A tabular pruning rule in tree-based fast Nearest Neighbor Search algorithms

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

[1] Dept. Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03071 Alicante, Spain
{oncina,eva,mico,paco}@dlsi.ua.es
[2] Laboratoire Hubert Curien (ex EURISE) - UMR CNRS 5516
18 rue du Prof. Lauras - 42000 Saint-Étienne Cedex 2, France
thollard@univ-st-etienne.fr

Abstract. Some fast nearest neighbor search (NNS) algorithms using metric properties have appeared in the last years for reducing computational cost. Depending on the structure used to store the training set, different strategies to speed up the search have been defined. For instance, pruning rules avoid the search of some branches of a tree in a tree-based search algorithm. In this paper, we propose a new and simple pruning rule that can be used in most of the tree-based search algorithms. All the information needed by the rule can be stored in a table (at preprocessing time). Moreover, the rule can be computed in constant time. This approach is evaluated through real and artificial data experiments. In order to test its performance, the rule is compared to and combined with other previously defined rules.

1 Introduction

Nearest Neighbor Search (NNS) techniques aim at finding the nearest point of a set to a given test point using a distance function [4]. The naïve approach is some times a bottleneck due to the large number of distances to be computed. Many methods have been developped in order to avoid the exhaustive search (see [3] and [2] for a survey). Tree-based structures are very popular in most of the proposed algorithms [6, 5, 10, 1, 9], as this structure provides a simple way to avoid the exploration of some subsets of points. Among these methods, only some of them are suitable for general metric spaces, i.e., spaces where the objects (prototypes) need not to be represented as a point, and only require a properly defined distance function. The most popular and refereed algorithm of such a type was proposed by Fukunaga and Narendra (FNA) [6]. This algorithm is very suitable for studying new tree building strategies and new pruning rules [7, 8] as a previous step for extending the new ideas to other tree-based algorithms.

In this paper a new pruning rule is presented. The two keypoints in favor of this rule are its simplicity (only a table of "distances" is stored) and its efficiency (it allows a constant time pruning). The new rule may be used with the FNA

algorithm in any metric space (even in a vector space with an appropiate distance metric). In a classical way, the FNA algorithm will serve as a baseline for the comparison with other techniques.

The paper is organized as follow: we will first introduce the basic algorithm (section 2). We introduce the different pruning rules that were used in the experiment in section 3 and 4. We will provide a comparative experiment on either artificial and real world data (section 5). We then conclude suggesting some future works (section 6).

2 The basic algorithm

The FNA is a fast search method that uses a binary tree structure. Each leaf stores a point of the search space. At each node t is associated S_t, the set of the points stored in the leaves of t sub-tree. Each node stores M_t (the representative of S_t) and the radius of S_t, $R_t = \max_{x \in S_t} d(M_t, x)$.

The tree is generally built using recursive calls to a clustering algorithm. In the original FNA the c-means algorithm was used. In [7] some other strategies were explored: in the best method, namely the *Most Distant from the Father tree* (MDF), the representative of the left node was the same than the representative of its father. Thus, each time an expansion of the node is necessary, only one new distance must be computed (instead of two), reducing the number of distances computed. As the pruning rules apply on any tree, in the following, the tree will be built using the MDF method.

In algorithm 1, a simplified version of FNA is presented; only the `Prune_FNR` function call must be changed when considering another pruning rule. In order to make the pseudo-code simpler, the $d_{\min}$ and nn are considered global variable. Also, only binary trees with one point on the leaves are considered.

The use of the Fukunaga and Narendra Rule (FNR) for pruning internal nodes is detailed in [6].

When a new sample point x is given, its nearest neighbor nn is searched in the tree using a depth-first strategy. At a given level, the node t with a smaller distance $d(x, M_t)$ is explored first. In order to avoid the exploration of some branches of the tree the FNA uses the FNR rule.

3 A review of pruning rules

Fukunaga and Narendra Rule (FNR)

The pruning rule defined by Fukunaga and Narendra for internal nodes only makes use of the information in the node t to be pruned (with representant M_t and radius R_t) and the hyperspherical volume centered in the sample point x with radius $d(x, nn)$, where nn is the nearest prototype considered up to the moment.

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

Algorithm 1: **search(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);
    d_r = d(x, M_r) ;      d_ℓ = d(x, M_ℓ);
    update d_min and nn;
    if d_ℓ < d_r then
        if not Prune_FNR(ℓ) then
            search(ℓ, x);
        if not Prune_FNR(r) then
            search(r, x);
    else
        if not Prune_FNR(r) then
            search(r, x);
        if not Prune_FNR(ℓ) then
            search(ℓ, x);
```

The Sibling Based Rule (SBR)

Given two sibling nodes r and ℓ, this rule requires that each node r stores the distance between the representative of the node, M_r, and the nearest point, e_ℓ, in the sibling node ℓ (S_ℓ).

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

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 (see [8] for more details). Given a node ℓ, a set of prototypes $\{e_i\}$ is defined in the following way:

$$\begin{aligned} G_1 &= S_\ell \\ e_i &= \text{argmax}_{p \in G_i} d(p, M_\ell) \\ G_{i+1} &= \{p \in G_i : d(p, M_r) < d(e_i, M_r)\} \end{aligned}$$

where M_r is the representative of the sibling node r, and G_i are auxiliary sets of prototypes.

At preprocessing time, the distances $d(M_r, e_i)$ are stored in each node ℓ. This process is repeated similarly for the sibling node.

Rule: No $y \in S_\ell$ can be the nearest neighbor if there is an integer i such that:

$$d(M_r, e_i) \geq d(M_r, x) + d(x, nn) \quad (1)$$
$$d(M_\ell, e_{i+1}) \leq d(M_\ell, x) - d(x, nn) \quad (2)$$

Cases $i = 0$ and $i = s$ are also included not considering equations (1) or (2) respectively. Note that condition (1) is equivalent to SBR rule when $i = s$ and condition (2) is equivalent to FNR rule when $i = 0$.

4 The table rule (TR)

This rule prunes by taking the current nearest neighbor as a reference. In order to do so the distance from a prototype p to a set of prototypes S is defined as $d(p, S) = \min_{y \in S} d(p, y)$. At preprocess time, the distances from each prototype to each node set S_t in the tree are computed and stored in a table, allowing a constant time pruning. Note that the size of this table grows with the square of the number of prototypes since, as the tree is binary, the number of nodes is two times the number of prototypes.

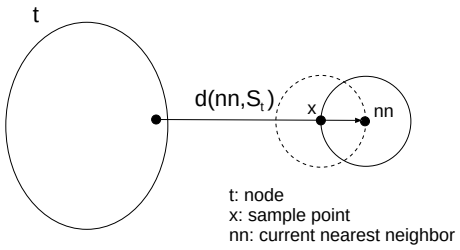

Fig. 1. Application of the table rule

Rule: Figure 1, Present a graphical view of the table rule.

Proposition 1 (Table Rule) *Given the table rule* $(2d(x, nn) < d(t, nn))$*, no prototype* e_i *in node* t *can be nearest to the test sample* x *than* nn*, i. e.*

$$\forall e_i \in t, \quad d(x, e_i) \geq d(x, nn)$$

Proof:
Let $e_i \in S_t$. By the definition of the distance between a point and a node

$$d(nn, S_t) = min_{e_i \in S_t} d(e_i, nn)$$

and thus

$$d(nn, S_t) \leq d(e_i, nn)$$

Moreover, by the triangle inequality, we have:

$$d(e_i, nn) \leq d(e_i, x) + d(x, nn)$$

Combining these inequalities, we have:

$$\begin{aligned} & d(nn, S_t) \leq d(e_i, nn) \leq d(e_i, x) + d(x, nn) \\ \Rightarrow \;& d(e_i, x) \;\geq d(nn, S_t) - d(x, nn) \end{aligned}$$

using the table rule, we finally have:

$$d(e_i, x) \geq 2d(x, nn) - d(x, nn) = d(x, nn)$$

which completes the proof.

5 Experiments

As seen in the proof of the correctness of the table rule, it is only required that d is a true distance. In particular, on the contrary to other techniques such as the well known kd-tree algorithm, a vector space is not needed in order to apply the table rule.

In order to evaluate the power of the *table rule*, the performance of the algorithm has been measured in real and artificial data experiments using the most significative combinations of the pruning rules.

In the artificial data set up, the prototypes where obtained from a 5 and 10-dimensional uniform distribution in the unit hypercube.

A first experiment was performed using increasing size prototypes sets from $1,000$ prototypes to $8,000$ in steps of $1,000$ for 5 and 10 dimensional data. Each experiment measures the average distance computations of $16,000$ searches ($1,000$ searches over 16 different prototypes sets). The samples were obtained from the same distribution.

Figures 2 and 3 show the results for some combinations of the pruning rules where "f", "s", "g" and "t" stand for the "Fukunaga", "sibling", "generalized" and "table" pruning rules respectively. Standard deviation of measures is also included (though with value almost negligible).

As it can be observed, the table pruning rule, when applied alone, can achieve $\sim 50\%$ distance computations reduction, although additional reduction (up to $\sim 70\%$) can be achieved when combined with "f", "fs" or "g" pruning rules. In these three cases the differences are not noticeable. Obviously, as the time complexity of the generalized pruning rule is not constant, the combinations with "f" or "fs" are more appealing.

To show the performance of the algorithm with real data, some tests were carried out on a spelling task. A database of $38,000$ words of a Spanish dictionary was used.

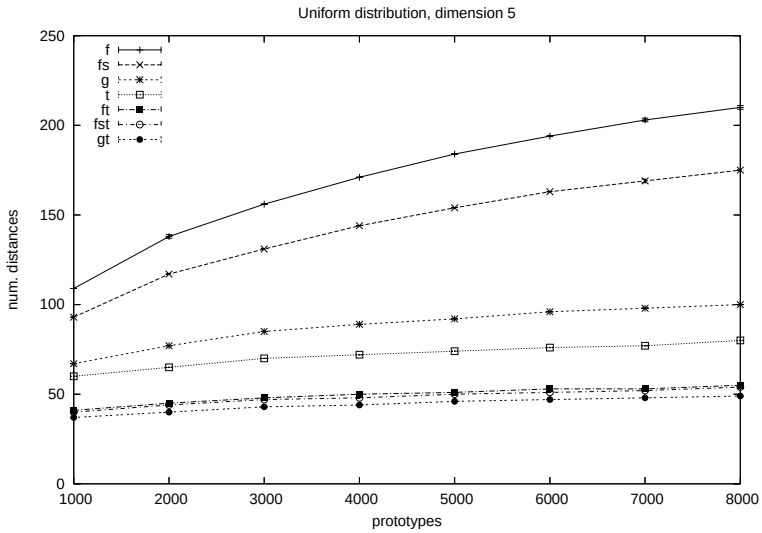

Fig. 2. Pruning rules combinations in a uniform distribution 5-dimensional space.

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 edit distance was used to compare the words. In these experiments, the values of the weighting operations costs of the edit distance (insertion, deletion and substitution) were fixed to 1. This makes the edit distance a mathematical distance which makes the table rule applicable. Please note that some fast NN search techniques (i.e. kd-tree) could not be applied here as the data could hardly be represented in a vector space.

Dictionaries of increasing size (from $1,000$ to $8,000$) were obtained extracting randomly words of the whole dictionary. The test points were $1,000$ distorted words obtained from randomly selected dictionary words. To obtain reliable results the experiments were repeated 16 times. The averages and the standard deviation are showed on the plots.

The experiment performed in Figures 2 and 3 for artificial data (average number of distance computations using increasing size prototype sets) were repeated in the spelling task. Results are shown in Figure 4.

The experiments show a reduction in the number of distance computations (around 40%) for the table rule when combined with "f", "fs" or "g" pruning rules.

On the contrary to the artificial data case, the table rule alone does not perform better than the generalized rule. Nevertheless, this is not problematic as combining the table rule with the two constant time pruning rules – namely the Fukunaga and/or the Sibling rule – outperforms the generalized rule performances.

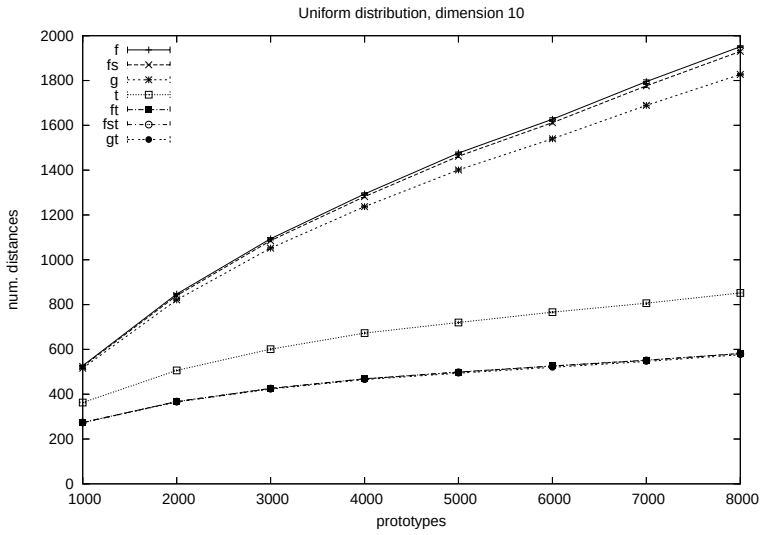

Fig. 3. Pruning rules combinations in a uniform distribution 10-dimensional space.

6 Conclusions and Further works

To summarize, a new pruning rule has been defined that can been applied in tree-based search algorithms. To apply the rule, a distance table should be computed and stored in preprocess time. This table rule stores the distances between each prototype in the training set and every node of the tree; its space complexity is therefore quadratic in the size of the training set.

As the experiments suggest, this rule save the computation of 70% of distances in the case of 10-dimensional data and 40% in the case of strings with training set around $8,000$ points when compared with the generalized rule.

In future works, a more exhaustive study of the rule will be performed. In particular, the idea is to study on the one hand which is the better combination of rules (with the minor cost), and on the other hand, what is the condition and order where each rule can be applied.

Other problem that should be explored is how to reduce the space complexity of the table rule.

7 Acknowledgments

The authors thank the Spanish CICyT for partial support of this work through projects DPI2006-15542-C04-01, TIN2006-14932-C02, GV06/166, the IST Programme of the European Community, under the PASCAL Network of Excellence, IST–2002-506778.

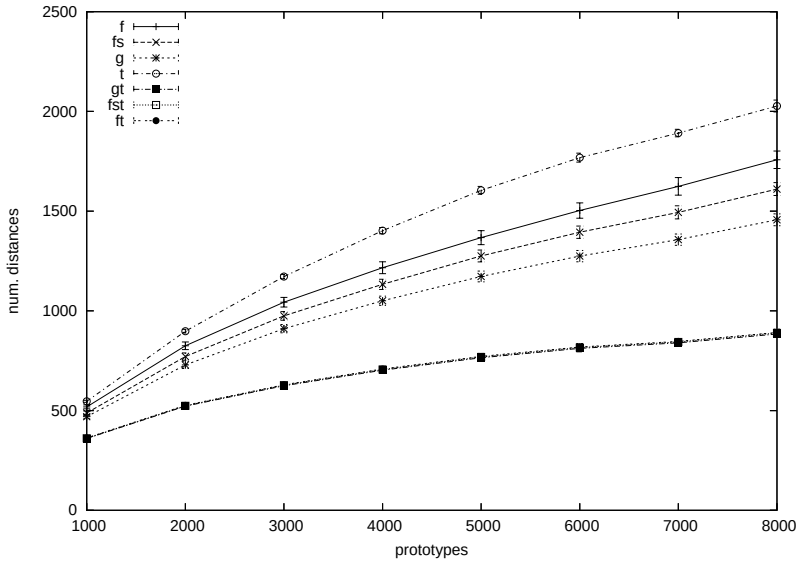

Fig. 4. Pruning rules combined in a spelling task.

References

1. S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the* 21^{st} *VLDB Conference*, pages 574–584, 1995.
2. E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.
3. B. V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, CA., 1991.
4. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2000. 2nd Edition.
5. J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.
6. 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.
7. E. Gómez-Ballester, L. Micó, and J Oncina. Some improvements in tree based nearest neighbour search algorithms. *0302-9743 - Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence*, (2905):456–463, 2003.
8. E. Gómez-Ballester, L. Micó, and J. Oncina. Some approaches to improve tree-based nearest neighbour search algorithms. *Pattern Recognition*, 39(2):171–179, February 2006.
9. J. McNames. A fast nearest neighbor algorithm based on a principal axis tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):964–976, September 2001.
10. 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.