Impact of the Initialization in Tree-Based Fast Similarity Search Techniques

Aureo Serrano, Luisa Micó, and Jose Oncina

Departamento de Lenguajes y Sistemas Informáticos,
Universidad de Alicante,
P.O. box 99, E-03080 Alicante, Spain
{aserrano,mico,oncina}@dlsi.ua.es
http://www.dlsi.ua.es

Abstract. Many fast similarity search techniques relies on the use of pivots (specially selected points in the data set). Using these points, specific structures (indexes) are built speeding up the search when queering. Usually, pivot selection techniques are incremental, being the first one randomly chosen.

This article explores several techniques to choose the first pivot in a tree-based fast similarity search technique. We provide experimental results showing that an adequate choice of this pivot leads to significant reductions in distance computations and time complexity.

Moreover, most pivot tree-based indexes emphasizes in building balanced trees. We provide experimentally and theoretical support that very unbalanced trees can be a better choice than balanced ones.

1 Introduction

Similarity search has become a fundamental task in many application areas, including data mining, pattern recognition, computer vision, multimedia information retrieval, biomedical databases, data compression or statistical data analysis. The simplest, yet most popular method for this task is the well-known k-Nearest Neighbor (kNN) classifier. However, one of the main constraints of using this technique to classify large datasets is its complexity: finding the k-Nearest Neighbors for a given query is linear on the database size. A classical method to speed up the search is to rely in some property of the dissimilarity measure to build up a data structure (index) in preprocess time. Once the index has been built, similarity queries can be answered with a significant reduction in the number of distance computations.

In this work we are going to focus our attention in dissimilarity functions that fulfills the conditions of being a distance and then define a metric space (A review of such techniques can be found in [3][7][18].) According to Navarro and Reyes [12], algorithms to search in metric spaces can be divided in pivot-based and clustering algorithms:

- *Pivot-based* algorithms use a set of distinguished objects (pivots) of the datasets. Usually, the distances between pivots and some (or all) the objects in the database are stored in the index. This information, along with

M. Pelillo and E.R. Hancock (Eds.): SIMBAD 2011, LNCS 7005, pp. 163–176, 2011.

the properties of the distance function, is used in query time to avoid some distance computations.
- *Clustering* algorithms divide the spaces in zones as compact as possible, storing a representative point for each zone and extra information that permit to discard a zone at query time.

According to Hjaltason and Samet [7] and Zezula et al.[18], the algorithms can be divided in:

- *Ball partitioning* algorithms, that requires only one pivot to divide a set S into two subsets using a spherical cut.
- *Generalized hyperplane partitioning* algorithms, where the division is done using two pivots.
- *Distance matrix* algorithms, where precomputed distances between the objects in the datasets are stored to be used in query time.

For example, the Vantage Point Tree (vp-tree) algorithm [17] is a *pivot-based* algorithm that uses *ball partitioning* metric trees. The simplest method selects randomly the vantage points. However, the author argues that a more careful selection procedure can yield better search performance.

One of the first works about the selection of pivots was done by Shapiro in 1977 [14]. He found that if the data set belongs to a uniform distribution on a hypercube in the Euclidean space, it is better to pick (vantage) points near corners of the hypercube as pivots to improve search performance. Taking into account these results, Yianilos argues that choosing the points near corners as pivots can be shown to minimize the boundary of the ball that is inside the hypercube, increasing search efficiency.

Other well-known example of *pivot-based* method was proposed by Ullmann in 1991 [15]. Ullmann defines a metric tree (gh-tree) using *generalized hyperplane partitioning*. Instead of picking just one object for partitioning the space as in the vp-tree, this method picks two pivots, usually, the samples farthest from each other, dividing the set of remaining samples based on the closest pivot. Similar strategies were applied in Faloutsos and Lin [5] or Merkwirth et al. [10].

Despite the taxonomy proposed by Navarro and Reyes, some clustering algorithms also use pivots to build indexes. GNAT (Geometric Near-neighbor Access Tree) [2] is a *clustering* algorithm, that is a generalization of a gh-tree, and then, where more than two pivots should be chosen to divide the data set at each node. The method for choosing the pivot samples is based on a philosophy similar to that of Yianilos for the vp-tree (and also suggested by others [1][14]). The method first randomly selects a set of candidate pivot samples from the dataset. Next, the first pivot is selected randomly from the candidates, and the remaining pivots are selected iteratively as the farthest away from the previously selected.

A bisector tree, proposed by Kalantari and McDonald [8], is a gh-tree augmented with the maximum distance to a sample in its subtree. Moreover, if in this structure one of the two pivots in each non leaf node is inherited from its parent node, the monotonous bisector tree (mb-tree) is obtained [13]. Since this strategy leads to fewer pivot objects, its use reduces the number of distance

computations during search (provided the distances of pivot objects are propagated downward during search), at the lower cost of a worse partitioning and a deeper tree. It should be clear that many different configurations are possible for obtaining a mb-tree, since there are many options to choose the second pivot at each step for the next decomposition. For example, one strategy is to select as the second pivot the one that tries to associate the same number of objects with each node (then, obtaining a balanced tree).

In the last few years we worked with MDF-trees. This indexes can be viewed as a particularization of the mb-tree. In order to build an MDF-tree, first a pivot is chosen from the database (usually randomly). The algorithm proceeds by dividing the database in two sets each time a node of the tree is split. To split a node, two pivots are chosen, the corresponding to the new left node is the pivot of the node to be split and the right pivot is the farthest element of the left pivot. Then, the objects of the original node are distributed according to its nearest pivot. This structure stores some additional information (i.e. the distance from the node pivot to the farthest object of the node) [4].

However, the selection of the first pivot (root of the tree) has not received special attention. Note that in mb-trees, the selection of the first pivot is affecting all levels of the tree because the decomposition of the space is done in a top-down manner.

In this paper we are interested in the initialization of MDF-trees, this index is used as the basis for building other more complex indexes ([11][4]). In this case the initialization involves only the selection of the representative of the root.

In this paper an experimental survey of several initializations for the MDF-tree has been done. Although these initializations involves only the selection of the representative of the tree root, significant variations on the properties of the trees and efficiency of the search can be observed.

Moreover, we show that, in this type of trees, it is not a good idea to force the tree to be balanced. The main reason is that forcing the nodes to be of equal size leads to big overlapping regions. We show that it is better to force to have wide unbalanced trees in order to reduce the overlapping regions.

In the next section, a quick review of the construction of the MDF-tree is presented. In section 3 different initialization proposals are detailed. In section 4 experiments have been carried out on several artificial and real datasets. Finally, some concluding remarks are depicted in Section 6.

2 The MDF-Tree

The MDF tree is a binary indexing structure based on a *hyperplane partitioning* approach [11][4]. The main difference with mb-trees is related to the selection of the representatives (pivots) each time a node is split.

2.1 Building an MDT-Tree

In order to build an MDF-tree, firstly a pivot is randomly selected as the root of the tree (first level). Secondly, the farthest object to the root is chosen as the

second pivot, then the rest of objects are distributed according to the closest pivot. This procedure is recursively repeated until each leaf node has only one object (see Figure 3). It is interesting to observe that the root of the tree is propagated recursively through all the levels of the tree as the left node.

Algorithm 1 describes how an MDF-tree is built. The function `build_tree(`ℓ`,`S`)` takes as arguments the future representative of the root node (ℓ) and the set of objects to be included in the tree (excluding ℓ) and returns the MDF-tree that contains $S \cup \{\ell\}$. The first time that `build_tree(`ℓ`,`S`)` is called, ℓ is selected *randomly* among the data set. In the algorithm, M_T is the pivot corresponding to T, r_T is the covering radius, and T_L (T_R) is the left (right) subtree of T.

Algorithm 1. build_tree(ℓ, S)

Data:
  $S \cup \{\ell\} = D$: set of points to include in T;
  ℓ: future left representative of T
create MDF-tree T
if *S is empty* **then**
  $M_T = \ell$
  $r_T = 0$
else
  $r = \text{argmax}_{x \in S}\, d(\ell, x)$
  $r_T = d(\ell, r)$
  $S_\ell = \{x \in S | d(\ell, x) < d(r, x)\}$
  $S_r = \{x \in S | d(\ell, x) \geq d(r, x)\} - \{r\}$
  $T_L =$ `build_tree(`ℓ`,`S_ℓ`)`
  $T_R =$ `build_tree(`r`,`S_r`)`
end
return T

2.2 The Search Algorithm

Given a query point, the search algorithm proceeds in a top down procedure.

At each step, the search algorithm computes the distance to the representatives of each child node and updates the current nearest neighbor candidate if necessary. Next, using the distance from the sample to the representatives and the radius of the node, it tries to discard each of the nodes. At last, the search continues with each of the undiscarded nodes.

Note that since in MDF-trees the representative of the left node is the same as the representative of its father, the distance computation of the query point to the left representative can be avoided. Then, only one distance is computed each time a node is explored (see alg. 2.)

As pruning rule, here we are going to consider the simplest one. Given a query point x and the current NN candidate nn, no object in the tree T can be nearest to x than the current NN candidate if (see fig. 1)

$$d(x, nn) \leq d(rep(T), x) - r_T$$

Algorithm 2. search_tree(T, x)

```
Data:
   T: tree node;
   x: sample
if not exists T_L then return
if not exists T_R then
   if not pruned T_L then search(T_L, x)
   return
end
d_r = d(rep(T_R), x); update nearest neighbour
if d_ℓ < d_r then
   if not pruned T_L then search (T_L, x)
   if not pruned T_R then search (T_R, x)
end
else
   if not pruned T_R then search (T_R, x)
   if not pruned T_L then search (T_L, x)
end
```

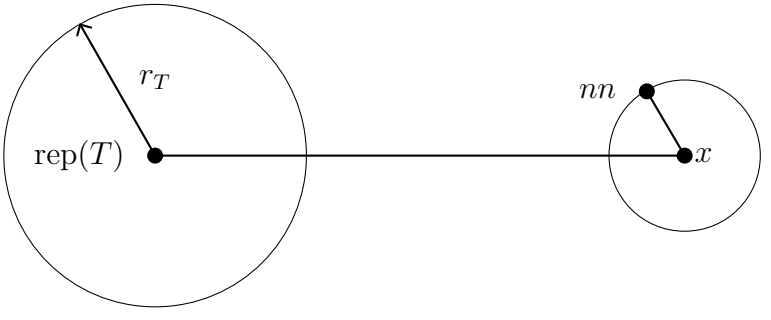

Fig. 1. Pruning rule

Note that as a consequence of this rule if the representatives of the children of a node are too near or their radius are too large, an overlapping region will appear where none of the children can be pruned (see fig. 2). This (usually unavoidable) situation provokes the algorithm to explore both subtrees.

The MDF-tree building procedure tries to weaken this effect by choosing, as representative for the right child, the farthest object of the left representative. Unfortunately, this usually leads to large radius.

3 Initialization Methods

As stated in the previous section, it is necessary to choose a pivot that will act as the root of the tree for construction purposes. To our knowledge, no work has been done before to study alternative initialization choices. In this work, we present experimental results (both for tree construction and for search performance) when performing different root initializations.

Let we enumerate the methods used for this purpose.

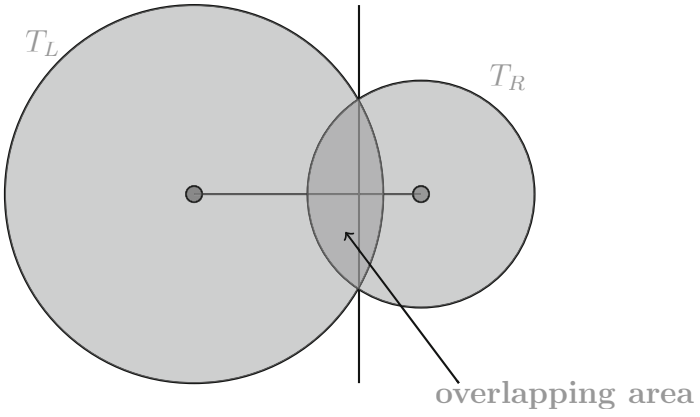

Fig. 2. Overlapping region

3.1 *Random* Method

This initialization is the usual method applied in the MDF-tree [11][4]. It consists on selecting randomly a sample from the database.

3.2 *Outlier* Method

The aim of this method is to choose an outlier as initialization. In this method, we first choose randomly one sample from the database, and then the most distant sample from it is selected as root of the tree.

The hypothesis is that by choosing two, probably, very distant points at the first level, and recursively dividing the space, the resulting subspaces will have similar size, producing a very balanced tree.

Given a dataset D, and given $p \in D$ randomly chosen, we select r as the root of the tree, where

$$r = \text{argmax}_{t \in D}\, d(p, t)$$

Figure 3 is and example of the partition an the MDF-tree produced by a set of points in a two dimensional space. As expected, the tree is reasonably balanced.

3.3 *Median* Method

The aim now is to choose a "centered" point as initialization. In this case, we choose the point that minimizes the sum of the distance to all the others, i.e. the set median of the training set.

Given a dataset D, we select the root of the tree r, where

$$r = \text{argmin}_{p \in D} \sum_{t \in D} d(p, t)$$

An example, using the same data set than in fig. 3, can be seen in fig. 4. Note that now the tree is very unbalanced.

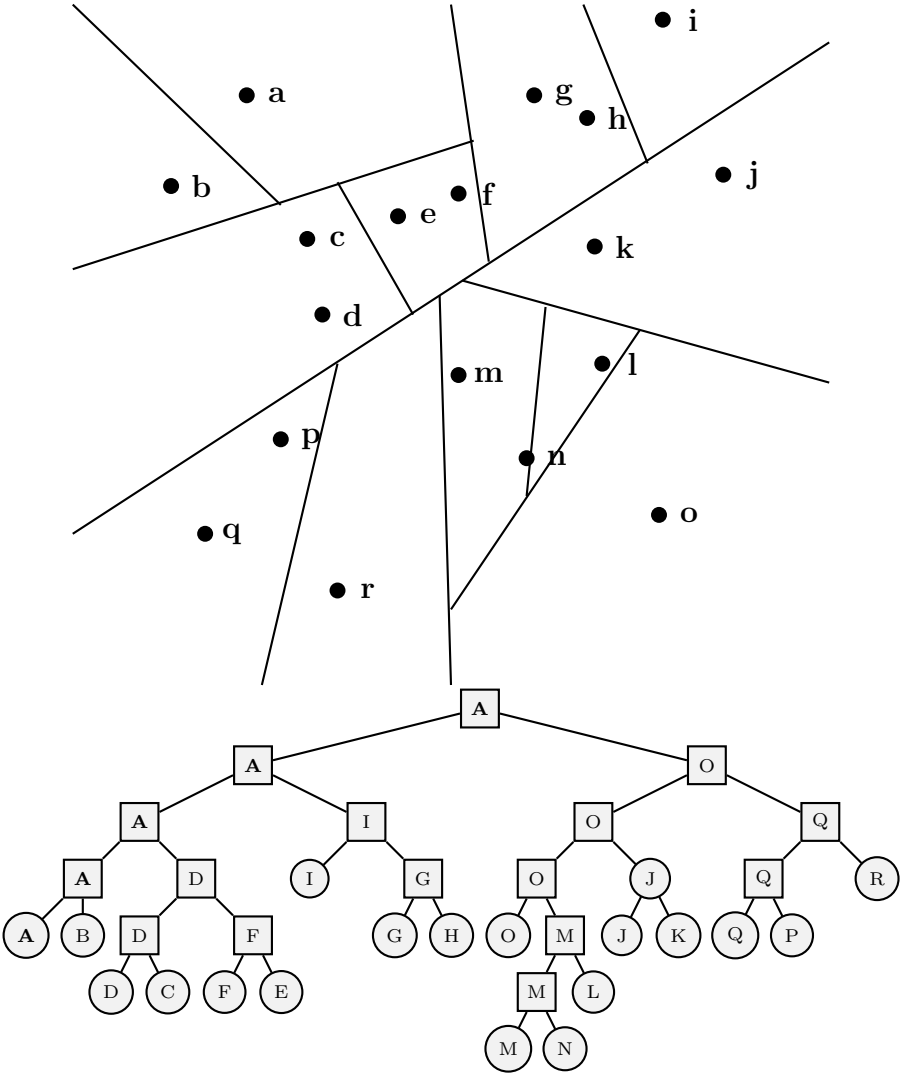

Fig. 3. Example of a space partitioning produced by a MDF-tree in a two-dimensional space (top). Given a random object, "L", in this example the root is the most distant object to "L" (label "A"), and then is propagated through all the levels of the three through the left child. The same criterion is used recursively for the propagation of the rest of pivots. The decomposition continues until there are only one object in each leaf node (down).

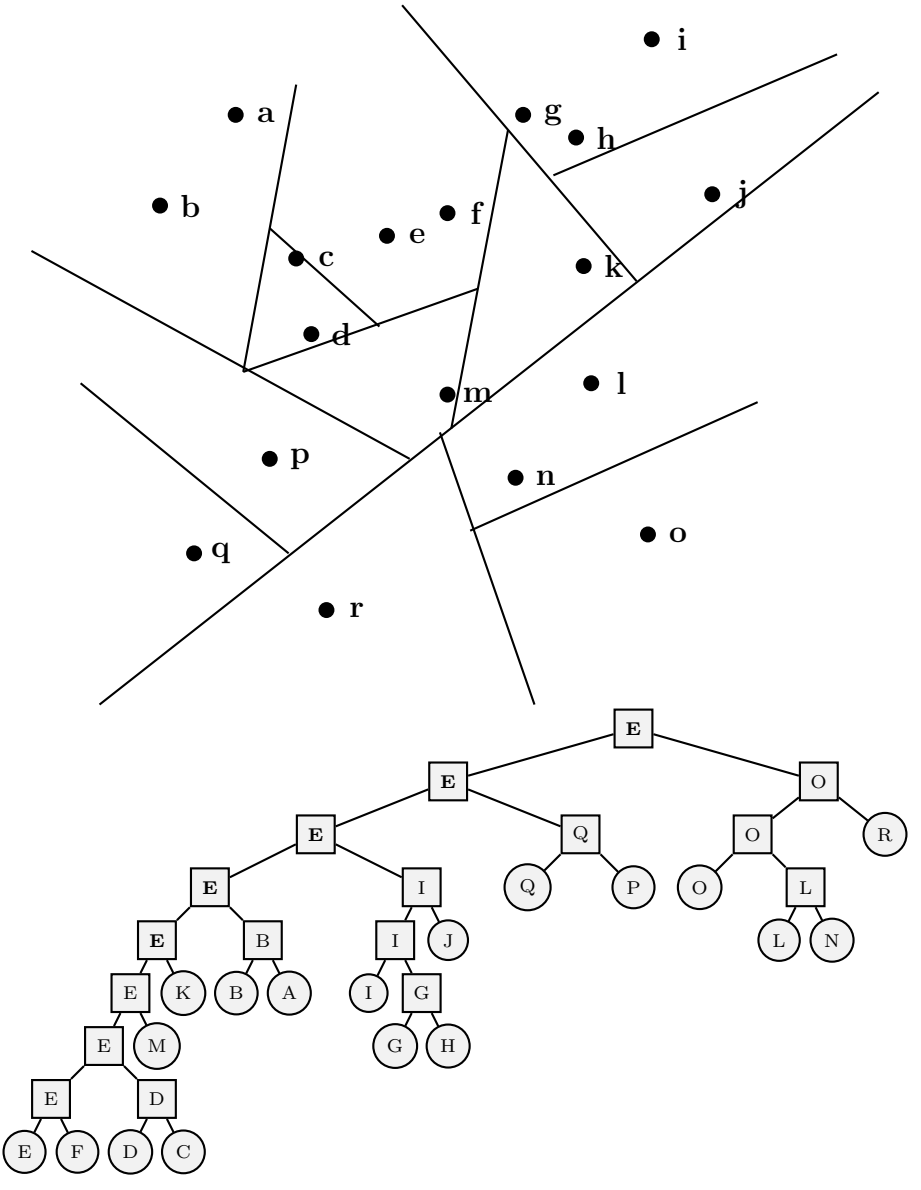

Fig. 4. Example of a space partitioning produced by a MDF-tree in a two-dimensional space (top). In this example the root is the set median of the set. The decomposition continues until there are only one object in each leaf node (down).

4 Experiments

We have carried out a series of experiments using synthetic and real data to study the influence of the different methods for the first pivot selection in the MDF-tree.

Two sets of databases were used in our experiments:

1. Synthetic prototype sets, generated from uniform distributions in the unit hypercube, from dimension 2 to 20. Each point in the plot shows the result of a experiment using 50 000 samples as training set, an 10 000 samples as test. The Euclidean distance was used as dissimilarity measure.
2. Two string databases:
   - A database of 69 069 words of an English dictionary was used. A training set of 50 000 samples with 10 000 test samples were used for the experiments. The words for the samples are randomly chosen from the entire dictionary. In order to obtain reliable results, several experiments were carried out, changing the value of the random seed to obtain different set of words in each case.
   - A database of 61 293 strings representing contour chains [6] of the handwritten digits in NIST database. A training set of 10 000 samples with 1 000 test samples were used for the experiments. Several experiments with a distinct random seed were carried out.

   In both cases, the edit distance [9][16] was used as dissimilarity measure. Edit distance between two strings is the minimum cost sequence of character insertions, deletions and substitutions to make equal the two strings. In our experiments the cost of apply any of the three operations are the same.

Figure 5 shows the depths of the trees using Random, Outlier and Median initializations, for a dataset with 50 000 samples uniformly distributed points in the unit hypercube for dimensions varying form 2 to 20. Unlike the other initializations, the depth of the tree for the Median initialization grows quickly with the dimension of the space. This is due to the fact that, in general, the space around the median is usually more populated than in the other choices.

To analyze the behavior of the trees during the search, several experiments were done. Figure 6 (left) shows the average number of distance computations in a nearest neighbor search for the three initialization. The same datasets as in the previous experiments were used. Figure 6 (ref) shows the time spent, in microseconds, by each search. The time was measured on a cpu running at 2660 MHz under a Linux system.

It can be observed that, unexpectedly, in high dimensional spaces the Median initialization (the method that obtained deepest trees) reduces significantly the number of distance computations with respect to the classical approaches (Random and Outlier).

Similar results have been obtained using the English dictionary and the NIST Database. Results can be seen in tables 1, 2 and 3.

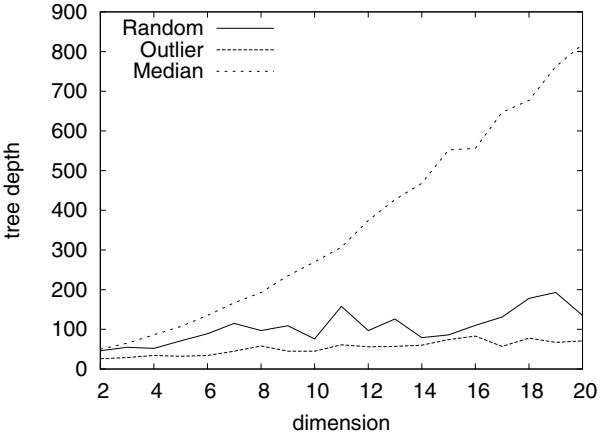

Fig. 5. Tree depths using Random, Outlier and Median initialization in the Euclidean Space

Table 1. Tree depths using Random, Outlier and Median initializations

	Random	**Outlier**	**Median**
English dic.	184.7	97.2	361.8
NIST	137.1	119.8	203.3

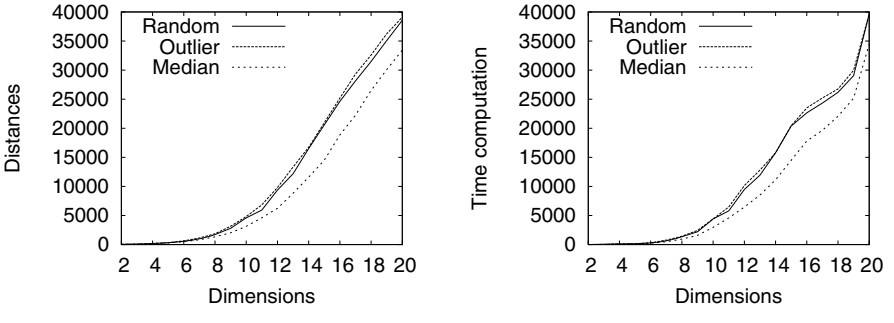

Fig. 6. Average number of distance computations (on the left) and time (in seconds) using Random, Outlier and Median initialization in the Euclidean Space

Table 2. Average number of distance computations using Random, Outlier and Median initializations

	Random	**Outlier**	**Median**
English dic.	4402.6	5324.6	3241.9
NIST	1713.8	1845.9	1501.2

Table 3. Time (in microseconds) using Random, Outlier and Median initializations

	Random	**Outlier**	**Median**
English dic.	16446.6	19906,4	12268.6
NIST	87769.9	95037.9	78800.6

5 Non Balanced Trees

In order to find an explanation to this result the MDF-tree was studied deeply.

Let suppose we have a very big database were a good candidate to NN is found quickly and its distance to the query object is negligible with respect to the radius of the nodes.

In a node we can distinguish four regions depending on two criteria:

- if we are in the left (right) node region.
- if we are in the overlapping region or not.

Let we call r the probability that a random point in the node goes to the left child. Let we call s the probability that a point falls in the overlapping region. We are going to assume now that this probabilities depends only on the size of the node.

In order to check if this is assumable, for each node of some MDF-tree, we have counted the number of points in the node and the number of points in the left child. The trees were obtained from a database of 20 000 points uniformly distributed in 5, 10 and 15 dimensional unit hypercube. The euclidean distance was used as dissimilarity function. The experiments were repeated using the Outlier and the Median initialization techniques. The results of this experiment is shown in fig. 7. Similarly, the number of points in the node versus the number of points in the overlapping region was represented in fig. 8. It can be seen that the ratios are quite linear (perhaps with the exception of the last point). Then the slope gives us the parameter r and s respectivelly. Table 4 shows their values.

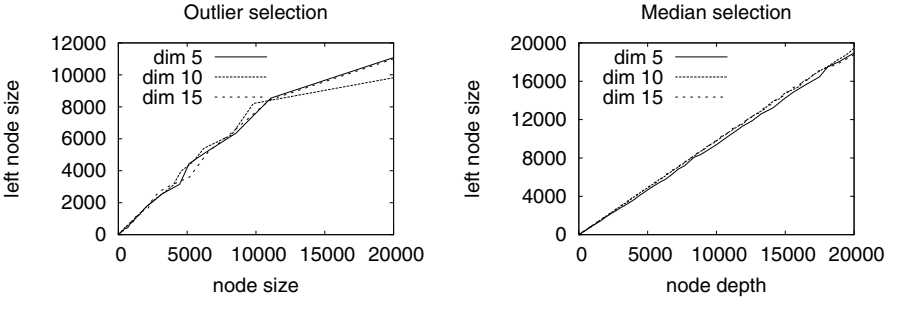

Fig. 7. Left node size versus parent node size in MDF-tree with Outlier and Median initializations for uniformly dirtributed points in the 5, 10 and 15 dimensional unit hypercube

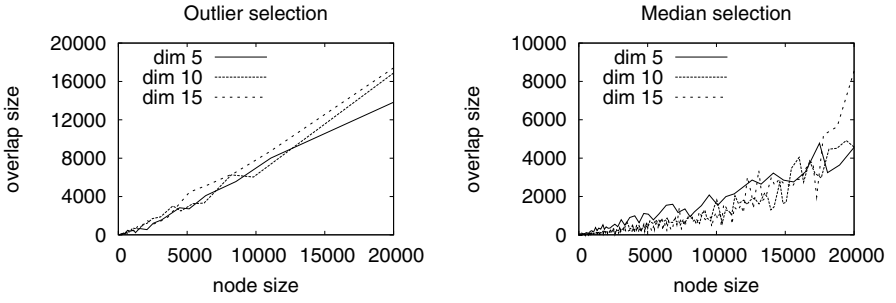

Fig. 8. Orevlapping node size versus parent node size in MDF-tree with Outlier and Median initializations for uniformly distributed points in the 5, 10 and 15 dimensional unit hypercube

Table 4. Values of the parameters r and s

dim.	param. r		param. s	
	Outlier	Median	Outlier	Median
5	0.65	0.94	0.67	0.20
10	0.63	0.97	0.75	0.15
15	0.65	0.97	0.81	0.15

Note that parameter r is smaller for the Outlier than for the Median initialization. That means that the trees are more balanced for the Outlier than for the Median initialization. In fact, for the Median initialization, the trees are very unbalanced.

Note also that parameter s is smaller for the Median than for the Outlier initialization. Assuming the distance to the NN is negligible, if a query point falls in the overlapping region, the algorithm should search in both children, otherwise it search only in one of them.

In this situation, the Median initialization is displacing the frontier of the two children towards the right one and reducing the overlapping region at the expenses of increasing the depth of the tree.

To get an idea of how efficient the effect can be, we can estimate the expected number of distance computations.

In order to do that we need to know the following probabilities:

- Probability of being in left node but not in the overlapping region: $r(1-s)$
- Probability of being in the left node and in the overlapping region: rs.
- Probability of being in the right node and in the overlapping region: $(1-r)s$.
- Probability of being in the right node but not in the overlapping region: $(1-r)(1-s)$.

The the expected number of distance computations in a tree with n objects $(c(n))$ can be expressed as:

$$c(n) = r(1-s)\,c(rn) + s\,[c(rn) + c((1-r)n)] + (1-r)(1-s)\,c((1-r)n) + 1$$
$$= (r(1-s)+s)\,c(rn) + (s + (1-r)(1-s))\,c((1-r)n) + 1$$

and obviously, $c(n) = 1$ if $n \leq 1$.

Table 5 shows the expected number of distance computations corresponding to the parameters in table 4.

Table 5. Expected number of distance computations

dim.	Outlier	Median
5	5502	879
10	10235	2103
15	16291	2374

6 Conclusions

In this work we show that an appropriate initialization technique can lead to significant distance computations reductions in MDF-trees based search algorithms.

Surprisingly, and far from what intuitively was expected, the method that produces a more degenerate tree is the one that computes the least number of distances. Moreover, the reduction is more important as the dimensionality of the data increases.

The lesson learnt from this work is that balanced trees can not be taken as a guide to increase the performance of the algorithm. We have developed a simple theory to explain why very unbalanced trees can lead to significant distance computation reductions.

Many questions remains open:

- Can we devise an expression to link the reduction factor r (which is related with the balance degree of the tree) with the overlapping factor s? This expression will guide us to find which is the optimal relative volume of the chidren nodes.
- In our case, we are manipulating the relative value of the children nodes just by choosing an initial pivot. Can we propagate this idea to the selection of pivots in all the nodes?
- Can we extrapolate this results to other tree based indexes?

Acknowledgements. The authors thank the Spanish CICyT for partial support of this work through projects TIN2009-14205-C04-C1, the Ist Programme of the European Community, under the Pascal Network of Excellence, (Ist–2006-216886), and the program Consolider Ingenio 2010 (Csd2007-00018).

References

1. Bozkaya, T., Özsoyoglu, Z.M.: Indexing large metric spaces for similarity search queries. ACM Trans. Database Syst. 24(3), 361–404 (1999)
2. Brin, S.: Near neighbor search in large metric spaces. In: Proceedings of the 21st International Conference on Very Large Data Bases, pp. 574–584 (1995)
3. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.L.: Searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
4. Gómez-Ballester, E., Micó, L., Oncina, J.: Some approaches to improve tree-based nearest neighbour search algorithms. Pattern Recognition 39(2), 171–179 (2006)
5. Faloutsos, C., Lin, K.: Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, SIGMOD 1995, pp. 163–174. ACM, New York (1995)
6. Freeman, H.: Boundary encoding and processing. Picture Processing and Psychopictorics, 241–266 (1970)
7. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces. ACM Trans. Database Syst. 28(4), 517–580 (2003)
8. Kalantari, I., McDonald, G.: A data structure and an algorithm for the nearest point problem. IEEE Trans. Software Engineering 9, 631–634 (1983)
9. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk 163(4), 845–848 (1965)
10. Merkwirth, C., Parlitz, U., Lauterborn, W.: Fast nearest-neighbor searching for nonlinear signal processing. Physical Review 62, 2089–2097 (2000)
11. Micó, L., Oncina, J., Carrasco, R.C.: A fast branch and bound nearest neighbor classifier in metric spaces. Pattern Recognition Letters 17, 731–773 (1996)
12. Navarro, G., Reyes, N.: Dynamic spatial approximation trees. J. Exp. Algorithmics 12, 1–68 (2008)
13. Noltemeier, H., Verbarg, K., Zirkelbach, C.: Monotonous bisector* trees – a tool for efficient partitioning of complex scenes of geometric objects. In: Monien, B., Ottmann, T. (eds.) Data Structures and Efficient Algorithms. LNCS, vol. 594, pp. 186–203. Springer, Heidelberg (1992)
14. Shapiro, M.: The choice of reference points in best-match file searching. Commun. ACM 20, 339–343 (1977)
15. Uhlmann, J.K.: Satisfying general proximity/similarity queries with metric trees. Inf. Process. Lett. 40(4), 175–179 (1991)
16. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the ACM 21(1), 168–173 (1974)
17. Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, pp. 311–321 (1993)
18. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Springer, Heidelberg (2006)