Some Approaches to Improve Tree Based Nearest Neighbour Search Algorithms

Eva Gómez-Ballester, Luisa Micó and Jose Oncina [1]

Dept. Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03071 Alicante, Spain,
`{eva,mico,oncina}@dlsi.ua.es`

Abstract

Nearest neighbour search is a widely used technique in Pattern Recognition. During the last three decades a large number of fast algorithms have been proposed. In this work we are interested in algorithms that can be used with any dissimilarity function provided that it fits the mathematical notion of distance.

Some of such algorithms organize, in preprocessing time, the data in a tree structure that is traversed in search time to find the nearest neighbour. The speedup is obtained using some pruning rules that avoid the traversal of some parts of the tree.

In this work two new decomposition methods to build the tree and three new pruning rules are explored. The behaviour of our proposal is studied through experiments with synthetic and real data.

Key words: Nearest Neighbour Search, Metric Spaces, Pruning rules, Branch and Bound Scheme

1 Introduction

Nearest neighbour search (NNS) is a simple technique widely used in Pattern Recognition problems, information retrieval, data mining, etc. The NNS method consists in finding the nearest point from a prototype set to a given sample point using a dissimilarity function [5].

[1] The authors thanks the Spanish CICyT for partial support of this work through projects TIC2003–08496-C04, GV04B-541, GV04B-631, and the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

To avoid the exhaustive search, many algorithms have been developed in the last thirty years [2]. One of the most common techniques is an application of the branch and bound technique [6][7][9][11][3][10][4]. In preprocessing time, prototypes are organized in a tree structure and, in search time, the tree is traversed to find the nearest neighbour. The speed up is obtained when the exploration of some parts of the tree are avoided using efficient pruning rules. If the rules use special properties of some dissimilarity measures then its application is restricted to such dissimilarity measures.

While a number of algorithms rely on the use of the vectorial representation of the data, others are applicable in any metric space.

An example belonging to the first group is proposed by Chen *et al.* [3]. In the preprocessing stage, the proposed algorithm constructs a lower bound tree, in which each leaf node represents a sample point and each internal node represents a mean point in a space of smaller dimension. Kim and Park [9] used an ordered partition algorithm based on the ordered list of the training samples of each projection axis. This algorithm can find k-nearest neighbours in a constant expected time using a branch and bound search. A PCA-based decomposition is used by D'haes *et al.* [4] to evaluate some elimination rules and the traversing order in the tree. Also, decomposition methods based on PCA have been used in [10]. In both cases, it was shown that these algorithms have a linear space complexity, an average number of distance computations bounded by a constant term, and a time complexity very close to logarithmic for a small number of dimensions.

The second group determines the nearest neighbours in a metric space, i.e., only the distance between points are used to build the tree. The search is accelerated using the properties of the distance.

The Fukunaga and Narendra's algorithm (FN) [6] follows this pattern. The algorithm is defined to work in an Euclidean metric space. However, the modification to make it work in any metric space is trivial. In the preprocess stage an n-ary tree is obtained. Each node represents a set of prototypes, but only its mean and radius (the distance to the furthest prototype) is stored in it. The tree is built applying recursively a clustering algorithm while the set of prototypes represented by the node is greater than a threshold. The search is a depth-first traversal of the tree, combined with a best-first traversal in each node. Two pruning rules (FN pruning rules) are used in the algorithm, one for leaf nodes and the other for non-leaf nodes to avoid the full exploration of the tree.

Note that usually, the mean of a set of prototypes does not belong to the set. Based on this fact, Kamgar-Parsi and Kanal [8] introduced two new pruning rules (KK pruning rules), one for leaves an one for non-leaves, using the distance from the mean to the nearest prototype of the node.

Kalantary and McDonalds (KM) [7] proposed a similar algorithm that can be used in any metric space. Now the tree is binary and such that the leaves represent only one prototype. Similarly to the FN algorithm, each node represents a set of prototypes and stores a representative (note that in this case the mean can not be computed) and the radius of the set. The tree is built incrementally storing each new prototype in the nearest node. The algorithm uses the FN pruning rule for nodes.

The TLAESA algorithm [11] is another fast branch an bound nearest neighbour search algorithm. The algorithm is an improvement of the LAESA algorithm [12] that is an improvement of the AESA algorithm [14] as well. In this family of algorithms a lower bound of the distance from the sample to each prototype is computed and used to eliminate candidates to nearest neighbour and to find a better candidate. In the AESA and LAESA all the prototypes should be visited in order to compute a lower bound of the distance and eliminate the prototypes whose lower bound is greater than the actual nearest neighbour prototype. Then, those algorithms are only competitive if the computation of the lower bound is much faster than a distance computation. The TLAESA organizes the prototypes in a tree structure similar to the used in the KM algorithm. Now, sets of prototypes can be eliminated in a step. The search is a traversal of the tree similar to the one used in the FN and KM algorithms but using the lower bound instead of the true distances in order to avoid distance computations. In [13], Micó explored alternative ways of building the tree. Nevertheless, as the lower bound was expensive to compute, this approach, similarly to AESA and LAESA algorithms, is only competitive when the distances are very expensive.

In this work, a tree similar to the one used in the KM algorithm is used. Three methods to build binary trees are tested taking ideas of the techniques used in the FN algorithm and the TLAESA algorithm. Also three pruning rules are used in the search procedure: the FN pruning rule, a new one based on sibling node information and an iterated combination of both.

2 The search algorithm

All the proposed improvements can be used on general metric spaces. Formally, a metric space (M, d) is a set of points M with an associated distance function (also called a metric) $d : M \times M \rightarrow \mathbb{R}$ (where $\mathbb{R}$ is the set of the reals). For

function search(**tree_node** t, **sample** x)
    if not exists(left_child(t)) **then** return
    $d_\ell = d(\text{rep}(\text{left\_child}(t)), x)$; **update** nearest neighbour
    if not exists(right_child(t)) **then**
        if not pruned(left_child(t)) **then** search(left_child(t), x)
        return
    end if
    $d_r = d(\text{rep}(\text{right\_child}(t)), x)$; **update** nearest neighbour
    if $d_\ell < d_r$ **then**
        if not pruned(left_child(t)) **then** search (left_child(t), x)
        if not pruned(right_child(t)) **then** search (right_child(t), x)
    else
        if not pruned(right_child(t)) **then** search (right_child(t), x)
        if not pruned(left_child(t)) **then** search (left_child(t), x)
    end if
end function

Fig. 1. Binary branch and bound search algorithm.

all $x, y, z \in M$, this function is required to satisfy the following properties:

$$\begin{array}{ll} d(x,y) \geq 0 \ (= 0 \text{ if } x = y) & \text{(identity, when } x = y) \\ d(x,y) = d(y,x) & \text{(symmetry)} \\ d(x,z) \leq d(x,y) + d(y,z) & \text{(triangle inequality)} \end{array}$$

A binary tree structure similar to the Kalantari and McDonald's algorithm has been used to organized the prototype set. The KM algorithm is a special case of the Fukunaga and Narendras's algorithm where the number of children is fixed to 2 and the leaves contain only one prototype.

Each node (t) of the tree represents a subset (S_t) of the prototype set. Among other information needed in the pruning, each node stores a representative (rep(t)) of S_t. When a node is visited, the distance from the sample x to the representatives of both children is computed, and the closer node is the first to be explored. Each time a distance is computed, the current nearest neighbour candidate is updated if necessary. Obviously, the exploration of a node is done only if the node exists and cannot be pruned using some of the pruning rules described on the following sections (see Fig. 1 for a scheme of the algorithm).

3 The pruning rules

In order to have a base line to test the improvements obtained with the subsequent pruning rules proposal, firstly the Fukunaga and Narendra's pruning rule is described.

In our case, as a binary tree with only one prototype at the leaves is used, the special rule for leaves in the FN rules is not applicable. Then, in the following, the FN pruning rule for non leaves is referred simply as the FN pruning rule.

3.1 The Fukunaga and Narendra's pruning rule

The Fukunaga and Narendra's pruning rule [6] (FNR) is based on the following lemma.

Lemma 1.
Let (M, d) be a metric space, let $S \subset M$ be a finite subset, let $l \in M$, and let $e = \operatorname{argmax}_{s \in S} d(s, l)$. Then $\forall x, n \in M$:

$$d(e, l) \leq d(x, l) - d(x, n) \Rightarrow \forall s \in S \; d(x, s) \geq d(x, n)$$

Proof.
let $s \in S$ and $x, n \in M$,

$$\begin{aligned} d(x, s) &\geq d(x, l) - d(s, l) && \text{by the triangular inequality} \\ &\geq d(x, l) - d(e, l) && \text{as } s \in S \Rightarrow d(s, l) \leq d(e, l) \\ &\geq d(x, l) - d(x, l) + d(x, n) && \text{by hypothesis} \\ &= d(x, n) \end{aligned}$$

□

Let x be the sample point, let n be the current nearest prototype, let t be a node in the tree, and let $r = \max_{s \in S_t} d(s, \operatorname{rep}(t))$(Fig. 2). Then, if $r \leq d(x, \operatorname{rep}(t)) - d(x, n)$, by lemma 1, no prototype in S_t can be nearer to the sample that the current nearest neighbour and then the exploration of the node can be avoided (Fig. 2).

Note that the distance r does not depend on the sample, thus it can be computed and stored in preprocessing time. Note also that, when the node t is checked for pruning, the distances $d(x, n)$ and $d(x, \operatorname{rep}(t))$ have been already computed, then the FNR can be computed in constant time.

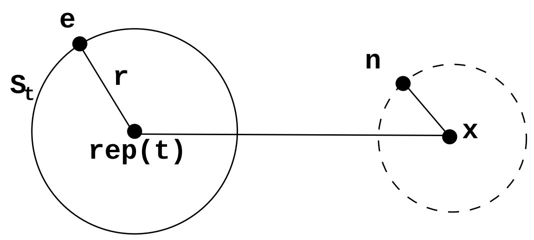


Fig. 2. Fukunaga and Narendra pruning Rule (FNR).

3.2 Sibling Based pruning Rule (SBR)

To define this rule in a node, it is necessary to know the distance between the representative of the node and the nearest prototype of the sibling node. If this distance is too big, the sibling node can be safely ignored.

The rule is based on the following lemma:

Lemma 2.
Let (M, d) be a metric space, let $S \subset M$ be a finite subset, let $r \in M$, and let $e = \text{argmin}_{s \in S} d(s, r)$. Then $\forall x, n \in M$:

$$d(e, r) \geq d(x, r) + d(x, n) \Rightarrow \forall s \in S \; d(s, x) \geq d(x, n)$$

Proof.
let $s \in S$ and $x \in M$,

$$\begin{aligned} d(s, x) &\geq d(s, r) - d(x, r) && \text{by the triangular inequality} \\ &\geq d(e, r) - d(x, r) && \text{as } s \in S \Rightarrow d(s, r) \leq d(e, r) \\ &\geq d(x, r) + d(x, n) - d(x, r) && \text{by hypothesis} \\ &= d(x, n) \end{aligned}$$

□

Let x be the sample, let n be the current nearest prototype, let t be a node in the tree, let t_s be its sibling node, and let $r = \min_{s \in S_{t_s}} d(s, \text{rep}(t))$(fig. 3). Then if $r \geq d(x, \text{rep}(t)) + d(x, n)$, by lemma 2, no prototype in S_{t_s} can be nearer to the sample that the current nearest neighbour and then the exploration of the node can be avoided.

Note that the distance r does not depend on the sample, it can then be computed and stored in preprocessing time. Note also that, when the node t is checked for pruning, the distance $d(x, n)$ is already computed, then the SBR can be computed in constant time (as de FNR rule).

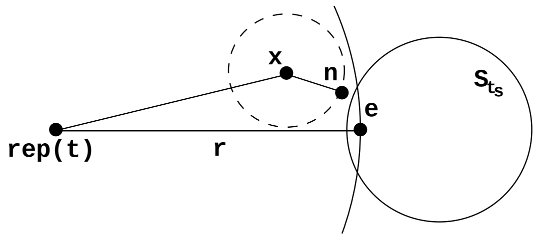


Fig. 3. Sibling based rule (SBR).

Moreover, if SBR is evaluated before the computation of the distance $d(x, \mathrm{rep}(t_s))$, in case of pruning, this distance computation can be avoided.

Kamgar-Parsi and Kanal [8] proposed, for the FN algorithm, a similar rule (KKR), but the distance from the mean (used as representative) to the nearest prototype in the node was used. It is important to note that in our case the representative is always a prototype belonging to the node, then this distance is always zero and then, the KKR can not be applied. Moreover, the new proposed rule allows the pruning of the sibling node but the KKR only can prune the own node.

3.3 Generalized pruning Rule (GR)

This rule is an iterated combination of the FNR and the SBR. The rule is based on the following lemma:

Lemma 3.
Let (M, d) be a metric space, let $S \subset M$ be some finite subset, let $l, r \in M$, let $e_i \in S$, and $G_i \subset S$ such that:

$$\begin{aligned} G_1 &= S \\ e_i &= \mathrm{argmax}_{g \in G_i} d(g, l) \\ G_{i+1} &= \{g \in G_i : d(g, r) < d(e_i, r)\} \end{aligned}$$

Then $\forall x, n \in M$:

$$\begin{aligned} &\exists i : d(r, e_i) \geq d(r, x) + d(x, n) \wedge \\ &\quad d(l, e_{i+1}) \geq d(l, x) - d(x, n) \Rightarrow \forall s \in S \quad d(s, x) \geq d(x, n) \end{aligned}$$

Proof.
Let $s \in S$, there are two cases,

(1) Let $s \in G_{i+1}$, then $e_{i+1} = \text{argmax}_{g \in G_{i+1}} d(g,l)$. Let $x \in M$,

$$\begin{aligned} d(x,s) &\geq d(x,l) - d(s,l) && \text{by the triangular inequality} \\ &\geq d(x,l) - d(e_{i+1},l) && \text{as } s \in S \Rightarrow d(s,l) \leq d(e_{i+1},l) \\ &\geq d(x,l) - d(x,l) + d(x,n) && \text{by hypothesis} \\ &= d(x,n) \end{aligned}$$

(2) Let $s \notin G_{i+1}$, then as $G_{i+1} = \{g \in G_i : d(g,r) < d(e_i,r)\}$ we have that $d(s,r) \leq d(e_i,r)$. Let $x \in M$,

$$\begin{aligned} d(s,x) &\geq d(s,r) - d(x,r) && \text{by the triangular inequality} \\ &\geq d(e_i,r) - d(x,r) && \text{as } s \notin G_{i+1} \Rightarrow d(s,r) \leq d(e_i,r) \\ &\geq d(x,r) + d(x,n) - d(x,r) && \text{by hypothesis} \\ &= d(x,n) \end{aligned}$$

□

Let x be the sample, let n be the current nearest prototype, let t be a node in the tree, t_s be the sibling node of t, $e_i \in S_t$, and $G_i \subset S_t$ such that:

$$\begin{aligned} G_1 &= S_t \\ e_i &= \text{argmax}_{g \in G_i} d(g, \text{rep}(t)) \\ G_{i+1} &= \{g \in G_i : d(g, \text{rep}(t_s)) < d(e_i, \text{rep}(t_s))\} \end{aligned}$$

and let m be the smaller natural number such that $|G_m| = 0$.

Then if it exists i such that:

$$d(\text{rep}(t_s), e_i) \geq d(\text{rep}(t_s), x) + d(x,n) \wedge \quad (1)$$
$$d(\text{rep}(t), e_{i+1}) \geq d(\text{rep}(t), x) - d(x,n) \quad (2)$$

no prototype in S_t can be nearer to the sample than the current nearest neighbour, and then the exploration of the node t can be avoided. Note that the computation of the distances $d(\text{rep}(t_s), e_i)$ and $d(\text{rep}(t), e_{i+1})$ does not depend on the sample, those distances can then be computed and stored on preprocessing time (fig. 4).

Note also that, when the node t is checked for pruning, the distances $d(x,n)$, $d(\text{rep}(t), x)$ and $d(\text{rep}(t_s), x)$ are already computed. Then, for each i, the test can be computed in constant time. In order to find if there is a i fulfilling the conditions, a binary search can be used.

Note also that the rules FNR and SBR are special cases of GR in the two following cases:

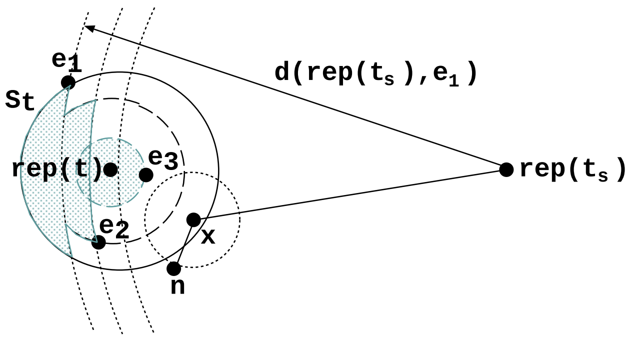


Fig. 4. Generalized rule (GR).

- if $i = 0$ and only equation 2 is considered, then we obtain FNR
- if $i = m$ and only equation 1 is considered, then we obtain SBR.

4 The tree construction

As said in a previous section, a binary tree with one prototype at the leaves is used by the algorithm. In this section three different constructions of the tree are described.

In order to have a base line to compare the other proposals a technique similar to the used in the FN algorithm is going to be described.

4.1 c-means tree

Fukunaga and Narendra [6] proposed the construction of the tree applying recursively a clustering algorithm, the *c*-means algorithm [5]. The centroid provided by the clustering algorithm was used as representative of each node instead of the mean.

Figure 5(a) shows the partition induced in a set of 1000 2-dimensional points in the first stages of the algorithm.

4.2 Most Scattered Points tree

The use of the centroid as the representative of a set of prototypes, is a common choice in clustering techniques. However, it is not a good idea if the FNR is used to prune. With regard to this rule, each node can be considered as a ball centered in the representative and with radius the distance between the representative and the furthest prototype of the node. Then, to maximize the chances to prune a node, a good choice is to minimize the overlapping regions of a node with respect to its sibling node. This can be done by choosing both representatives as scattered as possible. This scheme is followed by the *Most Scattered Points* (MSP) strategy.

Given a node t:

- use as representatives of the two children of node t the two most scattered prototypes in the node;
- classify the rest of the prototypes in the node of the nearest representative;
- recursively repeat the previous steps until each final node has only a prototype, the representative.

In the worst case, when the tree is degenerated, the time complexity is $O(n^3)$, where n is the number of points. But in a typical case, the tree is balanced, and then the cost is $O(n^2)$.

Figure 5(b) shows the partition induced in a set of 1000 2-dimensional points in the first stages of the algorithm.

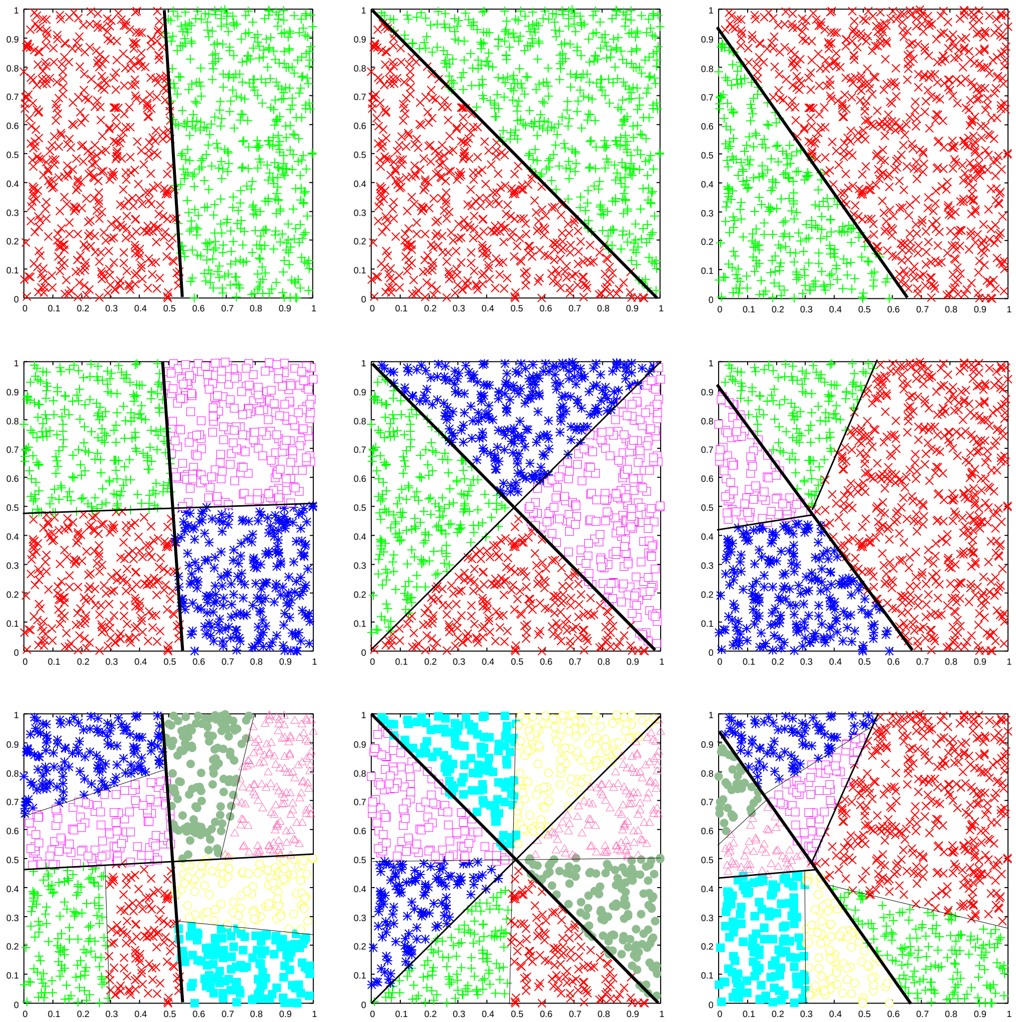
4.3 Most Distant from the Father tree

This approach is based on a similar technique used in the TLAESA algorithm [11]. In the search, each time an internal node is explored, the distance to the representatives of the children is computed. In the *Most Distant from the Father* (MDF) technique one of the distance computations is avoided using as representative of the left child the representative of its father. Following the ideas of the MSP technique, the right representative is the most distant point of the left child representative. Obviously, in this case, the tree is not as balanced as using the MSP technique, but the saving on distance computations in the search procedure compensates this drawback.

Given a node t, let us call the children nodes t_r, t_ℓ:

- select $\mathrm{rep}(t)$ as the representative of t_ℓ;
- select $\mathrm{argmax}_{s\in S_t} d(t_\ell, s)$ as representative of t_r;
- classify the rest of the prototypes in the node of the nearest representative;
- recursively repeat the process until each leaf node has only one point, the representative.

In the worst case, when the tree is degenerated, the time complexity is $O(n^2)$, where n is the number of points. But in a typical case, the tree is balanced, and then the complexity is $O(n \log(n))$.



(a) (b) (c)

Fig. 5. Left column shows the way in which 1000 points of a 2-dimension uniform distribution are divided in the first stages of the building of the tree following the c-means strategy. The middle column shows the behaviour when MSP is used and the right column when MDF strategy is used.

Figure 5(c) shows the partition induced in a set of 1000 2-dimensional points in the firsts stages of the algorithm.

5 Experiments

Some experiments with synthetic and real data were carried out to study the behaviour of our proposals. Two different distances were used in these experiments: Euclidean distance and string edit distance. The experiments were developed by combining the proposed decomposition methods and pruning

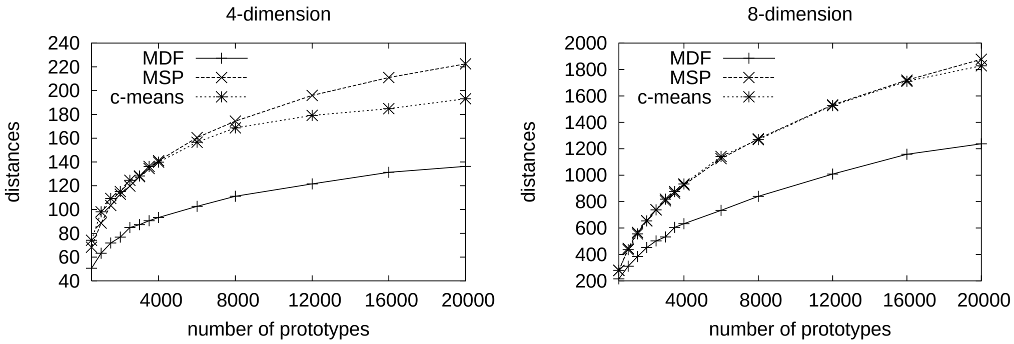


Fig. 6. Behaviour of the algorithm when c-means, MSP and MDF techniques are used in the tree building process using FNR as pruning rule.

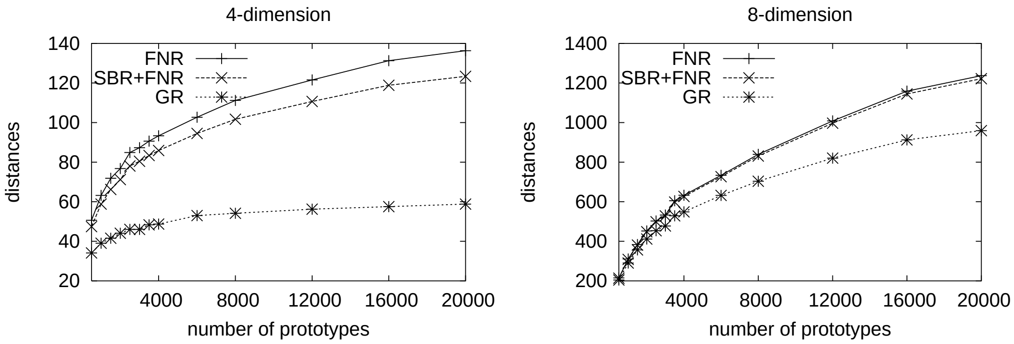


Fig. 7. Behaviour of the algorithm when MDF is used to build the tree and the pruning rules FNR, FNR+SPR and GR are used in the search.

rules. Different sizes of the training set were used and 1000 samples were used as test set in all the experiments both with synthetic data and with real data.

5.1 Experiments with Euclidean distance

In the synthetic experiments, prototype sets were produced from uniform distributions in the unit hypercube. Experiments for a range of dimensions were tested. For clarity only results for dimensions 4 and 8 are showed in this section. Each plot in the graphics shows the average of 10 experiments, using 1000 samples as test in each of them.

A first set of experiments were carried out in order to study the behaviour of the algorithm using c-means (the proposed by Fukunaga and Narendra), and the MSP and MDF techniques to build the tree for increasing sizes of the training set (fig. 6). In all these experiments the FNR pruning rule was used. The experiments show that MDF is clearly superior due to the saving of one distance computation at each level, compensating the fact that the trees are deeper. It can also be observed that MSP also improves the c-means technique, although its behaviour degenerates quickly with the dimension.

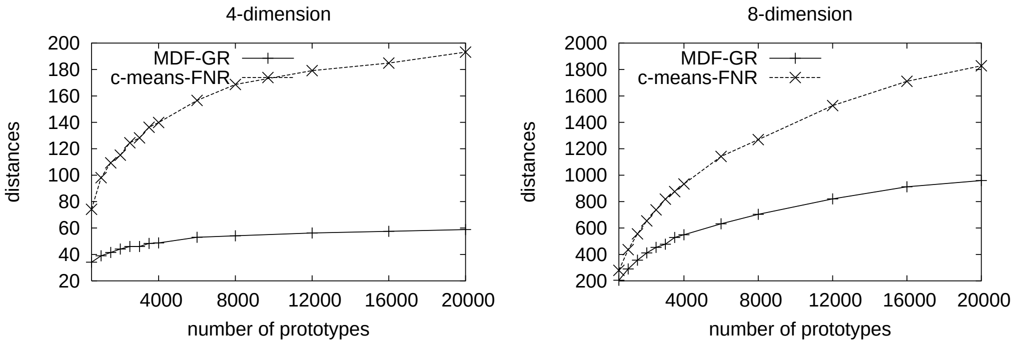


Fig. 8. Improvement of using the combination of MDF technique to build the tree and GR in the searching process in relation to the use of c-means to build the tree and FNR in the searching process.

A second set of experiments were carried out to show the behaviour of the algorithm for FNR, SBR+FNR and GR pruning rules (fig. 7). All the experiments were done using the MDF technique in the tree building process. The experiments show that the addition of the SBR to the FNR reduces the distance computations, although degenerates quickly when the dimension grows. It can also be observed that GR pruning rule reduces drastically the distance computations with respect to the other rules. It seems that this performance degenerates also with the dimension, but no so quickly.

In order to illustrate the improvement of our proposal, figure 8 shows the a-verage number of distance computations using the classical approach (c-means tree building strategy and FNR pruning rule), and the best of our techniques (combining MDF tree building strategy and GR pruning rule). In both 4-dimension and 8-dimension it can be seen a considerable saving on distance computations. We can observe, that in 8-dimension and using 20000 prototypes as training set, a saving near a 50% distance computations is possible.

The previous experiments were repeated using real data, the PHONEME database from the ROARS ESPRIT project [1]. The PHONEME database consists of 5404 5-dimension vectors from 2 classes. The set was divided in 5 subsets, using 4 sets as prototypes and 1 set as samples. A leaving one out technique was used.

Figure 9 shows the distance computations when the size of the training set increases for the three tree building strategies and the FNR pruning rule. Figure 10 shows the effect of the pruning rules on the computations using MDF technique in the building tree process.

Finally, figure 11 shows the distance computations using the classical approach and the best of our techniques. As the results show a saving of near a 75% distance computations is obtained using around 4500 prototypes as training

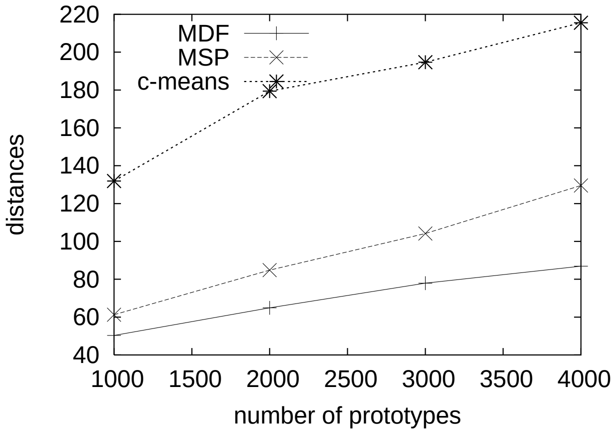


Fig. 9. Average number of distance computations by sample in relation to the size of the training set for the PHONEME database when c-means, MSP and MDF are used to build the tree and FNR pruning rule is used in the search.

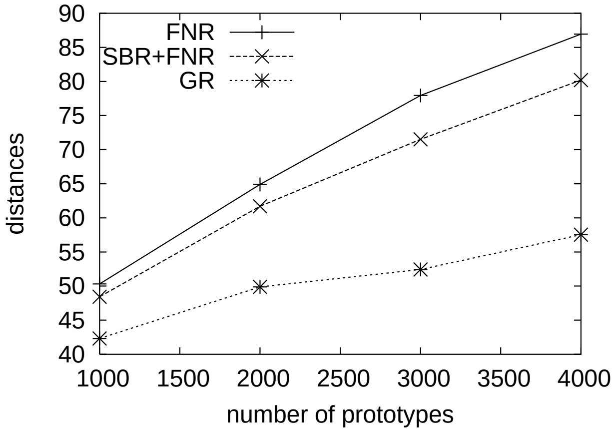


Fig. 10. Average number of distance computations by sample in relation to the size of the training set for the PHONEME database when MDF technique is used in the tree building process and FNR, SBR and GR pruning rules are used in the search.

set.

5.2 Experiments with edit distance

To test our proposals with other type of distances, similar experiments using edit distance were carried out. Edit distance [15] between two strings is the minimum cost sequence of character insertions, deletions and substitutions to make equal the two strings. In our experiments the values of the weighting parameters were fixed to 1.

A database of 51,589 words of the Spanish dictionary were used. Words containing upper and lower letters, accentuated vowels and diaeresis were included. The experiment consists of a simulation of a real spelling task. The

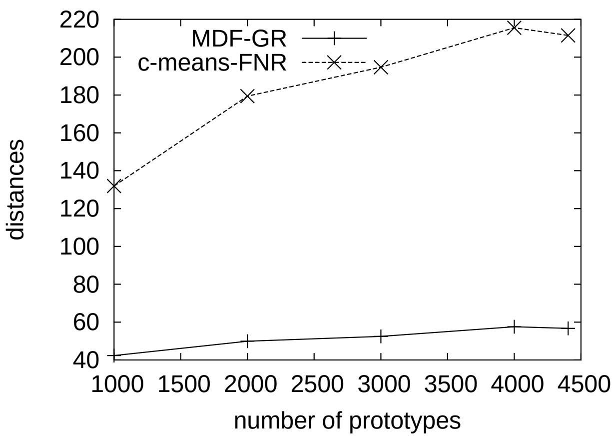


Fig. 11. Improvement of using the combination of MDF technique to build the tree and GR in the searching process in relation to the use of c-means to build the tree and FNR in the search.

input test of the speller was simulated distorting these 51 589 words. To obtain the distorted dictionary, insertion, deletion and substitution operations were applied to the words in the original dictionary.

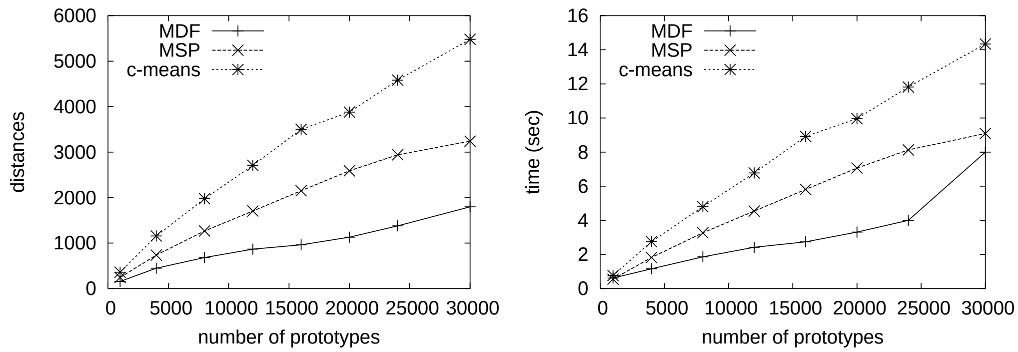


Fig. 12. Average number of distance computations by sample and average search time in relation to the size of the training set for the different constructions of the search tree.

Increasing sizes of the set of prototypes were used in the training. The test size consists of 1000 randomly selected words of the distorted dictionary.

In order to obtain reliable results, 10 different experiments were carried out for each size of the training set, all the figures show the averages.

Figure 12 illustrates the distance computations and the search time by test sample using c-means, MSP and MDF as tree building techniques and FNR as pruning rule. As it was expected, the MDF method reduces the search time.

A second set of experiments were carried out in order to study the behaviour of the algorithm with the FNR, FNR+SBR and GR pruning rules using the MDF tree building technique (see fig 13). As was expected the addition of the SBR reduces slightly the number of distance computations but GR reduces it

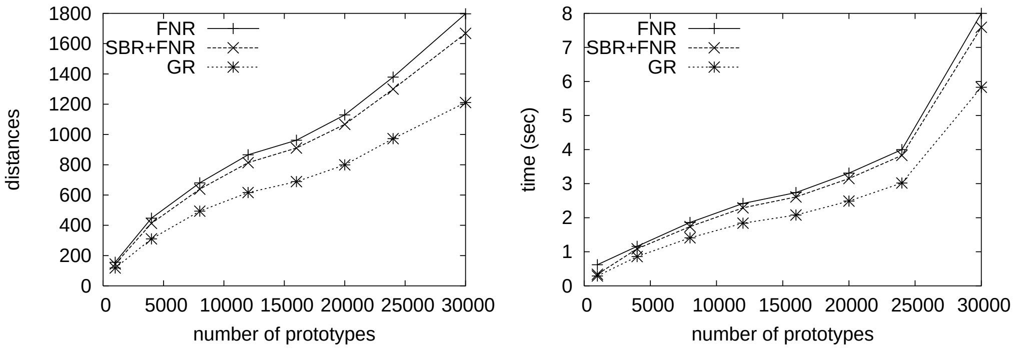


Fig. 13. Average number of distance computations by sample and computation time in relation to the size of the training set when MDF technique is used in the tree building process and FNR, SBR and GR are used in the search.

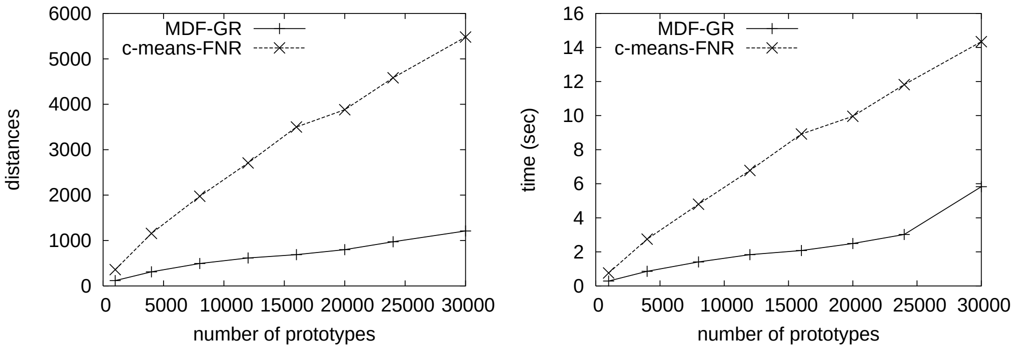


Fig. 14. Improvement of using the combination of MDF technique to build the tree and GR in the searching process in relation to the use of c-means to build the tree and FNR in the searching process.

noticeably. The same behaviour is observed with the search time.

Finally, to illustrate the improvement of our proposal, figure 14 shows the distance computations and the computation time using the classical approach and the best of our techniques.

Note that for 30,000 prototypes as training set, a saving of more than a 50% search time is obtained using the combination of MDF tree building strategy and GR pruning rule instead of the classical approach.

Obviously, lower search time can be obtained with tailored methods for the spelling task. It should be noted that this algorithm is completely general and then it can work in any metric space.

6 Conclusions

In this paper we have developed a series of improvements based on a branch and bound search scheme.

On the one hand, two simple new methods to build the tree has been proposed. These techniques allows the search of the nearest neighbour with less distance computations.

On the other hand, two new elimination rules are proposed to speed up the nearest neighbour search.

Both techniques can be combined. The experiments suggest that high speeds ups can be obtained.

In the future, we plan to apply these approximations to other nearest neighbour search algorithms based on a tree structure. We are also interested in testing these techniques in general metric spaces studying the behaviour of these techniques in relation to the dimensionality of the data.

References

[1] Alinat, P.: Periodic progress report 4, ROARS project ESPRIT II - Number 5516. Thomson Technical Report TS ASM 93/S/EGS/NC/079 (1993)

[2] Belur V. Dasarathy: Nearest Neighbour(NN) Norms: NN Pattern Classification Techniques. IEEE Computer Society Press (1991)

[3] Chen, Y.S., Hung, Y.P., Fuh, C.S.: Fast Algorithm for Nearest Neighbour Search Based on a Lower Bound Tree. Proceedings of the 8th International Conference on Computer Vision, Vancouver, Canada, (2001), **1** 446–453

[4] D'haes, W., Dyck, D., Rodet, X.: PCA-based branch and bound search algorithms for computing K nearest neighbors. Pattern Recognition Letters (2002) **24** 1437–1451

[5] Duda, R., Hart, P., Stork, D.: Pattern Classification. Wiley Interscience (2001)

[6] Fukunaga, K., Narendra, M.: A branch and bound algorithm for computing k–nearest neighbours. IEEE Trans. Computing (1975) **24** 750–753

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

[8] Kamgar-Parsi, B., Kanal, L.: An improved branch and bound algorithm for computing k-nearest neighbors. Pattern Recognition Letters (1985) **3** 7–12

[9] Kim, B.S., Park, S.B.: A Fast K Nearest Neighbor Finding Algorithm Based on the Ordered Partition. Pattern Analysis and Machine Intelligence (1986), **6** 761–766

[10] McNames, J.: A fast nearest neighbor algorithm based on a principal axis search tree. Pattern Analysis and Machine Intelligence (2001) **9**, 964–976

[11] Micó, L., Oncina, J., Carrasco, R.C.: A fast branch and bound nearest neighbour classifier in metric spaces. Pattern Recognition Letters (1996) **17** 731–739

[12] Micó, L., Oncina, J., Vidal, E.: A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing-time and memory requirements. Pattern Recognition Letters (1994) **15**, 9–17

[13] Micó, L.: Algoritmos de búsqueda de vecinos más próximos en espacios métricos. Tesis doctoral (1996)

[14] Vidal, E.: New formulation and improvements of the Nearest-Neigbour Approximating and Eliminating Search Algorithm (AESA). Pattern Recognition Letters (1994) **15**, 1–7

[15] Wagner, R.A., Fisher, M.J.: The string-to-string correction problem. J. ACM (1974) **21** (1), 168–173