Clustering-Based Similarity Search in Metric Spaces with Sparse Spatial Centers

Nieves Brisaboa[1], Oscar Pedreira[1], Diego Seco[1],
Roberto Solar[2,3], Roberto Uribe[2,3]

[1] Database Laboratory, University of A Coruña
Campus de Elviña s/n, 15071 A Coruña, Spain
[2] Dpto. Ingeniería en Computación, Universidad de Magallanes
Casilla 113-D, Punta Arenas, Chile
[3] Grupo de Bases de Datos (UART), Universidad Nacional de la Patagonia Austral
Rio Turbio, Santa Cruz, Argentina
{brisaboa,opedreira,dseco}@udc.es,{rsolar,ruribe}@ona.fi.umag.cl

Abstract. Metric spaces are a very active research field which offers efficient methods for indexing and searching by similarity in large data sets. In this paper we present a new clustering-based method for similarity search called SSSTree. Its main characteristic is that the centers of each cluster are selected using *Sparse Spatial Selection* (SSS), a technique initially developed for the selection of pivots. SSS is able to adapt the set of selected points (pivots or cluster centers) to the intrinsic dimensionality of the space. Using SSS, the number of clusters in each node of the tree depends on the complexity of the subspace it represents. The space partition in each node will be made depending on that complexity, improving thus the performance of the search operation. In this paper we present this new method and provide experimental results showing that SSSTree performs better than previously proposed indexes.

Key words: Similarity search, metric spaces, sparse spatial selection, cluster center selection.

1 Introduction

Searching in metric spaces is a very active research field since it offers efficient methods for indexing and searching by similarity in non-structured domains. For example, multimedia databases manage objects without any kind of structure like images, audio clips or fingerprints. Retrieving the most similar fingerprint to a given one is a typical example of similarity search. The problem of text retrieval is present in systems that range from a simple text editor to big search engines. In this context we can be interested in retrieving words similar to a given one to correct edition errors, or documents similar to a given query. We can find more examples in areas such as computational biology (retrieval of DNA or protein sequences) or pattern recognition (where a pattern can be classified from other previously classified patterns) [1, 2].

The problem of similarity search can be formalized through the concept of metric spaces. A metric space $(\mathbb{X}, d)$ is composed of a universe of valid objects $\mathbb{X}$ and a distance function $d : \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}^+$ satisfying the properties of *non-negativity* ($d(x, y) > 0$ and if $d(x, y) = 0$ then $x = y$), *symmetry* ($d(x, y) = d(y, x)$) and the *triangle inequality* ($d(x, z) \leq d(x, y) + d(y, z)$). The distance function determines the similarity between any two objects from that universe. A collection of words with the edit distance (computed as the number of characters to insert, delete or modify to transform a word into another), is an example of a metric space. A *vector space* is a particular case of a metric space, in which each object is composed of k real numbers. In vector spaces we can use any distance function from the family $L_s(x, y) = (\sum_{1 \leq i \leq k} |x_i - y_i|^s)^{\frac{1}{s}}$. For example, L_1 is the *Manhattan* distance, L_2 is the Euclidean distance and $L_\infty = max_{1 \leq i \leq k} |x_i - y_i|$ is the maximum distance. The *dimensionality* of a vector space is the number of components of each vector. Although general metric spaces do not have an explicit dimensionality, we can talk about their *intrinsic dimensionality* following the idea presented in [1]. The higher the dimensionality the more difficult the search.

Similarity search can involve different types of queries. Range search retrieves the objects that are within distance r (query range) to the query q, i.e., $\{u \in \mathbb{U} / d(q, u) \leq r\}$. *k-nearest neighbor* search retrieves the k nearest objects to the query q, i.e., the set $A \subseteq \mathbb{U}$ such that $|A| = k$ y $\forall\ u \in A,\ v \in \mathbb{U} - A,\ d(q, u) \leq d(q, v)$. Range search is the most general and k-nearest neighbor search can be implemented in terms of it [1].

Similarity search can be trivially implemented comparing the query with all the objects of the collection. However, the high computational cost of the distance function, and the high number of times it has to be evaluated, makes similarity search very inefficient with this approach. This has motivated the development of indexing and search methods in metric spaces that make this operation more efficient trying to reduce the number of evaluations of the distance function. This can be achieved storing in the index information that, given a query, can be used to discard a significant amount of objects from the data collection without comparing them with the query.

Although reducing the number of evaluations of the distance function is the main goal of indexing algorithms, there are other important features. Some methods can only work with discrete distance functions while others admit continuous distances too. Some methods are static, since the data collection cannot grow once the index has been built. Dynamic methods support insertions in an initially empty collection. Another important factor is the possibility of efficiently storing these structures in secondary memory, and the number of I/O operations needed to access them.

Search methods in metric spaces can be grouped in two classes [1]: *pivot*-based and *clustering*-based search methods. Pivot-based methods select a subset of objects from the collection as pivots, and the index is built computing and storing the distances from each of them to the objects of the database. During the search, this information is used to discard objects from the result without

comparing them with the query. The most important pivot-based methods are *Burkhard-Keller-Tree* (BKT) [3], *Fixed-Queries Tree* (FQT) [4], *Fixed-Height FQT* (FHQT) [5], *Fixed-Queries Array* (FQA) [6], *Vantage Point Tree* (VPT) [7] and its variants [8] [9], *Approximating and Eliminating Search Algorithm* (AESA) [10] and LAESA (*Linear AESA*) [11].

Clustering-based methods partition the metric space in a set of regions or clusters, each of them represented by a cluster center. In the search, complete regions are discarded from the result based on the distance from their center to the query. The most important clustering methods are *Bisector Tree* (BST) [12], *Generalized-Hyperplane Tree* (GHT) [13] and *Geometric Near-neighbor Access Tree* (GNAT) [14]. In section 2 we will briefly review how these methods work.

This paper presents a new index structure for searching in metric spaces called SSSTree. It is a clustering-based search method, and its main feature is that the cluster centers are selected applying *Sparse Spatial Selection* (SSS) [15], a strategy initially designed for pivot selection. SSS is adaptive, dynamic and the set of selected objects are well-distributed in the metric space. Applying SSS, the SSSTree is not balanced, but it is adapted to the complexity of the space, which is an important difference with previously proposed methods. Our hypothesis is that, applying SSS to select the cluster centers, the space partition will be more efficient and the search operation will show a better performance. Experimental results show that this new approach performs better than previously proposed methods.

Next Section briefly explains some basic concepts about clustering-based methods for similarity search. Section 3 analyzes the problem of pivot and cluster center selection and describes SSS. Section 4 presents SSSTree. In Section 5 experimental results are discussed and Section 6 finishes the paper with the conclusions and future work.

2 Previous work on clustering-based similarity search

Clustering-based search methods partition the metric space in several regions or clusters, each of them represented by a cluster center. The index stores the information of each cluster and its center. Given a query, complete clusters can be discarded from the result based on the distance from their centers to the query. In those clusters that can not be discarded, the query is sequentially compared with all the objects of the cluster.

There are two pruning criteria to delimit the clusters in clustering-based methods: generalized hyperplane and covering radius. The algorithms based in the hyperplane partitioning divide the metric space in a Voronoi partition. Given a set of cluster centers $\{c_1, c_2, \ldots, c_n\} \subset \mathbb{X}$, the Voronoi diagram is defined as the space subdivision in n areas in such a way that, $x \in Area(c_i)$ if and only if $d(x, c_i) < d(x, c_j)$, for $j \neq i$. In this type of algorithms, given a query (q, r), the clusters with no objects in the result set are directly discarded based in the query range and the distance from the query to the hyperplane that separates that cluster from the next one.

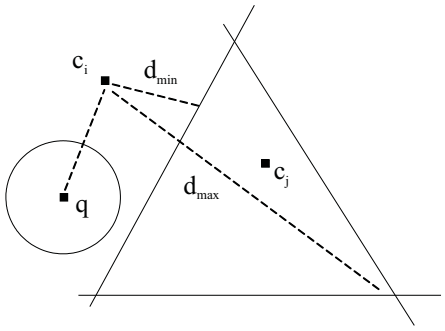


Fig. 1. Use of the max and min distances to discard objects in GNAT

In the case of the algorithms that use the covering radius for pruning, the space is divided in a set of spheres that can intersect, and a query can be included in more than one sphere. The covering radius is the distance from the cluster center to the furthest object from it. Knowing the distance from the query to the cluster center, the query range, and the covering radius, we can decide if that cluster contains objects in the result set or if it does not.

Bisector Tree (BST) [12] recursively divides the space storing the information about the clusters in a binary tree. In the root of the tree, two cluster centers are selected to make the division. The objects that belong to the first cluster are assigned to the left child node, and those that belong to the second cluster are assigned to the right child. This process is repeated in each node to recursively partition the space. For each cluster, the cluster center and the covering radius are stored. Given a query, the tree is traversed deciding in each node what branches (clusters) can be discarded from the result. *Generalized Hyperplane Tree* (GHT) [13] also applies a recursive partitioning to the space. However, during the search GHT does not use the covering radius to determine what clusters can be discarded. In this case, this decision is taken using the hyperplane place between the two cluster centers stored in each node.

Geometric Near-neighbor Access Tree (GNAT) [14] uses m cluster centers in each internal node of the tree. Each node of the tree stores a table with m rows (one for each cluster center) and m columns (one for each cluster). Cell (i, j) stores the minimum and maximum distances from the cluster center c_i to an object belonging to $Cluster_j$. During the search, the query q is compared with a cluster center c_i. Now, we can discard the clusters $Cluster_j$ such that $d(q, c_i)$ is not between that minimum and maximum distances. Figure 1 intuitively shows the meaning of these distances and how they can be used to decide in which clusters continue the search.

EGNAT [16] belongs to the group of algorithms based on compact partitions and is a secondary memory optimization of GNAT, in terms of space, disk accesses and distance evaluations.

3 The problem of pivots and cluster centers selection

3.1 Previous work

Something that most of the algorithms we have mentioned have in common is that both pivots and cluster centers are usually selected at random. However, it is evident that the specific set of selected reference objects has a strong influence in the efficiency of the search. The number of objects, their position in the space and their position with respect to the others, determine the ability of the index for discarding objects without comparing them with the query. Other important problem is how to determine the optimal number of reference objects. For example, in the case of pivot-based algorithms, we could think that the higher the number of elements chosen as pivots, the more efficient the search. But the query has to be compared both with the pivots and the objects that could not be discarded, so we have to reach a good trade-off between the number of pivots and the amount of objects that can be discarded using them.

Previous works have proposed several heuristics for pivot selection. For example, [11] selects as pivots the objects maximizing the sum of the distances to the already selected pivots. [7] and [14] try to obtain pivots far away from each other. [17] extensively analyzes this problem and shows experimentally the importance of this factor in the search performance. In addition, [17] proposes a criterion to compare the efficiency of two sets of pivots of the same size, and three pivot selection techniques based on that criterion. The first one is called *Selection*, which selects N random sets of pivots and finally uses the one maximizing the efficiency criteria. *Incremental* iteratively selects the set of pivots, adding to the set the object that more contributes to the efficiency criteria when added to the current set of pivots. *Local optimum* starts with a random set of pivots and in each iteration substitutes the pivot that less contributes to the efficiency criterion with other object. Although all these techniques select pivots that improve the search performance, in all of them the number of pivots has to be stated in advance. Therefore, the optimal number of pivots has to be computed by trial and error on a significant data collection (and this inevitably makes the search method static).

3.2 Sparse Spatial Selection (SSS)

Sparse Spatial Selection (SSS) [15] dynamically selects a set of pivots well distributed in the space. The hypothesis behind this selection strategy is that, if the pivots are "spatially" sparse in the metric space, they will be able to discard more objects in the search operation. To do this, when a new object is inserted in the database, it is selected as a new pivot if it is far away enough from the pivots already selected. We consider that the new object is far enough if its distance to any pivot is greater than or equal to $M\alpha$, being M the maximum distance between any two objects from the collection, and α a constant parameter that usually takes values are around 0.4. The parameter α has effect on the number of pivots selected. Figure 2 shows with several vector spaces that the optimal

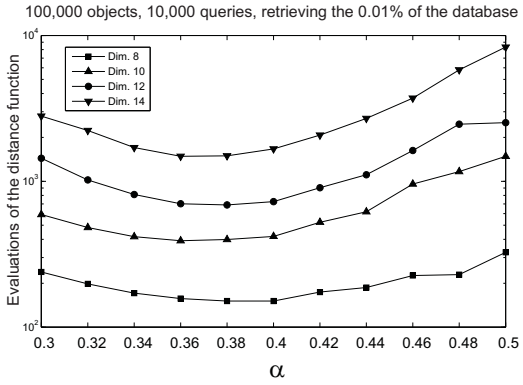


Fig. 2. Search efficiency in terms of the parameter α

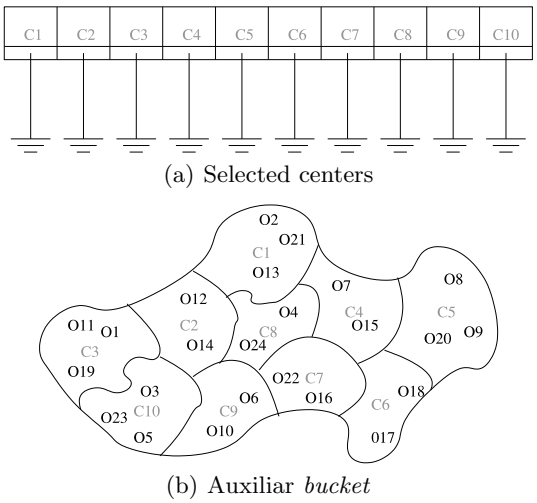
values of this parameter are always in the range from 0.35 to 0.4 (note also that the efficiency of the search is virtually the same for all the values included in this interval).

The results presented in [15] show that this strategy is more efficient than others previously proposed in most cases. Furthermore, SSS has other important features. SSS is dynamic, this is, the database can be initially empty, and the pivots will be selected when needed as the database grows. SSS is also adaptive, since it is no necessary to state in advance the number of pivots to select. As the database grows, the algorithm determines if the collection has become complex enough to select more pivots or not. Therefore, SSS adapts the index to the intrinsic dimensionality of the metric space [15].

Although SSS was initially designed for pivot selection, in this work it has been used for cluster center selection. Our hypothesis is that if the cluster centers are well distributed in the metric space, the partition of the space will be better and the search operation will be more efficient.

4 SSSTree: Sparse Spatial Selection Tree

SSSTree is a clustering-based method for similarity search based on a tree structure where the cluster centers of each internal node of the tree are selected applying *Sparse Spatial Selection* (SSS). Our hypothesis is that, using these cluster centers, the partition of the space will better and the performance of the search operation will be increased. In each node, the corresponding subspace is divided in several regions or clusters, and for each of them the covering radius is stored. An important difference with methods like GNAT is that SSSTree is not a balanced tree and not all the nodes have the same number of branches. The tree structure is determined by the internal complexity of the collection of objects.



(a) Selected centers

(b) Auxiliar *bucket*

Fig. 3. Situation after the selection of the cluster centers of the root tree.

4.1 Construction

In the construction process we use an additional data structure called *bucket*, which simply contains a subset of the objects of the collection. All the objects of the collection are initially in the bucket. In the root of the tree the cluster centers are selected applying SSS (the maximum distance can be estimated as in [15]). Therefore, the first object of the bucket will be the first cluster center. For the rest of the objects in the bucket, an object is selected as a new cluster center if it is at a distance greater than $M\alpha$ to the centers already selected. If an object is not a new cluster center, it is inserted in the bucket of its corresponding cluster (the cluster of its nearest center). Figures 3 and 4 show an example of tree construction after the selection of the cluster centers of the root node. The process is recursively repeated in each internal node of the tree (in each node, the objects of its corresponding cluster are stored in a bucket and we can proceed exactly in the same way), until clusters reach a minimum number of elements. The partition applied in each new node depends on the complexity of the cluster associated to it and does not have to use the same number of clusters.

In each internal node we have to estimate again the maximum distance in the cluster associated to it (since the maximum distance of the metric space is not valid for each new cluster). The maximum distance can vary even between different clusters in the same level. In next subsection we explain different ways to efficiently estimate the maximum distance into a cluster. Once estimated, the new cluster centers are selected, and the new subspaces are created. The process stops when a cluster has a number of objects less than or equal to a threshold δ or, alternatively, when the covering radius of the cluster is smaller than a given threshold.

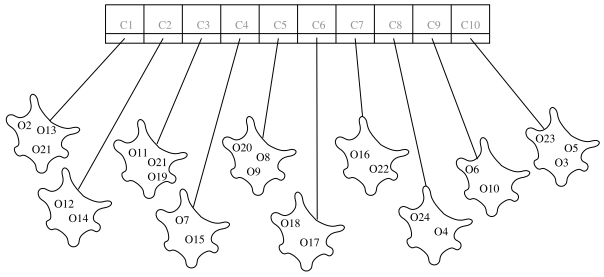


Fig. 4. SSSTree tree after the first space partition.

Applying this strategy for the selection of cluster centers, not all the nodes of the tree will have the same number of child nodes. Each cluster is divided in a number of regions which depends on the distribution and complexity of the data of that cluster. This is a very important difference with other structures like GNAT. The index construction adapts the index to the complexity and distribution of the objects of the metric space, and in each level of the tree only those needed clusters will be created. This property is derived from the fact that SSS is a dynamic selection strategy able to adapt the set of reference objects to the complexity of each space or subspace.

4.2 Estimation of the maximum distance M

One of the problems of the construction process is the need of computing the maximum distance M in each cluster. The naive way to compute this distance is to compare each object of the cluster with all the other objects (and this approach is too expensive). Although the index construction is usually an *off-line* process, the value of the maximum distance should be estimated to improve the index construction.

M is the maximum distance between any pair of objects of the cluster, and the covering radius RC is the distance from the cluster center to the furthest object of the cluster. Thus, $M \leq 2 \times RC$. As we can observe in figure 5, if we use $2 \times RC$ as the cluster diameter we can cover the same objects as using M as diameter. Therefore, $2 \times RC$ can be used as a good estimation of the maximum distance during the construction process.

4.3 Searching

During the search operation the covering radius of each cluster is used to decide in each step in which clusters we have to continue the search. The covering radius is the distance from the cluster center c_i, and the object of the cluster furthest to it. Therefore, in each step we can discard the cluster with center c_i if $d(q, c_i) - r > rc(c_i)$, where $rc(c_i)$ is the covering radius of that cluster. When the search reaches the leaves of the tree which could not be discarded, we have

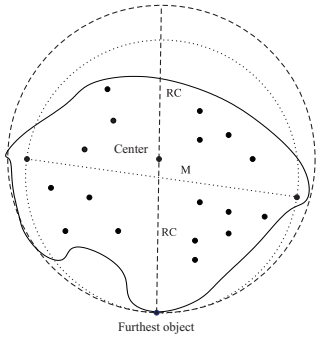


Fig. 5. Maximum distance estimation.

to compare the query against all the objects associated to that leaf to obtain the final result of the search.

5 Experimental results

5.1 Experimental environment

The performance of SSSTree was tested with several collections of data. First, we used a collection of $100,000$ vectors of dimension 10, synthetically generated, and with Gaussian distribution. The Euclidean distance was used as the distance function when working with this collection. In addition to this synthetic vector space, we also worked with real metric spaces. The first one is a collection of $86,061$ words taken from the Spanish dictionary, using the edit distance as the distance function. The second one is a collection of $40,700$ images taken from the NASA image and video archives. Each image is represented by a feature vector of 20 components obtained from the color histogram of the image. The Euclidean distance was used with this collection to measure the similarity between two images. The algorithm was compared with other well-known clustering-based indexing methods: M-Tree [18], GNAT [14] and EGNAT [16].

5.2 Search efficiency

The first experiment consisted in the evaluation of the search efficiency obtained with SSSTree compared with other similar methods. Figure 6 shows the results obtained with the collection of vectors, expressed as the number of evaluations of the distance function (average of $10,000$ queries), in terms of the percentage of the database retrieved in each query (the higher this percentage, the more difficult the search). As we can see in this figure, SSSTree obtains better results than the other methods.

Figures 7 and 8 show that SSSTree is also more efficient in the collection of words taken from the Spanish dictionary and the collection of images from

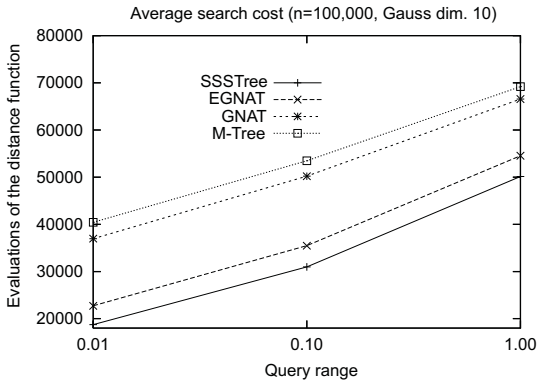


Fig. 6. Evaluations of the distance function with M-Tree, GNAT, EGNAT and SSSTree in a collection of vectors of dimension 10 and Gaussian distribution.

NASA archives, which both are real metric spaces. In the case of the collection of words, the performance of the algorithms was compared for different query ranges, since the higher the range the more difficult the search. These results obtained with both synthetic and real metric spaces show that SSSTree is more efficient than previously proposed methods.

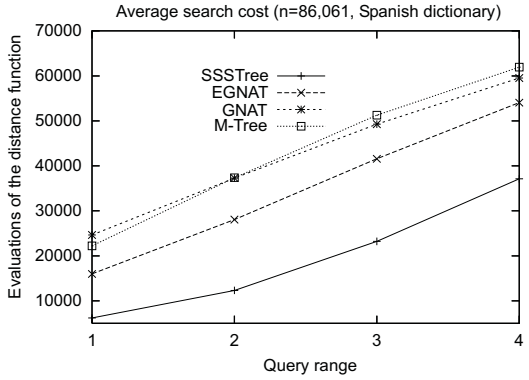


Fig. 7. Evaluations of the distance function with M-Tree, GNAT, EGNAT, and SSSTree, in a collection of words taken from the Spanish dictionary.

6 Conclusions and future work

In this paper we present a new method for similarity search in metric spaces called SSSTree. Its main characteristic is that the cluster centers are selected

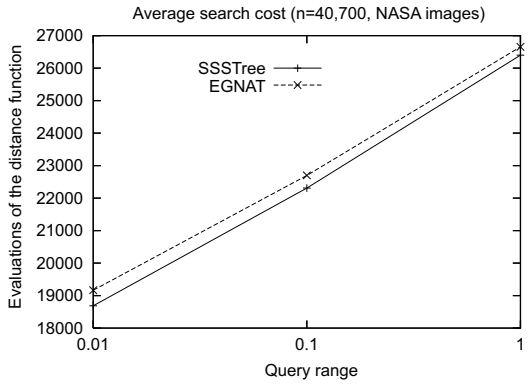


Fig. 8. Evaluations of the distance function with EGNAT and SSSTree, in a collection of images from NASA archives.

applying *Sparse Spatial Selection* (SSS), a selection strategy initially developed for pivot selection. SSS is an adaptive strategy that selects well distributed reference points to improve the search performance. These two properties are also present in SSSTree. The adaptive cluster center selection makes each node of the tree to partition its corresponding subspace as needed in terms of its complexity, a very important difference with previously proposed methods.

The paper also presents experimental results with vector spaces, a collection of words and a collection of images, that show the efficiency of SSSTree against other methods. The obtained improvement is obtained due to the fact that the cluster centers are well distributed in the space and only the clusters needed to cover the complexity of the metric space are created. The space partition is more efficient and this is reflected in the search efficiency.

Our work line still maintains some open questions for future work. First, we are testing the performance of SSSTree with other real metric spaces, as collections of text documents or images. We are also studying the stop condition for the construction process in terms of the covering radius instead of the number of objects contained in the cluster. We are also working in experiments with nested metric spaces [15].

Acknowledgments

This work has been partially supported by: For N. Brisaboa, O. Pedreira and D. Seco by "Ministerio de Educación y Ciencia" (PGE and FEDER) refs. TIN2006-16071-C03-03 and (Programa FPU) AP-2006-03214 (for O. Pedreira), and "Xunta de Galicia" refs. PGIDIT05SIN10502PR and 2006/4. For R. Uribe by Fondecyt 1060776, Conicyt PR-F1-002IC-06, Universidad de Magallanes, Chile y CYT-EDGRID Proyecto 505PI0058.

References

1. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Computing Surveys **33** (2001) 273–321
2. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity search. The metric space approach. Volume 32. Springer (2006)
3. Burkhard, W.A., Keller, R.M.: Some approaches to best-match file searching. Communications of the ACM **16** (1973) 230–236
4. Baeza-Yates, R., Cunto, W., Manber, U., Wu, S.: Proximity matching using fixed-queries trees. In: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching. Volume LNCS(807). (1994) 198–212
5. Baeza-Yates, R.: Searching: an algorithmic tour. Encyclopedia of Computer Science and Technology **37** (1997) 331–359
6. Chávez, E., Marroquín, J.L., Navarro, G.: Overcoming the curse of dimensionality. In: European Workshop on Content-based Multimedia Indexing (CBMI'99). (1999) 57–64
7. Yianilos, P.: Data structures and algorithms for nearest-neighbor search in general metric space. In: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete Algorithms. (1993) 311–321
8. Bozkaya, T., Ozsoyoglu, M.: Distance-based indexing for high-dimensional metric spaces. In: Proceedings of the ACM International Conference on Management of Data (SIGMOD 1997). (1997) 357–368
9. Yianilos, P.: Excluded middle vantage point forests for nearest neighbor search. In: Proceedings of the 6th DIMACS Implementation Challenge: Near neighbor searches (ALENEX 1999). (1999)
10. Vidal, E.: An algorithm for finding nearest neighbors in (aproximately) constant average time. Pattern Recognition Letters **4** (1986) 145–157
11. Micó, L., Oncina, J., Vidal, R.E.: A new version of the nearest-neighbor approximating and eliminating search (aesa) with linear pre-processing time and memory requirements. Pattern Recognition Letters **15** (1994) 9–17
12. Kalantari, I., McDonald, G.: A data structure and an algorithm for the nearest point problem. IEEE Transactions on Software Engineering **9** (1983) 631–634
13. Uhlmann, J.K.: Satisfying general proximity/similarity queries with metric trees. Information Processing Letters **40** (1991) 175–179
14. Brin, S.: Near neighbor search in large metric spaces. In: 21st conference on Very Large Databases. (1995)
15. Brisaboa, N.R., Pedreira, O.: Spatial selection of sparse pivots for similarity search in metric spaces. In: SOFSEM 2007: 33rd Conference on Current Trends in Theory and Practice of Computer Science. LNCS (4362) (2007) 434–445
16. Uribe, R., Navarro, G., Barrientos, R.J., Marín, M.: An index data structure for searching in metric space databases. In: Proc. of International Conference on Computational Science 2006 (ICC 2006). Volume 3991 of Lecture Notes in Computer Science., Springer (2006) 611–617
17. Bustos, B., Navarro, G., Chávez, E.: Pivot selection techniques for proximity searching in metric spaces. Pattern Recognition Letters **24**(14) (2003) 2357–2366 Elsevier.
18. 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 Very Large Data Bases (VLDB'97). (1997) 426–435