the average and the minimal possible value of the splitting degree in the k-means balanced indexing tree so the balancing procedure assigns the degree 2 to all nodes and thus it behaves identically as in the case of the constant degree 2. Then the comparison of the balanced and the constant degree selections makes sense for the value of k greater than 2. Figure 6 presents the comparison between the k-means based balanced trees where k is the average degree of child nodes and the corresponding k-means trees with the constant degree k. The results shows that the balancing procedure does not improve performance of a tree with a constant degree and in many experiments searching in a tree of a constant degree is even faster. It indicates that in order to make profit from balancing more sophisticated procedures are required and up to now it is not known whether there is a balancing policy within acceptable computational cost that provides a significant advantage over non-balanced structures.

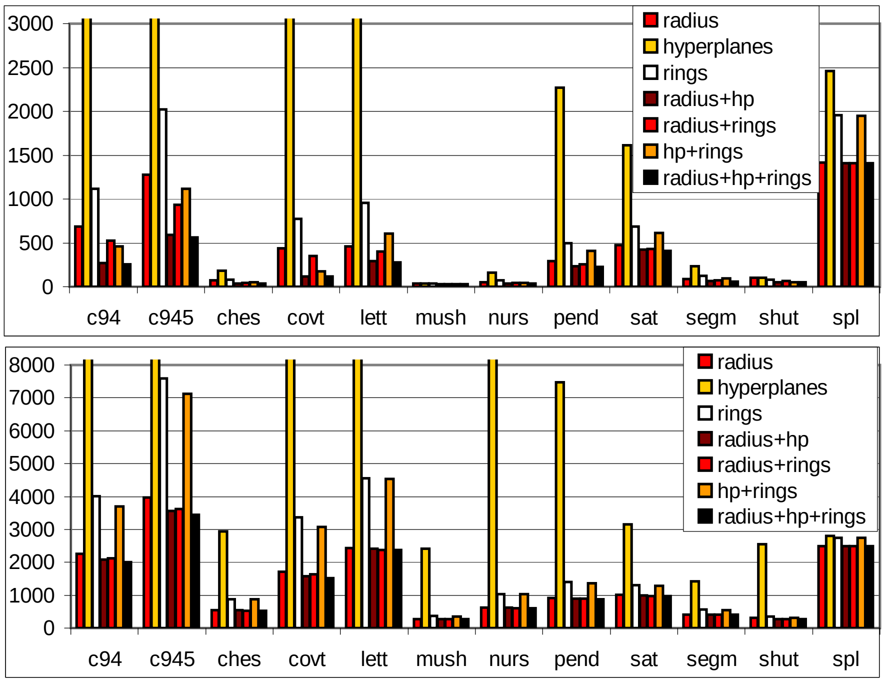


Figure 7. The average number of distance computations per single object of 1-nn (the upper graph) and 100-nn (the lower graph) search algorithms applied to the 2-means indexing tree with the use of different combinations of three search pruning criteria: the covering radius, the hyperplanes and the rings.

10. Search Pruning Criteria

Algorithm 3 presents a searching procedure that uses search pruning criteria to discard nodes while traversing an indexing tree. The procedure $discard(n, q, r_q)$ checks whether a tree node n can contain an object that is closer to the query q than any previously found near neighbor from $nearestQueue$. The value r_q is the distance $\rho(q, x)$ between the query q and the farthest from q object $x \in nearestQueue$. All search pruning criteria described in the literature [8, 11, 21, 31, 37] are based on the triangular inequality.

The most common criterion applied in BST [21], SS-tree [37] and M-tree [11] uses the covering radius (Figure 2a). Each node n keeps the center c_n computed with the function $getCenter(n)$ and the covering radius r_n:

$$r_n := \max_{x \in n} \rho(c_n, x).$$

A node n is discarded if the intersection between the ball around q containing all nearest neighbors from $nearestQueue$ and the ball containing all members of the node n is empty:

$$\rho(c_n, q) > r_q + r_n$$

Uhlmann has proposed another criterion for his Generalized-Hyperplane Tree (GHT) [31] based on the assumption that the splitting procedure assigns each object to the node with the nearest center. It uses the hyperplanes separating the subnodes of the same parent (Figure 2b). A node n_i is discarded if there is a brother node n_j of n_i (another child node of the same parent node as n_i) such that the whole query ball is placed beyond the hyperplane separating n_i and n_j on the side of the brother node:

$$r_{n_i} - r_q > r_{n_j} + r_q.$$

GNAT pruning criteria [8] is also based on mutual relation among brother nodes but is more complex (Figure 2c). If the degree of a tree node is k then each child node n_i keeps the minimal $m_{i,1}, \ldots, m_{i,k}$ and the maximal $M_{i,1}, \ldots, M_{i,k}$ distances from its elements to the centers of the remaining brother nodes:

$$\begin{aligned} m_{i,j} &= \min_{x \in n_i} \rho(c_{n_j}, x) \\ M_{i,j} &= \max_{x \in n_i} \rho(c_{n_j}, x) \end{aligned}$$

A node n_i is discarded if there is a brother node n_j such that the query ball is entirely placed outside the ring around the center of n_j containing all members of n_i:

$$\text{either } \rho(c_{n_j}, q) + r_q < m_{i,j} \text{ or } \rho(c_{n_j}, q) - r_q > M_{i,j}.$$

The covering radius and the hyperplane criteria require from each node n only to store the center c_n and the covering radius r_n. The criterion based on rings requires more memory: each node stores the $2(k-1)$ distances to the centers of brother nodes.

Figure 7 presents the experimental comparison of the performance for all possible combinations of the three criteria. In a single form the most effective criterion is the covering radius, the least effective is the hyperplane criterion and the differences in performance among all three criteria are significant. In the case of the 100-nn search the covering radius alone is almost as powerful as all three criteria and adding the remaining two does not increase the performance. The other situation is in case of the 1-nn search: none of them is comparable to the case when all three criteria are applied. Although the hyperplane criterion alone is least effective it saves more than rings when added to the covering radius, in particular these two work equally well as all three criteria.

The presented results indicate that the combination of different criteria improves the performance of the k-nn search over a single criterion at least for a small values of k. On the other hand, in both cases of the 1-nn and the 100-nn search adding the memory consuming criterion based on rings does not improve the combination of the remaining two. This result may suggest that the covering radius and the hyperplanes provide the optimal pruning combination and there is no need to search for more sophisticated pruning mechanisms.

11. Comparative Study

The most interesting question is how much the search process profits from the additional cost due to the iterative splitting procedure and the combined search pruning criterion in comparison to the one-step case with a single pruning criterion. The iterative procedures select initial centers, assign the data objects to the nearest centers and compute new centers of clusters. Then assignment of data objects to centers and computation of new cluster centers is iterated as long as the same set of cluster centers is generated in two subsequent iterations. The one-step procedure works as in the other indexing trees: BST, GHT, GNAT, SS-tree and M-tree: it stops after the first iteration and uses the initial centers as the final. As the

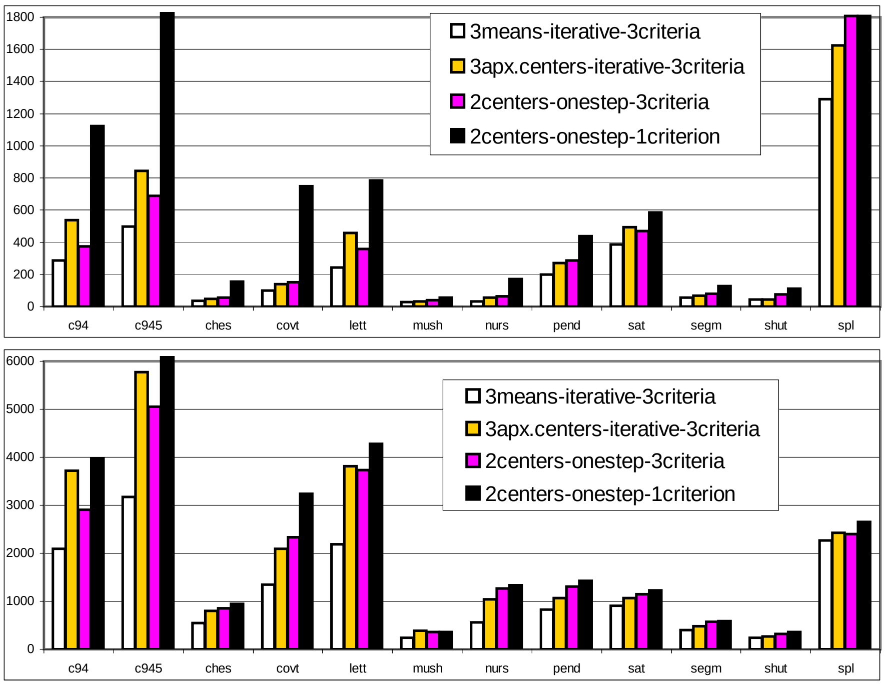


Figure 8. The average number of distance computations per single object of the 1-nn (the upper graph) and the 100-nn (the lower graph) search algorithms in the indexing trees with the iterative 3-means, the iterative 3-approximate-centers and two variants of the one-step 2-centers based splitting procedures: one with the combination of 3 search pruning criteria and one with the single covering radius criterion.

set of initial centers the globally farthest data objects are used both for the iterative and the non-iterative splitting procedures.

Figure 8 presents the cost of searching in the trees with the iterative k-means, the iterative k-approximate-centers and two variants of the one-step k-centers splitting procedures. The results for both trees with the iterative procedures and the first tree with the one-step procedure are obtained with the combination of all three search pruning criteria. The fourth column at each data set presents the performance of the one-step tree with the single criterion based on the covering radius. We chose this criterion for comparison since it has the best performance among all three tested criteria (see Section 10). Apart from a single case we have observed that the performance of the one-step based trees deteriorates while increasing k (it has been checked for $k = 2, 3, 5$ and 7). Then for comparison we have selected the most competitive value $k = 2$ (the exception was the 100-nn search in the data set *splice*, the case $k = 5$ has provided the best performance and then this case has been presented at the graph instead of $k = 2$). In case of both iterative procedures the value $k = 3$ was used since it is one of the most optimal values (see Section 9).

While comparing the performance of the iterative 3-means (the first column) and the one-step 2-centers (the third column) procedures the profit from applying the iterative procedure is visible. In case of the 1-nn search the saving ranges from 20% (*satimage*) to 50% (*nursery*), in case of the 100-nn search the saving is similar to the 1-nn case except for a single data set *splice* where the saving is 5%. These results indicate that replacing the one-step procedure with the iterative 3-means procedure can improve the performance even twice.

The comparison between the third and the fourth column presents the profit for the tree with the one-step procedure only from applying the combined search pruning criteria. For the 1-nn search the combined criterion outperforms a single one significantly, in particular for the largest data sets (*consus94, census94-95, covertype*) the acceleration reaches up to several times. For the 100-nn search the difference is not so large but still it is noticeable. These results show that for the tree with the one-step splitting procedure the complex criterion is crucial for the performance of the tree. In case of the tree with the k-means splitting procedure the results are different, i.e., the difference in performance between the single covering radius and the combined criteria is much smaller (see Section 10). It indicates that the iterative k-means procedure has very good splitting properties and the choice of the search pruning criterion for this case is not as crucial as for the non-iterative case.

The comparison between the first and the fourth columns shows that the tree with the 3-means splitting procedure and the complex search pruning criteria is always at least several tens of percent more effective than the tree with the one-step procedure and a single criterion. In case of the 1-nn search the former tree is usually even several times as effective as the latter one. A particularly advanced acceleration level has been reached for the largest data sets. The size of the data set *covertype* is almost 400 thousands whereas the average number of distance comparisons per single object (the fourth data set from the right) is less than 100 for the 1-nn search and close to 1300 for the 100-nn search. It means that the 3-means based tree reduces the cost of searching 4000 times in case of the 1-nn search and 300 times in case of the 100-nn search. For the second largest data set *census94-95* (the second data set from the right, the size almost 200 thousands) the reductions in cost are 400 and 60 times, respectively. Such good performance has been reached both due to the improved splitting procedure and due to the complex search criterion described in Section 10.

The different situation is while comparing the iterative k-approximate-centers (the second column) and the one-step (the third column) procedures. Although for most of data sets the iterative procedure outperforms the non-iterative one, the differences in performance are usually insignificant and for three large data sets (*census94, census94-95, letter*) the performance of the iterative procedure is even worse than the performance of the non-iterative one. These results indicate that in the case of the tree with the k-approximate-centers the profit in performance is mainly due to the complex search criteria. Since the only feature that differentiates the k-means and the k-approximate-centers procedures is how the algorithm selects and represents the center of a cluster of data objects this feature seems to be important issue for the performance of the indexing trees.

Uhlmann has introduced another type of an indexing structure: the vantage point tree [31]. It is a binary tree constructed in this way that at each node data objects are split with the use of a spherical cut. Given a node n the splitting algorithm picks an object $p \in n$ that is called the vantage point and computes the median radius M such that half of the data objects from n fall inside the ball centered at the vantage point p with the median radius M and half fall outside this ball. The objects inside the ball $\{x \in n : \rho(p,x) \leq M\}$ are inserted into the left branch of the node n and the objects outside the ball $\{x \in n : \rho(p,x) > M\}$ are inserted into the right branch. The vantage point tree is balanced and the construction takes $O(n \log n)$ time in the worst case. While searching for the nearest neighbors of a query q the branch with objects inside the ball is pruned if $M + \rho(q, x_{nearest}) < \rho(p,q)$ and the branch with the objects outside the ball is pruned if $M - \rho(q, x_{nearest}) > \rho(p,q)$. Yianilos has described an implementation of the vantage point tree with sampled selection of the vantage point [38].

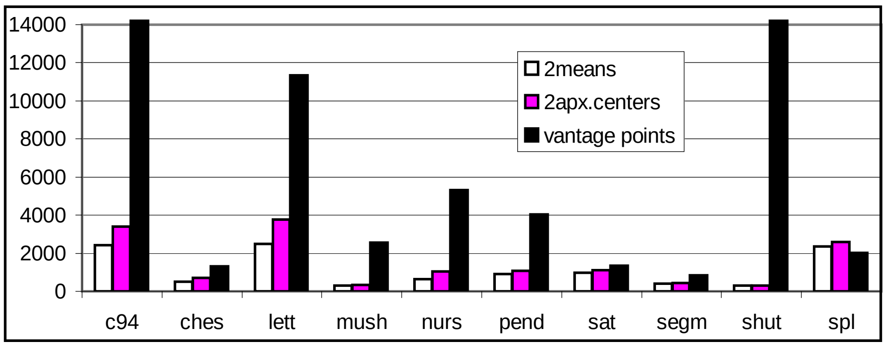


Figure 9. The average number of distance computations of the 100-nn search algorithm in the indexing trees with the k-means, the k-approximate-centers and the vantage point based splitting procedures.

For experimental comparison we have implemented this structure as described by Yianilos.

Figure 9 presents the comparison of the performance of the trees with the 2-means, the 2-approximate-centers and the vantage point based splitting procedures (since the vantage point tree is a binary tree we used $k = 2$ for comparison). The results shows a large advantage of the trees based on the centers over the tree based on the vantage points. In order to save the space we have not presented the results for the 1-nn search but they are similar to the 100-nn case. The results indicate that the center based representation of tree nodes provides better search pruning properties than the vantage point based representation.

12. Indexing vs Searching

The results from the previous section have proved that the k-means indexing tree is a good accelerator while searching for the nearest neighbors. The question arises whether the cost of constructing a tree is not too large in comparison to the search process.

Figure 10 presents the comparison between the number of distance computations per single object in the indexing process (in other words the average cost of indexing a single object) and the average number of distance computations in the 100-nn search.

The results for the k-means and the k-approximate centers procedures are similar. For $k = 2$ they are quite optimistic, for all tested data sets except *shuttle* the cost of indexing a single object is few times lower than the cost of searching for the 100 nearest neighbors of a single object. It means that if the size of the training and the test sets is of the same order the main workload remains on the side of the search process. For the data sets *shuttle* and *mushroom* the differences in the cost are smaller but it results from the fact that the search process is more effective for these two data sets than for the others.

The situation changes to worse while increasing the degree k. For the 5-means case the cost of indexing for five data sets: *chess, covertype, mushroom, segment* and *shuttle* is at least comparable and sometimes higher than the cost of searching. It has been mentioned in Section 9 that the computational cost of searching is stable for $k \geq 3$. On the other hand, the cost of indexing significantly increases while increasing the degree k. It means that the larger degree k the lower profit from applying the advanced indexing structure. The results from this section and Section 9 indicate that the best trade-off between the indexing cost and the search performance is obtained at $k = 2$ or $k = 3$ and increasing the value of

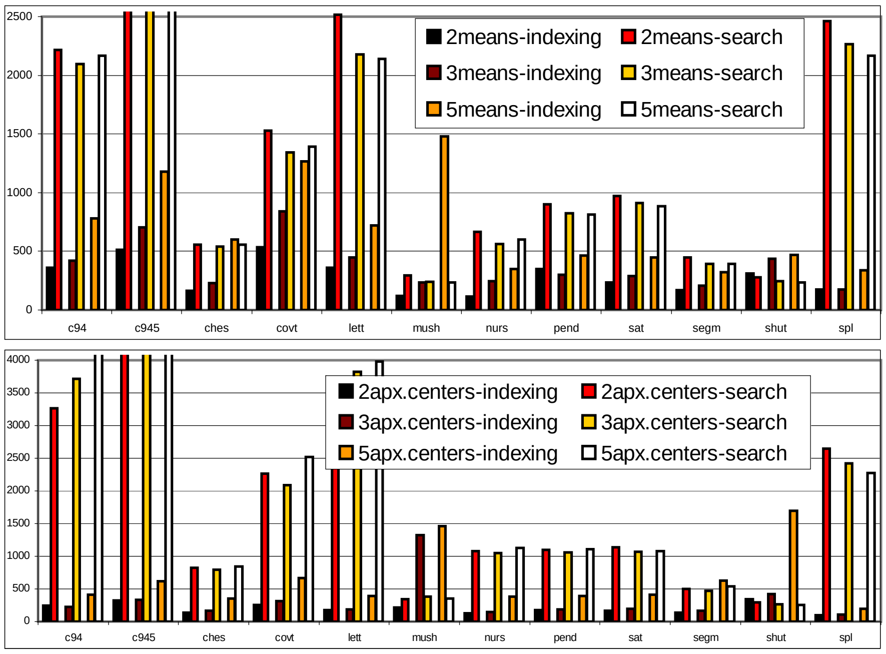


Figure 10. The average number of distance computations per single object in the indexing and the 100-nn search processes for the indexing trees with the k-means (the upper graph) and the k-approximate-centers (the lower graph) splitting procedures. For each data set the first pair of columns represents the costs of indexing and searching for $k = 2$, the second pair represents the costs for $k = 3$ and the third one for $k = 5$.

k more increases the cost of indexing without any profit from searching.

We have analyzed the case of the 100-nn searching. In many application, e.g., while searching for the optimal value of k or for geometrical properties in a data set, there is a need to search for a large number of nearest neighbors and in this case the presented trees keep the appropriate balance between the costs of construction and searching. We have not discussed the results for the 1-nn case since they are not uniformly interpretable. The usefulness of the presented structures for queries with a small k depends more on individual properties of a data set and on the number of queries to be performed.

The upper graph at Figure 11 provides some information about the shape of the indexing trees. The fact that the height of a tree, i.e., the distance between the root and the deepest leaf, rarely exceeds 25 indicates that the shape of a tree is quite balanced: it does not contain very long thin branches. The lower graph presents the average number of iterations in a splitting procedure. In many experiments, especially for $k = 2$, this number does not exceed 5 what indicates that the construction cost for a tree with the iterative splitting procedure is usually a few times larger than for a tree with the non-iterative splitting.

13. Conclusions and Future Research

The research reported in the paper attempts to analyze different properties of the indexing algorithms with the center based splitting procedures and to work out the optimal parameters of the indexing algorithm. For the analysis the well-known VDM metric [7, 12, 14, 18, 19, 27, 30] has been used as a distance

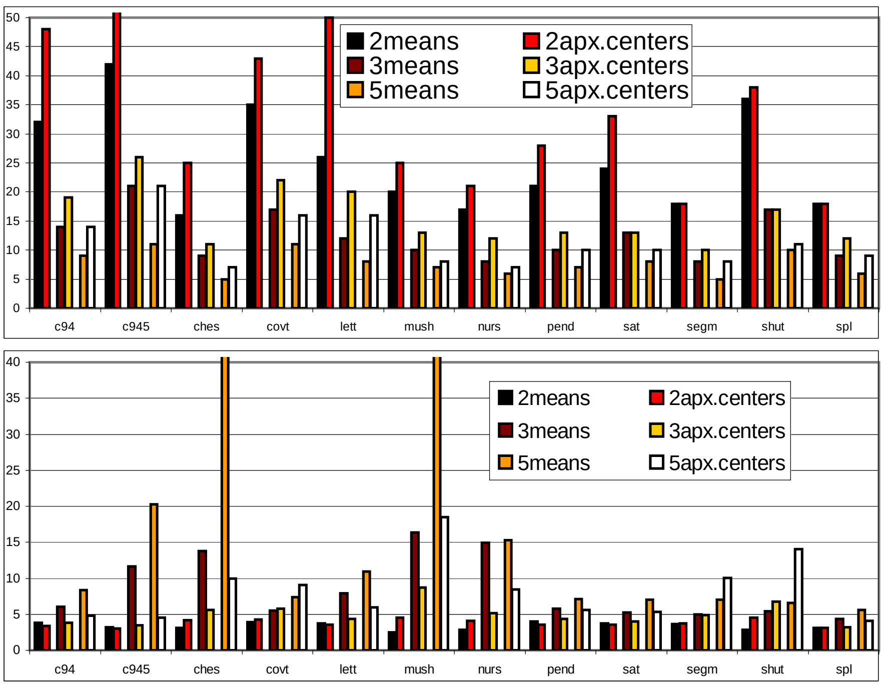


Figure 11. The height of a tree (the upper graph) and the average number of iterations of a splitting procedure (the lower graph) in the k-means based and the k-approximate-centers based indexing trees. For each data set the first pair of columns represents the height (at the upper graph) and the iterations (at the lower graph) for the 2-means and the 2-approximate-centers based trees, the second pair represents these values for $k = 3$ and the third one for $k = 5$.

measure because the results from the literature indicate that it is one of the best measures used for the k-nn classification model.

We proposed the iterative procedure for splitting nodes of an indexing tree: the k-means algorithm in the case of a vector space and the k-approximate-centers in the case of a metric space. We have compared three methods for selection of initial centers in an iterative splitting procedure: the random, the sampled and the global. Savaresi and Boley have reported that the 2-means algorithm has good convergence properties [28] so the selection of initial centers is not very important. This result has been confirmed with the experimental results for most data sets. However, it is obtained for an infinite theoretical model and there are real-life data sets where the global selection of initial centers gives a little better performance than the other two methods. We have also observed that the performance of trees with different splitting degrees is comparable except for the fact that $k = 2$ has a little worse performance than the remaining values $k \geq 3$. On the other hand the cost of indexing increases significantly while increasing the degree k. These observations lead to the conclusion that the value $k = 3$ is the optimal trade-off between the performance of the search process and the cost of indexing.

The representation of tree nodes is based on the centers and we have compared the significance of three different search pruning criteria based on this representation. The two of them are based on the covering radius and the hyperplanes, the third one based on rings requires more information to store at tree nodes. As the experimental results indicate the most important one is the covering radius. In case of searching for the large number $k = 100$ of nearest neighbors this single criterion is equally efficient as all three criteria combined together. In case of $k = 1$ none of the tree criteria alone is comparable to all of them together but the combination of two of them: the covering radius and the hyperplane criteria is. These results indicate that two simple criteria define the optimal combination and there is no need to search for more sophisticated mechanisms like the rings based criterion.

The center based indexing trees outperform the vantage point trees. However, the performance of a center based tree still depends much on how the center of a set of data objects is constructed or selected. While comparing the iterative k-means algorithm to the non-iterative one the advantage of the former one is visible but the k-means algorithm is applicable only to vector spaces. As a general solution we have proposed the approximate centers with selection from a sample instead of the means. Although there are some evidences that the approximate centers perform a little better than the non-iterative centers the difference is small and does not seem to be significant whereas the gap between the performance of the means and the approximate centers is much larger. These observations indicate that the representation of the center of a set of data objects is crucial for the performance of the center based search pruning and we consider the problem of the center selection as an important issue in our future research.

The experimental results show that the tree with the iterative 3-means splitting procedure and the combined search pruning criteria is up to several times as effective as the one-step based tree with a single criterion. A particularly advanced acceleration level in comparison to the linear search has been reached in case of the largest tested data sets. The presented structure has reduced the 1-nn search cost 4000 times in case of the data set *covertype* and 400 times in case of the data set *census94-95*. For the 100-nn search the reductions in cost are 300 and 60 times, respectively. These results indicate that the only constraint for the size of a data set to be searched is the memory capacity used for storing data.

It is known that bottom-up constructions give a very good performance but such an immediate construction requires $O(n^2)$ time. On the other hand, the top-down constructions presented in the paper are limited by strong assumptions and they may not fit to data well. Brin, in conclusions of [8], has considered the iterative transformation of a tree from a top-down construction to a bottom-up construction in such a way that at each iteration the tree is constructed with the use of the structure from the previous iteration. As a future work we consider to extend this idea and to develop a structure that reflects more geometrical properties of a data set to be represented.

Acknowledgments

The author is very grateful to professor Andrzej Skowron for useful remarks on this presentation. This work was supported by the grants 8 T11C 009 19, 8 T11C 025 19 and 4 T11C 040 24 from the Polish State Committee for Scientific Research.

References

[1] Aggarwal, Ch.C., Hinneburg, A., Keim, D.A. (2001), *On the surprising behaviour of distance metrics in high dimensional space.* Proceedings of the Eighth Internatinal Conference on Database Theory (ICDT-2001), London, UK, pages 420-434.

[2] Aha, D.W. (1998). *The omnipresence of case-based reasoning in science and application.* Knowledge-Based Systems, 11 (5-6), pages 261-273.

[3] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B. (1990). *The $R^{\star}$-tree: an efficient and robust access method for points and rectangles.* Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, pages 322-331.

[4] Bentley, J.L. (1975). *Multidimensional binary search trees used for associative searching*. Communications of the ACM, 18(9).

[5] Berchtold, S., Keim, D., Kriegel, H.P. (1996). *The X-tree: an index structure for high dimensional data.* Proceedings of the Twenty Second International Conference on Very Large Databases (VLDB-1996), pages 28-39.

[6] Beyer, K.S., Goldstein, J., Ramakrishnan, R., Shaft, U. (1999). *When Is "Nearest Neighbor" Meaningful?* Proceedings of the Seventh International Conference on Database Theory (ICDT-1999), Jerusalem, Israel, pages 217-235.

[7] Biberman, Y. (1994). *A context similarity measure.* Proceedings of the Ninth European Conference on Machine Learning (ECML-1994), Catania, Italy, pages 49-63.

[8] Brin, S. (1995). *Near neighbor search in large metric spaces.* Proceedings of the Twenty First International Conference on Very Large Databases (VLDB-1995), pages 574-584.

[9] Blake, C.L., Merz, C.J. (1998). *UCI Repository of machine learning databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html], Department of Information and Computer Science, University of California, Irvine, CA.

[10] Chavez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.L. (1999). *Saerching in metric spaces.* Technical Report TR/DCC-99-3, Department of Computer Science, University of Chile.

[11] Ciaccia, P., Patella, M., Zezula, P. (1997). *M-tree: an efficient access method for similarity search in metric spaces.* Proceedings of the Twenty Third International Conference on Very Large Databases (VLDB-1997), pages 426-435.

[12] Cost, S. and Salzberg, S. (1993). *A weighted nearest neighbor algorithm for learning with symbolic features.* Machine Learning, 10, pages 57-78.

[13] Cover, T.M. and Hart, P.E. (1967). *Nearest neighbor pattern classification.* IEEE Transactions on Information Theory, 13, pages 21-27.

[14] Domingos, P. (1996). *Unifying instance-based and rule-based induction.* Machine Learning, 24(2), pages 141-168.

[15] Duda, R.O. and Hart, P.E. (1973). *Pattern classification and scene analysis.* Wiley, New York, NY.

[16] Finkel, R., Bentley, J. (1974). *Quad-trees: A data structure for retrieval and composite keys*. ACTA Informatica 4(1), pages 1-9.

[17] Fukunaga, K., Narendra, P.M. (1975). *A branch and bound algorithm for computing k-nearest neighbors.* IEEE Transactions on Computers, 24(7), pages 750-753.

[18] Góra, G., Wojna, A. (2002). *RIONA: a classifier combining rule induction and k-nn method with automated selection of optimal neighbourhood.* Proceedings of the Thirteenth European Conference on Machine Learning (ECML-2002), Helsinki, Finland.

[19] Góra, G., Wojna, A. (2002). *RIONA: a new classification system combining rule induction and instance-based learning.* Fundamenta Informaticae, 51(4), pages 369-390.

[20] Guttman, A. (1984). *R-trees: A dynamic index structure for spatial searching.* Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, Boston, MA, pages 47-57.

[21] Kalantari, I. and McDonald, G. (1983). *A data structure and an algorithm for the nearest point problem.* IEEE Transactions on Software Engineering, 9 (5).

[22] Katayama, N., Satoh, S. (1997). *The SR-tree: an index structure for high dimensional nearest neighbor queries.* Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, pages 369-380.

[23] Lin, K.I., Jagadish, H.V., Faloustos, C. (1994). *The TV-tree: an index structure for high dimensional data*. VLDB Journal, 3(4), pages 517-542.

[24] Mitchell T.M. (1997). *Machine learning*. McGraw-Hill, Portland.

[25] Nievergelt, J., Hinterberger, H., Sevcik, K. (1984). *The grid file: an adaptable symmetric multikey file structure.* ACM Transactions on Database Systems, 9(1), pages 38-71.

[26] Robinson, J. (1981). *The k-d-b-tree: A search structure for large multi-dimensional dynamic indexes.* Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, ACM, New York, pages 10-18.

[27] Salzberg, S. (1991). *A nearest hyperrectangle learning method.* Machine Learning, 2, pages 229-246.

[28] Savaresi, S.M., Boley D.L. (2001), *On the performance of bisecting K-means and PDDP.* Proceedings of the First SIAM International Conference on Data Mining (ICDM-2001), Chicago, USA.

[29] Sellis, T., Roussopoulos, N., Faloustos, C. (1987). *The R+-tree: A dynamic index for multi-dimensional objects.* Proceedings of the Thirteenth International Conference on Very Large Databases (VLDB-1987), pages 574-584.

[30] Stanfill, C. and Waltz, D. (1986). *Toward memory-based reasoning.* Communications of the ACM, 29, pages 1213-1228.

[31] Uhlmann, J. (1991). *Satisfying general proximity/similarity queries with metric trees.* Information Processing Letters, 40, pages 175-179.

[32] Veloso, M. (1994). *Planning and learning by analogical reasoning*. Springer 1994.

[33] Ward, J.Jr (1963). Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association, 58, pages 236-244.

[34] Weber, R., Schek, H.J., Blott, S. (1998). *A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces*. Proceedings of the Tenty Fourth International Conference on Very Large Databases (VLDB-1998), pages 194-205.

[35] Wettschereck, D. (1994). *A study of Distance-Based Machine Learning Algorithms*. Doctor of Philosophy dissertation in Computer Science, Oregon State University.

[36] Wettschereck, D., Aha, D.W., Mohri, T. (1997). *A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms*. Artificial Intelligence Review 11, pages 273-314.

[37] White, D.A., Jain R. (1996). *Similarity indexing with the SS-tree.* Proceedings of the Twelve International Conference on Data Engineering (ICDE-1996), New Orleans, USA, pages 516-523.

[38] Yianilos, P.N. (1993). *Data structures and algorithms for nearest neighbor search in general metric spaces*. Procee Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, Austin, Texas, pages 311-321.