A Pruning Rule Based on a Distance Sparse Table for Hierarchical Similarity Search Algorithms

Eva Gomez-Ballester, Luisa Micó, and Jose Oncina

Dept. Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03080 Alicante, Spain
{eva,mico,oncina}@dlsi.ua.es

Abstract. Nearest neighbour search is a simple technique widely used in Pattern Recognition tasks. When the dataset is large and/or the dissimilarity computation is very time consuming the brute force approach is not practical. In such cases, some properties of the dissimilarity measure can be exploited in order to speed up the search. In particular, the metric properties of some dissimilarity measures have been used extensively in fast nearest neighbour search algorithms to avoid dissimilarity computations. Recently, a distance table based pruning rule to reduce the average number of distance computations in hierarchical search algorithms was proposed. In this work we show the effectiveness of this rule compared to other state of the art algorithms. Moreover, we propose some guidelines to reduce the space complexity of the rule.

Keywords: fast nearest neighbour search, metric space, tree-based algorithms, pruning rules.

1 Introduction

Similarity search has been applied in many areas as pattern recognition, machine learning, computer vision, robotics or data mining were data has complex representations and then, complex functions are used to quantify their dissimilarities.

Some examples of proximity searches are range search, nearest neighbour search, k-nearest neighbours search, reverse nearest neighbour search or proximity join search [14]. In this work we focus on nearest neighbour search. That is, given a set P of *objects*, a *query* object q and a dissimilarity measure $d(\cdot,\cdot)$, the goal is to retrieve the object p_{nn} of the set P closest to the query object according to d:

$$p_{nn} = \underset{\forall p \in P}{\operatorname{argmin}}\, d(p,q)$$

A brute force implementation is impractical when the dissimilarity measure is very time consuming and/or the dataset is very large. As a result, many techniques have been devised to speed up the search.

Some of this techniques relies on the fact that some of the dissimilarity measures are metrics and then, they fulfils the the triangle inequality. For example,

N. da Vitora Lobo et al. (Eds.): SSPR&SPR 2008, LNCS 5342, pp. 926–936, 2008.

to avoid distance computations a lower bound of the distance can be defined using some precomputed distances [3][12].

In the last few years, a large number of works appeared in the literature related with such type of methods. The interest of these methods relies on the possibility of providing high extensibility when comparing with methods that requires more specific information, such as a vector representation. Some interesting surveys have been published recently [1,7,10].

The use of hierarchical data structures is very common in most of the proposed algorithms [11,9,15,17,13], as these structures provide a simple way to avoid the exploration of some subsets of objects. Several searching strategies can be performed in these structures, the most popular are depth-first and best-first strategies (even a combination of both), but independently of the strategy, similar pruning rules can be applied.

One of the first hierarchical-based search algorithms was proposed by Fukunaga and Narendra (`FN`) [11]. This algorithm is very suitable for studying tree building strategies (based on different types of space partitioning) with minimum changes in the other properties that we want to keep (pruning rules, searching strategy, ...) and *vice versa*. The algorithm is also very suitable to apply new pruning rules as a previous step for extending it to other hierarchical search methods.

Recently a new pruning rule for hierarchical similarity search algorithms based on the use of a distance table was proposed [8]. To apply the rule, the distance table should be built up and stored in preprocessing time. This table stores the distances between the nodes of the hierarchical structure and the objects in the data set.

In this work some strategies are proposed for reducing the space complexity of the table (quadratic with the size of the dataset) and some experiments are performed to test their effectiveness. To perform the experiments some state of the art fast Nearest Neighbour Search (NNS) algorithms have been used: the `FN` (using the same hierarchical structure mentioned above) [11], the Spatial Approximation Tree (`SAT`) [6] and the vp-tree [15] algorithms. They will serve as a baseline for the comparisons.

The paper is organized as follows: we first introduce the basic algorithm and the pruning rules (section 2). In section 3, we provide experiments on artificial and real world data. We conclude suggesting some future works in section 4.

2 The Basic Algorithm

The `FN` is a NNS algorithm that uses a hierarchical data structure where objects are stored in nodes. With each node t is associated S_t, the set of the objects stored in the leaves of the t sub-tree. Each node stores M_t (a representative of S_t) and the radius R_t of a ball centered in M_t that includes 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 `FN` algorithm the c-means algorithm was used. In [2] other strategies were explored. The best strategy lied in using the representative of the father

node as one of the representatives of the two child nodes and the other as its farthest object in the set. Thus, each time an expansion of the node is necessary, only one new distance (instead of two) should be computed during the search, reducing the number of distance computations.

In Algorithm 1, a simplified version of `FN` algorithm is presented. The `Prune_FNR` function call should be changed when considering other pruning rules. In order to make the pseudo-code simpler, p_{nn} (actual nearest neighbour to the query) and d_{nn} (distance from the query to p_{nn}, initially with $-\infty$ value) are considered global variable. Only binary trees with one object on the leaves are considered. Taken into account that leaf nodes only have one object, in the algorithm it is not necessary to define a special case pruning rule for leaves (this is why the leaves are not considered in the algorithm). When a new query is given, its nearest neighbour is searched in the tree using a depth-first strategy (the best-first strategy was also explored, but despite the results reported in [5], in this case, similar results were obtained. Due to lack of space we omit them in the experimental section).

Algorithm 1: **search(t, q)**

```
Data: t: a node tree ; q: query;
Result: p_nn: the nearest neighbour object; d_nn: the distance to q_nn;
if t is not a leaf then
    r = right_child(t); ℓ = left_child(t);
    d_r = d(q, M_r);        d_ℓ = d(q, M_ℓ);        //d_ℓ previously computed;
    update d_nn and p_nn;
    if d_ℓ < d_r then
        if not Prune_FNR(ℓ) then
            search(ℓ, q);
        if not Prune_FNR(r) then
            search(r, q);
    else
        if not Prune_FNR(r) then
            search(r, q);
        if not Prune_FNR(ℓ) then
            search(ℓ, q);
```

Next, we present the two pruning rules used in this work. If the disimilarity function is a metric, (the triangle inequality is fulfilled) it is guaranteed that the nearest neighbour can't be found among the eliminated elements with these rules.

2.1 Fukunaga and Narendra Rule (`FNR`)

This rule only makes use of the information in the node t to be pruned (with representative M_t and radius R_t) and the hyperspherical volume centered in the

query q with radius $d(q, p_{nn})$, where p_{nn} is the nearest object considered up to the moment. To apply this rule to a node t, the distance from the query q to the representative of the node should be available.

Proposition 1 ([8]). *No $y \in S_t$ can be the nearest neighbour to q if $d(q, p_{nn}) + R_t < d(q, M_t)$.*

2.2 Table Rule (TR)

In order to use the table pruning rule (TR), the distance from an object p to a node t is defined as $d(p, t) = \min_{e \in S_t} d(p, e)$ (this definition can be seen as a particular case of the distance between two sets [16]). At preprocessing time, the distances from each object to each node of the tree are computed and stored in a table. Note that the size of this table grows with the square of the number of objects in the database.

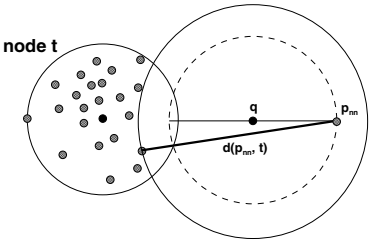

Fig. 1. Application of the table rule

The TR states that no object in a set can be the nearest neighbour to a query if the distance from the current nearest neighbour to the set is bigger than two times the distance from the current nearest neighbour to the query (see Figure 1).

Proposition 2 ([8]). *Let t be a node such that $d(p_{nn}, t) \geq 2d(p_{nn}, q)$. Then, $d(e, q) \geq d(p_{nn}, q)\ \forall e \in S_t$*

2.3 Reducing the Table Size

As it was stated in the previous section, the distances between the objects in the dataset and all the nodes in the tree should be computed and stored in the table to use the TR. Then, as the size of the table grows with the square of the number of objects, the space complexity can be a bottleneck for using this rule. In this section we will present some ideas about how to detect, in preprocessing time, in which circumstances the application of the TR will be useless. In these cases, the computation and storage of $d(n, t)$ (where n is an object and t a node) can be avoided. It is evident that in order to have effective storage reductions, the table should be implemented as a sparse matrix.

Let n be an object and let t be a node of the tree. Let we denote by $\hat{d}_{nn}$ the expected value of the distance to the nearest neighbour ($\hat{d}(q, p_{nn})$) (of course this

value is unknown in preprocessing time, we are going to use it as a parameter in the table reduction system). There are two circumstances where the TR is going to be probably useless:

1. When $\hat{d}_{nn} > \frac{d(n,t)}{2}$. In such case the TR is not going to prune and then the distance $d(n,t)$ is not stored in the table. This is the situation where TR is not fulfilled because the object n is very near to the node t.
2. When $\hat{d}_{nn} < d(M_t, n) - R_t$. This case arises when we expect that the FNR rule is going to prune and then, the result of the TR is useless. This formula derives from an approximation of the FNR where we suppose that n is very near to the query and then $d(M_t, n) \sim d(M_t, q)$ (this is usually the case after some steps of the algorithm [4]). In this case the FNR will be effective and then the TR is probably not needed to prune node t. In this case, the distance $d(n,t)$ is not stored in the table.

The experiment section analyses the price to pay, in distance computations, versus table size reduction obtained when our techniques are applied.

3 Experiments

Some experiments with synthetic and real data were carried out to evaluate our proposal. In the case of synthetic data, objects were obtained randomly from the unit hypercube with a uniform distribution and for several dimensions (5, 10, 15 and 20). In this case, the Euclidean distance was applied. As real data experiment, the Spanish dictionary in http://www.sisap.org using the Levenshtein distance has been used. In all the experiments sets of 1000 queries were used. Each experiment was repeated 10 times. The average and the deviations of each experiment are shown in the plots.

Moreover, comparison with other state of the art fast NNS methods was performed. In particular, the following algorithms were used: vp-tree (VPT) [15], SAT [6] and FN [11].

3.1 Experiments in the Unit Hypercube

In the first set of experiments, the results of the FN algorithm, using both FNR and TR (FNT in the figures), are compared with VPT, SAT[1] and the basic FN algorithm (using only FNR). Figure 2 shows the average number of distance computations by query when increasing the size of the dataset.

The results show that for lower dimensions, the average number of distance computations when FNR is used alone (FN curve) or in combination with TR (FNT curve) outperforms all the other methods, although this difference decreases when the dimension grows. This is an expected behaviour, because of the *curse of the dimensionality*: for high dimensions (30 or more) all the methods compute all the distances (see Figure 3) and then, all are equivalent to the brute force

[1] Code for VPT and SAT was obtained from http://www.sisap.org

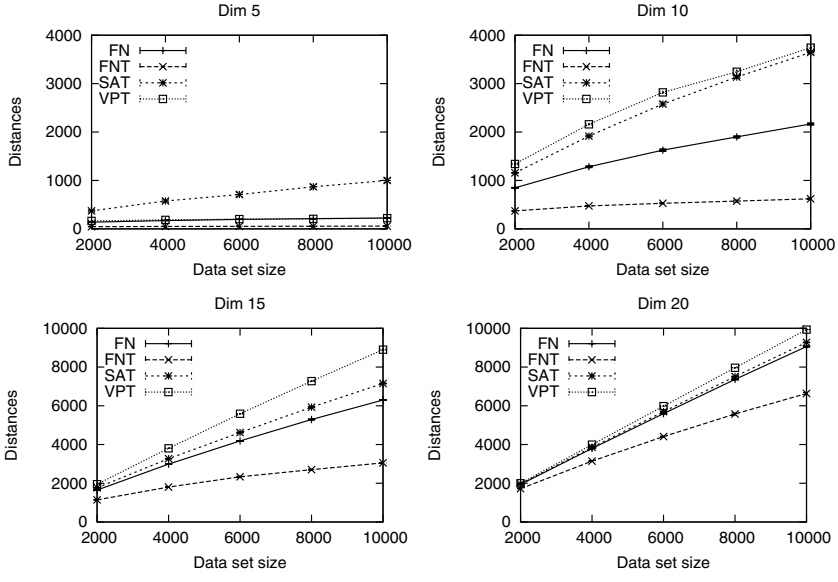

Fig. 2. Distance computations using increasing dataset size for several methods in the unit hypercube. In the particular case of 5-dimensional space, the best behaviour is obtained with the FNT method, and the worst for the SAT method.

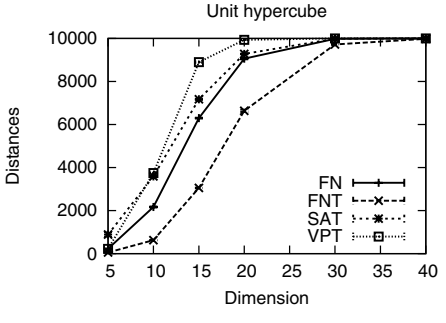

Fig. 3. Distance computations using a dataset of 10000 objects and different dimensions

algorithm. In this figure can be observed that for all dimensions the FNT method (use of both, FNR and TR rule) outperforms all the other methods. Moreover, although the behaviour of the FNT method also degenerates with the dimension, it degenerates for higher dimensions more slowly than the others. For example, the VPT, FN and SAT methods compute almost all the distances to the training set in a 20-dimensional space. To have a similar result with FNT method we have to go to a 30-dimensional space.

Reduction of the table. Due to the quadratic space complexity, the use of the `TR` pruning rule can be a problem when using very large data sets. A a second

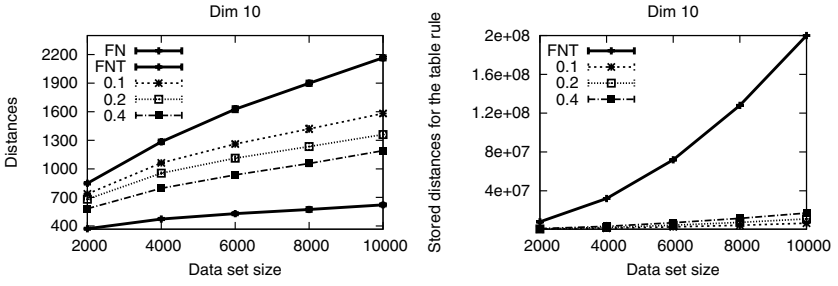

Fig. 4. Distance computations (left) and number of stored distances on the table (right) when the dataset size increases, using several values of the parameter $\hat{d}_{nn}$

group of experiments have been performed to study the power of the strategies described in section 2.2 to reduce the table size while trying to avoid an increase of the distance computations during the search.

Figure 4 shows the behaviour of FNT (left) and the reductions of the size of the table (right) when $\hat{d}_{nn}$ increases (from 0.1 to 0.8 depending on the dimension). This figure also shows the average number of distance computations when the table is full (FNT curve) and when is empty (FN curve). This behaviour is directly related with the number of distances stored in the table. As the number of distances stored (in preprocessing time) in the table decreases, the number of distance computations (in search time) increases.

In all the cases significative reductions in the table size are obtained. Saves of around 80% of the table in high dimensions (dimension 20) and around 98% in low dimensions (dimension 5) are obtained without increasing very much the number of distance computations in relation to the use of the whole table.

Some experiments using a wide range of values of $\hat{d}_{nn}$ show that:

1. using only $\hat{d}_{nn} > \frac{d(n,t)}{2}$ the reduction of the table increases with $\hat{d}_{nn}$ (in dimension 10, for $\hat{d}_{nn} = 0$ reductions are 0%, and for $\hat{d}_{nn} = 0.75$ the reduction is around 90%).
2. using only $\hat{d}_{nn} < d(M_t, n) - R_t$, the reduction of the table decreases with the increase of $\hat{d}_{nn}$ (in dimension 10, for $\hat{d}_{nn} = 0$ reductions are around 96%, and for $\hat{d}_{nn} = 0.75$ the reduction is around 60%).

If both criteria are used, a minimum in the number of distance computations for a specific value of $\hat{d}_{nn}$ can be found. This minimum is related with a minimum reduction in the size of the table. This behaviour can be observed in Figure 5. For very high values of $\hat{d}_{nn}$ the main reduction on the table is obtained using the first criterion. On the other hand, for very low values of d_{nn}, the main reduction on the table is obtained with the contribution of the second criterion.

These results confirm that a significative reduction in the number of computed distances can be obtained using only a small percentage of the table.

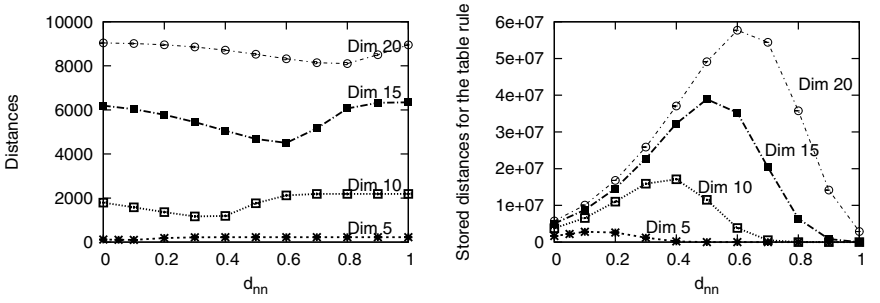

Fig. 5. Distance computations (left) and number of stored distances on the table (right) for several values of the parameter $\hat{d}_{nn}$ using a 10000 samples database

3.2 Experiments with the Spanish Dictionary

Similar experiments were repeated simulating a spell corrector task using a Spanish dictionary (`http://www.sisap.org`). The words to correct were obtained 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 operation costs for the edit distance were fixed to 1 for insertions, deletions and substitutions. As, in this case, the edit distance is a metric (symmetry, nonnegativity and triangle inequality properties are fulfilled), it makes the `FNR` and `TR` rules applicable.

Figures 6 and 7 show similar behaviour that Figures 2 and 4 when strings are used: the average number of distance computations when `FNR` is used alone or in combination with `TR` outperforms all the other methods. Moreover, similar reduction of the table can be achieved (around 85%) maintaining a good reduction in the average number of computed distance when comparing with the basic `FN` algorithm (`FNR` only).

Also, a final experiment (similar to Figure 5) was repeated using the dictionary. It can be observed in Figure 8 a minimum number of distance computations

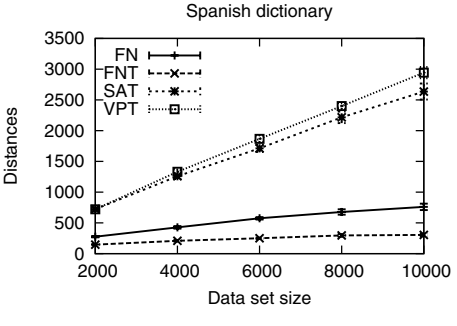

Fig. 6. Distance computations with increasing dataset size for different methods

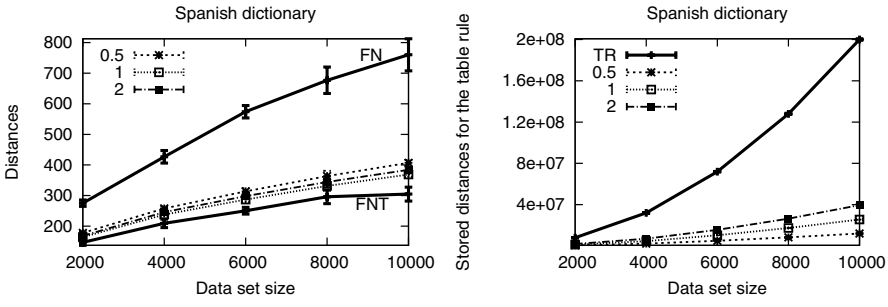

Fig. 7. Distance computations (on the left) and number of stored distances on the table (on the right) for different values of the parameter $\hat{d}_{nn}$

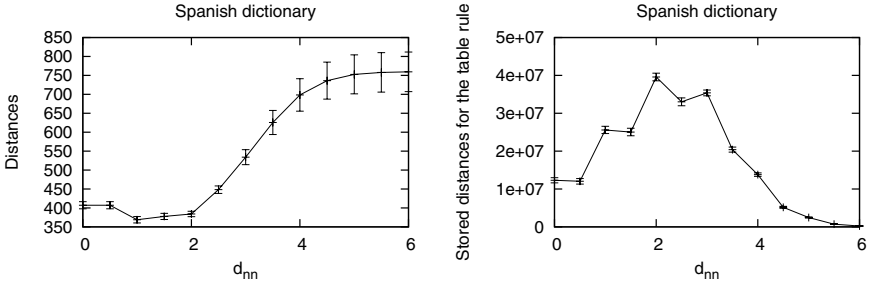

Fig. 8. Distance computations (left) and number of stored distances on the table (right) for different values of the parameter $\hat{d}_{nn}$ using 10000 samples are training set

for a specific value of $\hat{d}_{nn}$. This minimum is related with a minimum reduction in the size of the table. These results confirm that also for the dictionary a significative reduction in the number of computed distances can be obtained using only a small percentage of the table.

4 Conclusions and Further Works

A recently proposed pruning rule for hierarchical search algorithms based on the use of precomputed distances between nodes in the tree and the objects of the training set has been explored and compared with some state of the art methods. In this work some proposals to reduce the size of the table have also been presented. Two criteria, based on the use of a parameter ($\hat{d}_{nn}$) was defined. The key idea is to avoid the storage of distances that rarely would be used (`TR` is not fulfilled because objects are very close to nodes or `FNR` is fulfilled).

As the results suggest, significant reductions in distance computations can be obtained when comparing with the basic `FN` algorithm and other state of the art algorithms (`SAT` and `VPT`). Moreover, this criteria to reduce the table size

has shown its effectiveness allowing significant table reductions while increasing moderately the number of distance computations.

In future works, more selective reduction of the table should be explored.

Acknowledgments. The authors thank the Spanish CICyT for partial support of this work through projects DPI2006-15542-C04-01, the IST Programme of the European Community, under the PASCAL Network of Excellence, IST–2002-506778, and the program CONSOLIDER INGENIO 2010 (CSD2007-00018).

References

1. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.L.: Searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
2. Gómez-Ballester, E., Micó, L., Oncina, J.: Some improvements in tree based nearest neighbour search algorithms. In: Sanfeliu, A., Ruiz-Shulcloper, J. (eds.) CIARP 2003. LNCS, vol. 2905, pp. 456–463. Springer, Heidelberg (2003)
3. Vidal, E.: New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (aesa). Pattern Recognition Letters 15(1), 1–7 (1994)
4. Moreno-Seco, F., Micó, L., Oncina, J.: A modification of the laesa algorithm for approximated k-nn classification. Pattern Recognition Letters 22, 1145–1151 (2003)
5. Moreno-Seco, F., Micó, L., Oncina, J.: Approximate nearest neighbour search with the fukunaga and narendra algorithm and its application to chromosome classification. In: Sanfeliu, A., Ruiz-Shulcloper, J. (eds.) CIARP 2003. LNCS, vol. 2905, pp. 322–328. Springer, Heidelberg (2003)
6. Navarro, G.: Searching in metric spaces by spatial approximation. VLDB Journal 11(1), 28–46 (2002)
7. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces. ACM Trans. Database Syst. 28(4), 517–580 (2003)
8. Oncina, J., Thollard, F., Gómez-Ballester, E., Micó, L., Moreno-Seco, F.: A tabular pruning rule in tree-based pruning rule fast nearest neighbour search algorithms. In: Martí, J., Benedí, J.M., Mendonça, A.M., Serrat, J. (eds.) IbPRIA 2007. LNCS, vol. 4478, pp. 306–313. Springer, Heidelberg (2007)
9. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software 3, 209–226 (1977)
10. Clarkson, K.: Nearest-neighbor searching and metric space dimensions. In: Nearest-Neighbor Methods for Learning and Vision: Theory and Practice, pp. 15–59. MIT Press, Cambridge (2006)
11. Fukunaga, K., Narendra, P.M.: A branch and bound algorithm for computing k-nearest neighbors. IEEE Transactions on Computers, IEC 24, 750–753 (1975)
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 15, 9–17 (1994)
13. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Proceedings of the 23rd International Conference on VLDB, Athens, Greece, pp. 426–435. Morgan Kaufmann Publishers, San Francisco (1997)

14. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Springer, Heidelberg (2006)
15. 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)
16. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. Wiley, New York (2000)
17. 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)