# Center-Based Indexing for Nearest Neighbors Search

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 the k nearest neighbors (k-nn) search. It presents a tree-based top-down indexing method that uses an iterative $k$-means algorithm for tree node splitting and combines three different search pruning criteria from BST, GHT and GNAT into one. The experiments show that the presented indexing tree accelerates the k-nn searching up to several thousands times in case of large data sets.*

## 1 Introduction

In the similarity based searching problem first a distance measure is defined on data objects and next the problem is to find k objects from a database that are nearest to a given query object. The problem is important for multimedia, machine learning and data mining applications. To reduce the cost of searching one can construct an indexing tree. Such a tree is built using the top-down strategy starting with the whole data set at the root of a tree and recursively at each node splitting data objects into a fixed number $k$ of smaller clusters. The search algorithm traverses the constructed tree in the depth-first order and tries to discard some nodes from searching.

Great number of indexing methods (R-, R*-, R+-, X-, TV-, SS-, M-trees) described in the literature concern the case when data are stored on disk [6]. However, one of the most popular application of the k-nn method is object classification. It requires fast access to data and often the only solution is to assume that 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 researchers working in other application areas too. In the paper we focus on this case.

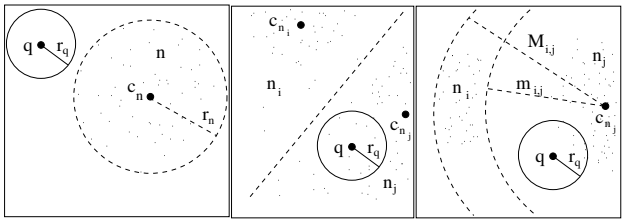
The first main memory based indexing structures for k-nn searching are k-d- and quad-trees [1, 4]. They use iso-oriented hyperplanes to split objects at each node of an indexing tree. BST [8] and GHT [10] use a more advanced algorithm for splitting nodes. It selects randomly two child node centers among objects in the parent node and assigns each object to the nearest center. This procedure may produce splitting hyperplanes in an arbitrary direction, what is more effective for the search process. Both trees have the same construction but different search pruning criteria are used: the covering radius in BST and the hyperplane cut in GHT. GNAT [3] is a more advanced version of the BST/GHT tree. To balance the tree GNAT computes the number of child nodes for each node separately. It uses the same splitting procedure as in BST and GHT but the centers for child nodes are selected more carefully. Finally, it uses more sophisticated search pruning criterion.

Our method differs from the above indexing structures in two ways. First, we use an iterative splitting procedure instead of a one-step procedure. Second, as a search pruning criterion we propose the combination of three different criteria from BST [8], GHT [10] and GNAT [3]. The experiments with real-life data show that the indexing tree with the iterative $k$-means based splitting procedure and the complex search pruning criteria is several times faster than the tree with a one-step splitting procedure and any single criterion. Hence, in case of large databases it can accelerate searching even up to several thousands times in comparison to the linear search.

## 2 Indexing

We assume that data objects are represented by vectors from a $d$-dimensional vector space $\mathbb{X}$ with a distance function $\rho : \mathbb{X}^2 \rightarrow \mathbb{R}_{\geq 0}$ satisfying the triangular inequality. 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 a metric and each data object $x \in \mathbb{U}$ is labeled with a decision. The metric combines the distances specialized for numerical and for symbolic attributes [7]. For numerical attributes we use the difference between attribute values normalized by the largest observed attribute value difference and for symbolic attributes the VDM metric (two symbolic values are similar if they have similar decision

**Figure 1. Search pruning: (a) the covering radius (b) the hyperplane cut (c) the rings**




distribution). In the second phase of the metric induction process the weights of attributes are adjusted to optimize the performance of the metric on the training set $\mathbb{U}$. The results in [7] show that this metric with the k-nn algorithm gives classification accuracy at least as good as other widely used classification methods such as C5.0.

To construct an indexing tree we use an indexing strategy introduced in [5]. It starts with the whole training data set $\mathbb{U}$ and recursively splits data objects into a fixed $k$ smaller clusters. After each split the new child nodes are inserted to the global priority queue and the node with the largest weight is selected from the queue to be split as the next. Each tree node has the center and the weight of a node is defined as the sum of the distances between the node members and the center. 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. This stopping criterion reflects the trade-off between the optimality of searching and the additional memory usage.

For node splitting procedure we use the $k$-means algorithm. Initially, it selects $k$ objects from the parent node as the centers of the child nodes. Then it assigns each object in the parent node to the child node with the nearest center and computes the means of the child nodes as the new centers. It iterates the assignment procedure until the same set of centers is obtained in two subsequent iterations.

For initial centers selection in the $k$-means algorithm we use the following global method. First the mean of a node is computed. As the first seed the farthest object from the mean is picked. Then the farthest object from this one is picked. Then the object that is farthest from these two is chosen, i.e., such that the minimum distance from the previous two seeds is the greatest one among all unchosen objects. Then the one farthest from these three is picked and so on until there are $k$ data objects chosen. The computational cost is $O(ndk^2)$ where $n$ is the size of the node and $d$ is the number of attributes. For small values of $k$ this cost is still acceptable. The above global method provides a little better results than the GNAT sampled and the random methods and this method was used in our experiments.

# 3 Searching

The searching algorithm is assumed to find a fixed number k of data objects nearest to a query object $q$. It traverses an indexing tree in the depth-first order and stores the k nearest data objects from already visited nodes in $nearestQueue$. The objects in $nearestQueue$ are sorted in the increasing order of the distance to the query $q$. The algorithm starts with the empty $nearestQueue$ and visits tree nodes unconditionally as long as $nearestQueue$ contains less than k objects. Since then the algorithm checks at each tree node $n$ with search pruning criteria whether $n$ is to be visited, i.e., whether $n$ can contain an object that is closer to the query $q$ than any other previously found nearest neighbor in $nearestQueue$. If not, the whole subtree of the node $n$ is discarded from searching. Otherwise, if 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 visits child nodes in the increasing order of the distance between the center of a child node and the query $q$. This heuristics guides the algorithm first to child nodes that are more probable to have data object close to the query $q$, what makes the searching considerably more effective than random order of visiting.

There are different search pruning criteria described in the literature and all of them are based on the triangular inequality. Figure 1 presents three different criteria for pruning tree nodes. The value $r_q$ denotes the distance $\rho(q, x)$ between the query $q$ and the farthest from $q$ object $x \in nearestQueue$. The most common criterion applied in BST [8] uses the covering radius (Figure 1a). Each node $n$ keeps the center $c_n$ 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 proposed another criterion for his Generalized Hyperplane Tree (GHT) [10] based on the assumption that the splitting procedure assigns each object to the node with the nearest center. It uses the hyperplanes separating the child nodes of the same parent (Figure 1b). 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 $n_j$:

$$\rho(c_{n_i}, q) - r_q > \rho(c_{n_j}, q) + r_q.$$

GNAT pruning criterion [3] is also based on mutual relation among brother nodes (Figure 1c). If the degree of a tree 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:

$$m_{i,j} = \min_{x \in n_i} \rho(c_{n_j}, x) \text{ and } M_{i,j} = \max_{x \in n_i} \rho(c_{n_j}, x).$$

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.

## 4 Experimental results

We have performed experiments with different indexing and search methods for 6 benchmark data sets from the UCI repository [2] (the indexed and the query set sizes are given in parenthesis): *census94* (30162, 15060), *census94-95* (199523, 99762), *covertype* (387308, 193704), *letter* (15000, 5000), *nursery* (8640, 4320), *pendigits* (7494, 3498). The data sets provided as a single file (*covertype*, *nursery*) have been randomly split into an indexed and a query parts with the ratio 2 to 1, the others have been tested with the originally provided partition.

The $k$-means splitting procedure used in our method selects initial centers, assigns each data object to the nearest center and computes the means as the new centers of clusters. Then assignment of data objects to the centers and computation of new cluster centers is iterated until the same set of cluster centers is generated in two subsequent iterations. The one-step splitting procedure used in the other indexing trees (BST, GHT and GNAT) stops after the first iteration and uses the initial centers as the final. The interesting question is how much the search process profits from the additional cost due to the iterative $k$-means splitting procedure and the combined search pruning criterion in comparison to the one-step case with a single pruning criterion. To answer this question we tested the iterative $k$-means based and the one-step $k$-centers based trees.

First we analyzed the performance of the $k$-means indexing tree as a function of the degree $k$ by testing all values in the range $2 \leq k \leq 9$. The experiment showed 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 farther experiments we have used the degree $k = 3$ in the $k$-means based indexing tree. A similar experiment was performed for the one-step $k$-centers based tree, for which $k = 2$ was the optimal.

**Figure 2. The average number of distance computations per single object of the 1-nn (the upper graph) and the 100-nn (the lower graph) search in different indexing trees**

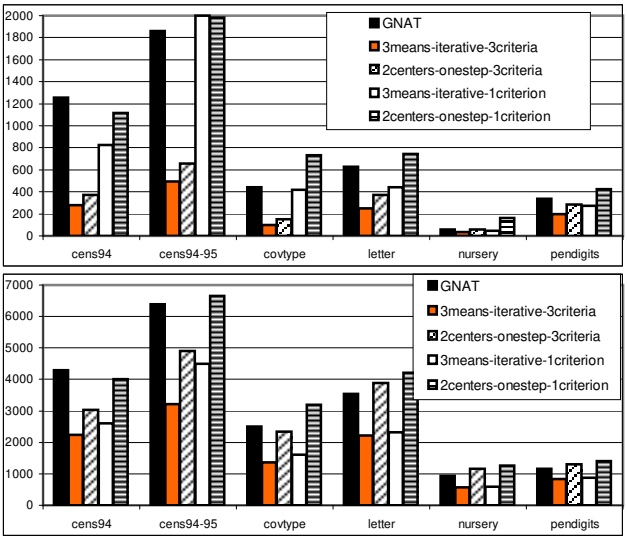



Figure 2 presents the cost of searching in the trees with the iterative $k$-means and the one-step $k$-centers splitting procedures. The performance of searching is presented for two cases: with all 3 pruning criteria and with a single covering radius criterion. We chose this single criterion for comparison since it has the best performance among all three tested criteria. For comparison we also present the performance of the GNAT tree [3]. We implemented different structures from the literature [3, 8, 10] and GNAT had the best performance among them. To make the results comparable all indexing trees were tested with the same distance function and the same partition for each data set.

While comparing the performance of the iterative 3-means (the second column) and the one-step 2-centers (the third column) procedures the profit from applying the iterative procedure is visible: it ranges from 20% to 50% and is similar for the 1-nn and the 100-nn search. The good experimental performance of the tree with the $k$-means procedure may result from the theoretical property proved by Savaresi and Boley [9]. They show that in an infinite theoretical model of data the 2-means procedure with random selection of initial centers converges to the partition orthogonal to the principal direction, what is in a sense the optimal partition of data.

The comparison of the second and the fourth column shows that the application of the combined pruning crite-

rion also accelerates the performance of the k-nn search in relation to a single criterion. In case of the 1-nn search the acceleration is visible for all data sets and reaches up to several times for the largest sets. In case of the 100-nn search the difference is visible only for three larger data sets (*census94, census94-94, covertype*) and is much smaller than in the 1-nn case. It indicates that the less number of neighbors k and the greater size of a data set, the improvement is more significant. We have compared the performance of all combinations among the three presented criteria and 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 hyperplane cut provide the optimal pruning combination and there is no need to search for more sophisticated pruning mechanisms.

The experimental results show that the tree with the iterative 3-means splitting procedure and the combined search pruning criteria (the second column) is up to several times as effective as the one-step based tree with a single criterion (the fifth column). 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. Such good performance has been reached both due to the improved splitting procedure and the complex search criterion.

The question arises whether the cost of constructing the $k$-means based tree is not too large in comparison to the cost of searching. We have compared the average cost of indexing a single object to the average cost of searching for nearest neighbors of a single object. In case of a small number of neighbors the results are not uniformly interpretable and the usefulness of the presented structure depends on individual properties of a data set and on the number of queries to be performed. However, while estimating the optimal size of a neighborhood or searching 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 tree keeps the appropriate balance between the costs of construction and searching: for all tested data sets the average cost of indexing a single object was lower than the average cost of the 100-nn search, usually several times lower. It means that if the size of an indexed database and the number of queries are of the same order the main workload remains on the side of the search process. The cost of indexing increases while increasing the degree of a tree $k$. The cost of searching is stable for $k \geq 3$. It indicates that the best trade-off between the indexing cost and the search performance is obtained at $k = 3$ and by increasing the value of $k$ the cost of indexing is increased without any profit for searching.

# 5 Summary

In the paper we present the searching tree with the iterative $k$-means splitting procedure and the combined search pruning criteria that is up to several times better than the one-step based tree with a single criterion and is particularly effective while indexing very large data sets. The effectiveness of indexing structures is measured by the average number of distance computations in a single k-nn search what allows us to measure the acceleration of searching in comparison to the linear search. Almost 100% of run-time is used by distance computation operations and the measured acceleration factors correspond directly to the real-time acceleration. The presented tree is used for the k-nn classifier included in RSES system (http://logic.mimuw.edu.pl/˜rses).

**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 and 4 T11C 040 24 from the Polish State Committee for Scientific Research and by the grant from Ministry of Scientific Research and Information Technology.

# References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[2] C. L. Blake and C. J. Merz. 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, 1998.

[3] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the Twenty First International Conference on Very Large Databases*, pages 574–584, 1995.

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

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

[6] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

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

[8] I. Kalantari and G. McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, 9(5):631–634, 1983.

[9] S. M. Savaresi and D. L. Boley. On the performance of bisecting K-means and PDDP. In *Proceedings of the First SIAM International Conference on Data Mining*, pages 1–14, Chicago, USA, 2001.

[10] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.