HAL open science

Combining Elimination Rules in Tree-Based Nearest Neighbor Search Algorithms

Eva Gómez-Ballester, Luisa Micó, Franck Thollard, Jose Oncina, Francisco Moreno-Seco

To cite this version:

Eva Gómez-Ballester, Luisa Micó, Franck Thollard, Jose Oncina, Francisco Moreno-Seco. Combining Elimination Rules in Tree-Based Nearest Neighbor Search Algorithms. Structural, Syntactic, and Statistical Pattern Recognition, Aug 2010, Cesme, Izmir, Turkey. pp.80-89. hal-00961322v2

HAL Id: hal-00961322

https://ujm.hal.science/hal-00961322v2

Submitted on 10 Sep 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Combining Elimination Rules in Tree-Based Nearest Neighbor Search Algorithms

Eva Gómez-Ballester[1], Luisa Micó[1], Franck Thollard[2], Jose Oncina[1], and Francisco Moreno-Seco[1]

[1] Dept. Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03071 Alicante, Spain
{eva,mico,oncina,paco}@dlsi.ua.es
[2] Grenoble University, LIG
BP 53, 38041 Grenoble Cedex 9
thollard@univ-st-etienne.fr

Abstract. A common activity in many pattern recognition tasks, image processing or clustering techniques involves searching a labeled data set looking for the nearest point to a given unlabelled sample. To reduce the computational overhead when the naive exhaustive search is applied, some fast nearest neighbor search (NNS) algorithms have appeared in the last years. Depending on the structure used to store the training set (usually a tree), different strategies to speed up the search have been defined. In this paper, a new algorithm based on the combination of different pruning rules is proposed. An experimental evaluation and comparison of its behavior with respect to other techniques has been performed, using both real and artificial data.

1 Introduction

Nearest Neighbor Search (NNS) is an important technique in a variety of applications including pattern recognition [6], vision [13], or data mining [1, 5]. These techniques aim at finding the object of a set nearest to a given test object, using a distance function [6]. The use of a simple brute-force method is sometimes a bottleneck due to the large number of distances that should be computed and/or their computational effort. In this work we have considered the computational problem of finding nearest neighbors in general metric spaces. Spaces that may not be conveniently embedded or approximated in an Euclidean space are of particular interest. Many techniques have been proposed for using different types of structures (vp-tree [16], GNAT [3], sa-tree [10], AESA [14], M-tree [4]): the tree-based techniques are nevertheless more popular. The Fukunaga and Narendra algorithm (FNA [7]) is one of the first known tree-based example of this type of techniques. It prunes the traversal of the tree by taking advantage, as the aforementioned methods, of the triangular inequality of the distance between the prototypes. This sets up a general framework for designing and evaluating new pruning rules, as stated in [9].

In this paper we study the combination of different pruning rules: recent table rule [12], a rule that is based on information stored in the sibling node (the sibling rule [9]), the original rule from the FNA (Fukunaga and Narendra rule, FNR), and a generalization of both the sibling rule and the FNR one [9]. We end up with a new algorithm for combining the rules that significantly reduces the number of distance computations.

The algorithm is evaluated on both artificial and real world data and compared with state-of-the-art methods.

The paper is organized as follows: we will first recall the FNA algorithm and define the general framework of the new algorithm (in particular how the tree is built). We then review the different rules we aim at combining (section 3). We then propose our new algorithm (section 4). Sections 5 presents the experimental comparison.

2 The basic algorithm

The FNA is a fast tree-based search method that can work in general metric spaces. In the original FNA the c-means algorithm was used to define the partition of the data. In the work by Gómez-Ballester et al [8] many strategies were explored: the best one, namely the *Most Distant from the Father tree* (MDF), in which the representative of the left node is the same as the representative of its father, is the strategy used in the experiments presented in this work. Thus, each time when an expansion of the node is necessary, only one new distance needs to be computed (instead of two), hence reducing the number of distances computed. This strategy was also successfully used by Noltomeier et al [11] in the context of bisector trees.

In the MDF tree each leaf stores a point of the search space. The information stored in each node t is S_t, the set of points stored in the leaves of t sub-tree, M_t (the representative of S_t) and the radius of S_t, $R_t = \text{argmax}_{x \in S_t} d(M_t, x)$. Figure 1 shows a partition of the data in a 2-dimensional unit hypercube. The root node will be associated with all the points of the set. The left node will represent all the points that belong to the hyperplane under the segment [(0, 0.95) ; (0.65,0)]; the right node will be associated with the other points. According to the MDF strategy, the representative of the right node (M_r) is the same as the father, and the representative of the left node (M_ℓ) is the most distant point to M_r. The space is then recursively partitioned.

3 A review of pruning rules

Fukunaga and Narendra Rule (FNR)

The pruning rule defined by Fukunaga and Narendra for internal nodes makes use of the information in the node t to be pruned (with representative M_t and radius R_t) and the hyperspherical surface centered in the sample point x with radius $d(x, nn)$, where nn is current nearest prototype. To apply this rule it is

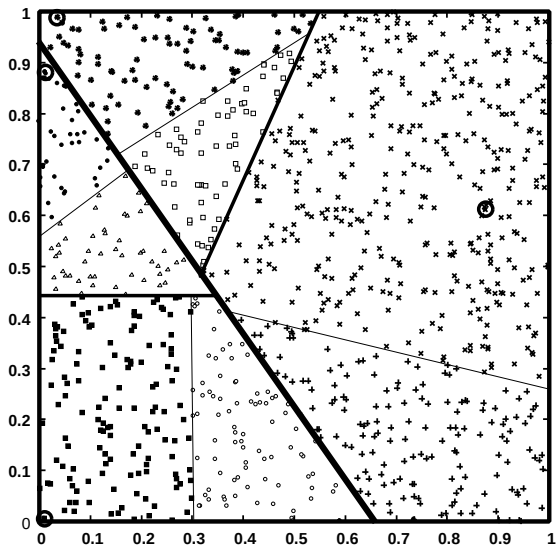


Fig. 1: Partition of the data using the MDF strategy. Representatives of each node in different levels are drawn as rings.

necesary to compute the distance from the test sample to the representative of candidate node that aim to be eliminated. Figure 2a presents a graphical view of the Fukunaga and Narendra rule.

Rule: No $y \in S_t$ can be the nearest neighbor to x if $d(x, nn) + R_t < d(x, M_t)$

The Sibling Based Rule (SBR)

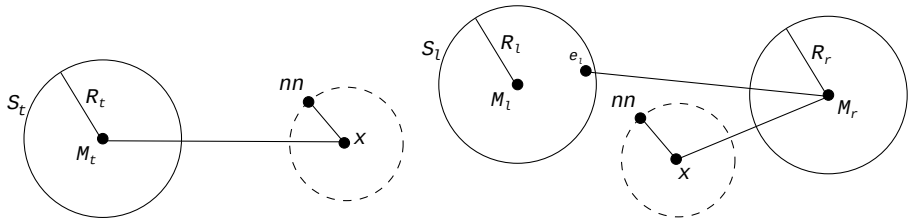
Given two sibling nodes r and ℓ, this rule requires that each node r stores the distance $d(M_r, e_\ell)$, that is the distance between the representative of the node, M_r, and the nearest point, e_ℓ, in the sibling node ℓ (S_ℓ). Figure 2b presents a graphical view of the Sibling based rule.

Rule: No $y \in S_\ell$ can be the nearest neighbor to x if $d(M_r, x) + d(x, nn) < d(M_r, e_\ell)$.

Unlike the FNR, SBR can be applied to eliminate node ℓ without computing $d(M_\ell, x)$, avoiding some extra distance computations at search time.

Generalized rule (GR)

This rule is an iterated combination of the FNR and the SBR (due to space constraints we refer the reader to [9] for details on the generalized rule). In GR, the distance to the representative of a given node is needed to know if the node can be pruned or not.



(a) Geometrical view of FNR rule. (b) Geometrical view of SBR rule.

The table rule (TR)

This recent rule [12] prunes the tree by taking the current nearest neighbor as a reference. In order to do so, a new distance should be defined:

Definition. Given a prototype or sample point p, the distance between p to a set of prototypes S is defined as

$$d(p, S) = \min_{y \in S} d(p, y)$$

At pre-process time, the distances from each prototype to each prototype set of each node t, S_t, in the tree are computed and stored in a table, allowing a constant time pruning. Note that the size of this table is quadratic in the number of prototypes since, as the tree is binary, the number of nodes is two times the number of prototypes.

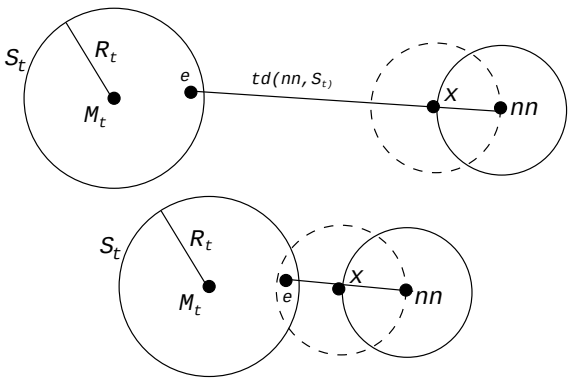


Fig. 2: Table rule and node S_t: situation where it can be pruned (up) and where it cannot (down)

Rule: No $y \in S_t$ can be the nearest neighbor to x if $2d(nn, x) < d(nn, S_t)$.

Figure 2 presents a graphical view of the table rule. Note that this rule can be used before computing the distance to the node that will be explored.

4 CPR: Combining Pruning Rules algorithm

In Algorithm 1 an efficient combination of pruning rules is proposed. Note that, as the GR generalizes both the FNR and the SBR, these two rules are not applied while the generalized one is activated (lines 11-19). When the MDF method is used to build the tree, it is important to note that each time a node is expanded, only one of the representatives is new (the left node), while the other (right) is the same as the father node (in this case, only the radius of the node can change). For this reason, in this case the distance $d_r = d(x, M_r)$ in line 9 is never computed (as it is already known). Then, when a node is examined during the search, every pruning that can be applied without computing a new distance is applied (lines 3 to 8). If none of these rules is able to prune, the distance to the current node is computed (line 9). The pruning rules that use the new distance are then applied (lines 11 to 28).

5 Experiments

We have performed some experiments in order to compare our algorithm with some state of the art methods. The first method, the multi-vantage-point tree (*mvp*), is a balanced tree requiring linear space where the arity can be extended and multiple pivots per node can be applied [2]. The second method is the Spatial Approximation Tree (*sat*), whose structure uses a graph based on Delaunay triangulation and it does not depend on any parameter [10]. The code of these algorithms comes from the SISAP library (`www.sisap.org`). We applied the *mvp* with only one pivot by node, a bucket size of 1 and an arity of 2 as this setting leads to better performances according to preliminary experiments on these data sets. All the experiments were performed on a Linux box with 16GB of memory.

From now and only for the graphs, the FNR rule (and respectively the SBR, GR and TR rules) will be abbreviated by "f" (respectively "s", "g" and "t"); consequently, combining the FBR and SBR will be referred as "fs". The combinations of rule "g" with "s" or "f" are not present as "g" generalizes these rules: every branch pruned by one of them is also pruned by "g".

In order to evaluate the performance of different combined rules, we present in this section the experiments on both artificial and real world data using different settings of our algorithm.

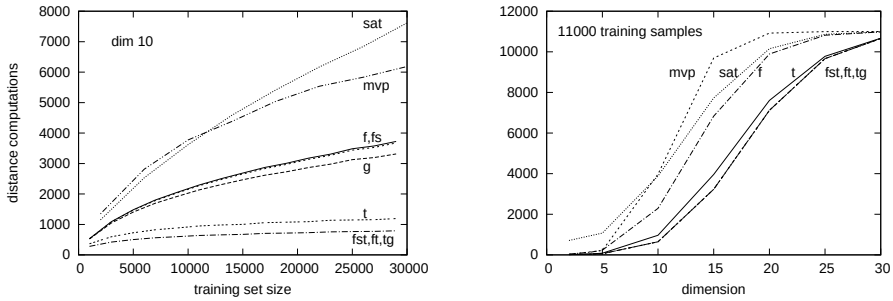
5.1 Artificial data with uniform distributions

We consider here points drawn in a space of dimension n ranging from 5 to 30. The algorithms are compared with a growing number of prototypes. The size of the prototype sets ranged from $2,000$ prototypes to $30,000$ in steps of $4,000$. Each experiment measures the average distance computations of $10,000$ searches ($1,000$ searches over 10 different prototype sets). The samples are drawn from the same distribution.

Algorithm 1: **CPR(t,x)**

```
Data: t: a node tree; x: a sample point;
Result: nn: the nearest neighbor prototype; d_min: the distance to nn;
if t is not a leaf then
    r = right_child(t); ℓ = left_child(t);
    if ( SBR(ℓ) || TR(ℓ) ) then
        if (no FNR(r)) && (no TR(r)) then
            CPR(r, x) /* left (sibling) node has been pruned */;
        end
        Return /* ie prune both */ ;
    end
    d_r = d(x, M_r) ;        d_ℓ = d(x, M_ℓ);
    update d_min and nn;
    if Activated(GR) then
        if d_ℓ < d_r then
            if ( no GR(ℓ) ) then CPR(ℓ, x);
            if ( no GR(r) ) then CPR(r, x);
        else
            if (no GR(r)) then CPR(r, x);
            if (no GR(ℓ)) then CPR(ℓ, x);
        end
    else
        if d_ℓ < d_r then
            if (no FNR(ℓ)) && (no SBR(ℓ)) then CPR(ℓ, x);
            if (no FNR(r)) && (no SBR(r)) then CPR(r, x);
        else
            if (no FNR(r)) && (no SBR(r)) then CPR(r, x);
            if (no FNR(ℓ)) && (no SBR(ℓ)) then CPR(ℓ, x);
        end
    end
end
```

Figure 3a shows the average number of distance computations in a 10-dimensional space following a uniform distribution. Standard deviation of measures is not included as it is almost negligible. As it can be seen, both *sat* and *mvp* are outperformed by the other pruning rules. Although the table rule also outperforms the FNR and GR ones, it is worth mentioning that these methods have a space consumption smaller than the table rule. In the case of small space capabilities, these methods should be preferred. Considering the classic FNA algorithm as a reference, we observe that GR and TR rules outperform the original rule, namely FNR. Moreover, it appears that combining the table rule, with either the sibling or generalized rule, does not perform better than combining the FNR and the table rule. This is important as the FNR rule has an effective computational cost smaller than the generalized rule. Furthermore, since the "g" rule also gen-



(a) Distance computations w.r.t. training set size in a 10-dimensional space.

(b) Distance computations w.r.t dimensionality.

Fig. 3: Comparison of different pruning rules combinations with *sat* and *mvp* algorithms

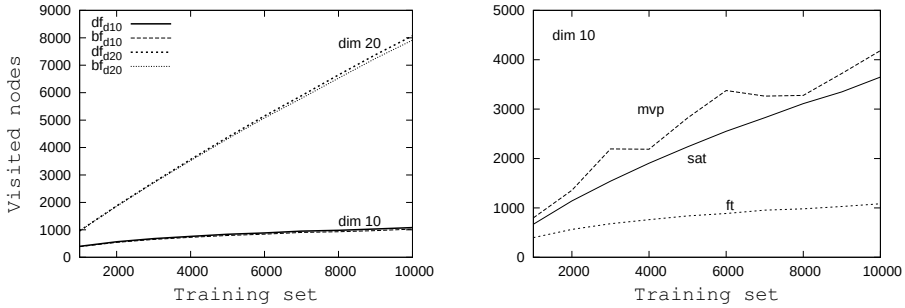
eralizes the sibling rule, the combination of "fst" does not perform better than "fg", as expected.

Another classic problem to address is *the curse of dimensionality*[3]. It expresses the fact that the volume of the unit hypercube increases exponentially with the dimension of the space. In other words, the points tend to be at the same distance one to each other in great dimensions. In our setting, this will obviously prevent a large number of prunings: the algorithm will tend to behave like the brute force algorithm as the dimension increases. This algorithmic limitation is not a real problem since looking for a nearest neighbor does not make sense in a space where the distances between each pair of points are similar.

Figure 3b addresses a comparative analysis of the behavior of the methods as the dimension increases. The number of prototype is set to $11,000$ points and the dimensionality ranges from 2 to 30. It can be observed here that the TR rule is less sensible to the dimensionality than the other methods. Moreover, as before, combining the TR rule with the FNR one still performs better than the other combinations: at dimension 25, the "ft" combination is able to save 20% of distance computations while the other methods compute all the distances, as the exhaustive search.

Two more experiments were performed: first, in order to show the differences when a best-first strategy is used instead of a depth-first strategy. In Figure 4a one can see that similar results are obtained, for this reason, only depth-first strategy is used in this work. Second, as well as the distance computations, the percentage of the database examined is analyzed for all the methods. Results can be seen in Figure 4b. As in the case of distance computations, the CPR method also reduces the overhead of the search visiting on average less nodes (or points in the data set).

[3] The *curse of dimensionality* is usually considered in Euclidean spaces.



(a) Best-first (bf) versus depth-first (df) strategies in 10 and 20 dimensional spaces.

(b) Visited nodes w.r.t. training set size.

Fig. 4: Average number of visited nodes during the search for the best pruning rule combination, different search strategies and *sat* and *mvp* algorithms.

5.2 Real world data

To show the performance of the algorithms with real data, some tests were conducted on a spelling task. For these experiments, a database of 69,069 words of an English dictionary was used [4]. The input test of the speller was simulated distorting the words by means of random insertion, deletion and substitution operations over the words in the original dictionary. The Levenshtein distance [15] was used to compare the words. Dictionaries of increasing size (from 2,000 to 30,000) were obtained by extracting randomly words of the whole dictionary. Test points were obtained distorting the words in the training set. For each experiment, 1000 distorted words were generated and used as test set. To obtain reliable results, the experiments were repeated 10 times. The averages are showed on the plots.

The experiment performed in Figure 3a for artificial data (average number of distance computations using increasing size prototype sets) was repeated in the spelling task. Results are shown in Figure 5. The experiments show a reduction in the number of distance computations around 20% when the SBR rule is combined with the FNR, and around 40% for generalized rule with respect to the reference FNR rule. Moreover, when combining both the "f" and "t" rules (with or without the "g" rule), the resulting combination clearly outperforms the other combinations, as it happens with other kinds of data, saving 60% of the average number of distance computations.

6 Conclusions and further works

A new algorithm has been defined to optimize the combination of several pruning rules using the FNA tree-based search algorithm. When the rules are applied

[4] here again the databases are taken from the SISAP repository

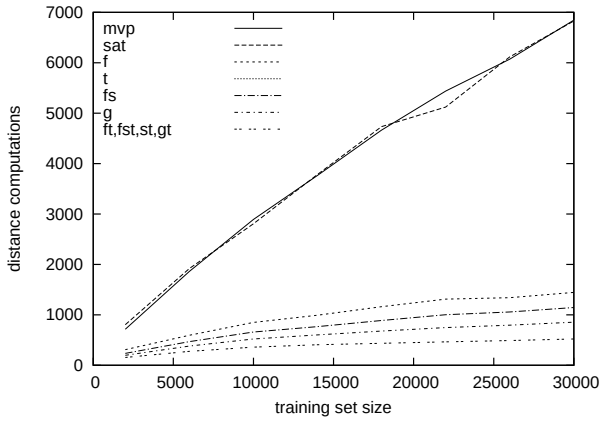


Fig. 5: Pruning rules combined in a spelling task in relation to others methods.

alone, reductions between 20% and 60% are obtained for low dimensions and this reduction decreases with the dimensionality (a normal behavior since the problem is getting harder with increasing dimensionalities) when comparing with the baseline FNR rule. When the rules are combined, more reductions in the average number of distance computations and in the overhead of the methods (measured as the average number of visited nodes or points), in particular can be observed (*e.g.* roughly 80% reduction in a 10-dimensional space). Similar results are also obtained on a real world task (namely a spelling task).

We are currently studying new pruning rules and combinations, and also how to use them in dynamic tree structures. We think also that this algorithm can be adapted with minor changes to other tree-based search methods not explored in this work.

7 Acknowledgments

The authors thank the Spanish CICyT for partial support of this work through projects TIN2009-14205-C04-C1 and TIN2009-14247-C02-02, 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. Christian Böhm and Florian Krebs. High performance data mining using the nearest neighbor join. In *ICDM'02: Proceedings of the 2002 IEEE International Conference on Data Mining*. IEEE Computer Society, 2002.
2. T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international*

conference on Management of data, pages 357–368, New York, NY, USA, 1997. ACM.

3. S. Brin. Near neighbor search in large metric spaces. In *VLDB Conference*, pages 574–584, 1995.
4. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB Conference*, pages 426–435. Morgan Kaufmann Publishers, Inc., 1997.
5. B. V. Dasarathy. Data mining tasks and methods: Classification: nearest-neighbor approaches. pages 288–298, 2002.
6. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2000. 2nd Edition.
7. K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers, IEC*, 24:750–753, 1975.
8. E. Gómez-Ballester, L. Micó, and J Oncina. Some improvements in tree based nearest neighbour search algorithms. *0302-9743 - LNCS - LNAIntelligence*, (2905):456–463, 2003.
9. E. Gómez-Ballester, L. Micó, and J. Oncina. Some approaches to improve tree-based nearest neighbour search algorithms. *Pattern Recognition*, 39(2):171–179, 2006.
10. G. Navarro. Searching in metric spaces by spatial approximation. In *SPIRE '99: Proceedings of the String Processing and Information Retrieval Symposium*, page 141. IEEE Computer Society, 1999.
11. H. Noltemeier, K. Verbarg, and C. Zirkelbach. Monotonous bisector* trees - a tool for efficient partitioning of complex scenes of geometric objects. In *Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative*, pages 186–203, London, UK, 1992. Springer-Verlag.
12. J. Oncina, F. Thollard, E. Gómez-Ballester, L. Micó L., and F. Moreno-Seco. A tabular pruning rule in tree-based pruning rule fast nearest neighbour search algorithms. *LNCS*, (4478):306–313, 2007.
13. Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors. *Nearest-Neighbor Methods in Learning and Vision*. The MIT Press, 2006.
14. E. Vidal. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA). *Pattern Recognition Letters*, 15:1–7, 1994.
15. R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, 1974.
16. P.N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.