Indexing Dense Nested Metric Spaces for Efficient Similarity Search[⋆]

Nieves R. Brisaboa, Miguel R. Luaces, Oscar Pedreira,
Ángeles S. Places, and Diego Seco

Database Lab., Universidade da Coruña
Facultade de Informática, Campus de Elviña s/n, 15071 A Coruña, Spain
{brisaboa,mluaces,opedreira,asplaces,dseco}@udc.es

Abstract. Searching in metric spaces is a very active field since it offers methods for indexing and searching by similarity in collections of unstructured data. These methods select some objects of the collection as reference objects to build the indexes. It has been shown that the way the references are selected affects the search performance, and several algorithms for good reference selection have been proposed. Most of them assume the space to have a reasonably regular distribution. However, in some spaces the objects are grouped in small dense clusters that can make these methods perform worse than a random selection. In this paper, we propose a new method able to detect these situations and adapt the structure of the index to them. Our experimental evaluation shows that our proposal is more efficient than previous approaches when using the same amount of memory.

1 Introduction

Similarity search has become a necessary operation for a large number of applications that deal with data without a semantically clear structure. For instance, multimedia databases manage unstructured objects as images, sound, or video. Content-based retrieval of the most similar images to another one given as a query is an example of similarity search. Applications related to strings and documents are present in systems that range from text editors to big search engines: finding words similar to another one for spelling correction, near-duplicate detection of documents, query rewriting, or spam detection are some examples. We can find more applications in areas such as computational biology (retrieval of DNA or protein sequences), or pattern recognition (where a pattern can be classified from similar, previously classified patterns) [1, 2].

Similarity search can be formalized through the concept of metric space. A *metric space* is a pair (X, d) composed of a *universe* of objects X and a *metric*

⋆ This work has been partially supported by "Ministerio de Educación y Ciencia" (PGE y FEDER) ref. TIN2006-16071-C03-03 and by "Xunta de Galicia" ref. PGIDIT05SIN10502PR., and by "Dirección Xeral de Ordenación e Calidade do Sistema Universitario de Galicia, da Consellería de Educación e Ordenación Universitaria-Xunta de Galicia" for Diego Seco.

$d : X \times X \longrightarrow \mathbb{R}^+$, a function that satisfies the properties of *strictly positiveness* ($d(x, y) \geq 0$, and $d(x, y) = 0 \Leftrightarrow x = y$), *symmetry* ($d(x, y) = d(y, x)$), and the *triangle inequality* ($d(x, y) \leq d(x, z) + d(z, y)$). The value $d(x, y)$ is called the distance from x to y with respect to the metric d, and it represents the dissimilarity between them [3]. The *database* or *collection* of objects where the searches are carried out is represented by a finite subset $U \subseteq X$ of size $n = |U|$.

A *query* is expressed as a query object $q \in X$ and a criterion of similarity on that object. The *result set* is the subset of objects of the collection that satisfy that criterion of proximity. There are two main queries of interest in metric spaces: *(i) range search* retrieves all the objects of the collection up to a certain distance r to the query object q; *(ii) k-nearest neighbors search* retrieves the k most similar objects to the query. Range search is the most important, since it is the more general and other queries can be implemented in terms of it [1].

Vector spaces are a particular case of metric spaces, where each object is a tuple of real numbers. In this case, we can use any of the metrics of the L_p family, defined for $\mathbb{R}^l$ as $L_p(x, y) = (\sum_{1 \leq i \leq l} |x_i - y_i|^p)^{1/p}$. For instance, L_1 is the Manhattan distance, L_2 the Euclidean distance (the usual choice), and $L_\infty = max_{1 \leq i \leq l} |x_i - y_i|$ is the maximum distance. The set of all the strings of a given alphabet with the edit distance (computed as the number of symbols we need to add, remove, or replace to transform one string into another) is another example of a metric space. The dictionaries of different languages are possible databases, and we could be interested in retrieving all the words up to a certain distance of the query to correct spelling errors.

The naive way of implementing similarity search consists in sequentially comparing the query object with all the objects of the database. However, this solution is not feasible in practice since the comparison of two objects is supposed to involve a high computational cost and the database may have a large number of objects. This has motivated the development of methods for indexing and searching that make this operation more efficient by trying to reduce the number of comparisons of objects needed to solve a query. This can be achieved by building indexes that store information that, during the search, permits to directly discard a significant amount of objects from the result set without directly comparing them with the query object.

Although reducing the number of evaluations of the distance function is the main goal of these methods, there are other issues to take in account. Some methods can only work with discrete distance functions, while others can work with continuous distances too. This constraint limits the range of problems in which the algorithm can be applied. Processing the information stored in the index for solving a query involves an extra CPU time that also affects the overall search performance. The possibility of efficiently storing the index in secondary memory and the number I/O operations needed to process it is other important aspect.

Methods for searching in metric spaces use a set of objects from the collection as reference points that are used to obtain the information stored in the index to speed-up the search. It has been shown that the specific set of reference objects affects the final search cost [4, 12] and several techniques have been proposed for their effective selection. These references are selected without caring about the topology of the space, assuming that it is reasonably regular. However, in some cases the spaces present irregularities that may cause these techniques to perform worse than even a random selection.

In this paper we propose *Sparse Spatial Selection for Nested Metric Spaces* (SSS-NMS), a new method for searching in metric spaces that adapts the index structure to the distribution of the collection. The rest of the paper is organized as follows: Section 2 introduces some concepts and related work we use in this paper. Section 3 presents SSS-NMS. In Section 4 we describe the experimental evaluation we carried out and the results we obtained. Finally, in Section 5 we present the conclusions of this work and possible lines of future work.

2 Background and Related Work

Methods for searching in metric spaces can be grouped in two classes [1]: *pivot-based* methods, and *clustering-based* methods. In pivot-based methods, the index is a data structure that stores precomputed distances from the objects of the database to a subset of objects used as pivots. When given a query (q, r), the query object is compared with the pivots. For every $x \in U$ and every pivot $p \in U$, we know (by the triangle inequality) that $d(p, x) \leq d(p, q) + d(q, x)$, and therefore $d(q, x) \geq |d(p, x) - d(p, q)|$. If $|d(p, x) - d(p, q)| > r$, then $d(q, x) > r$ and x can be discarded from the result set without comparing it with the query. The simplest index consists in a table storing the distances from all the objects of the database to all the pivots. Well-known pivot-based methods are BKT [5], FQT [6], VPT [7], AESA [8] and LAESA [9].

Clustering-based methods partition the metric space in a set of clusters, each of them represented by a cluster center $c \in U$ and the covering radius $r_c \in \mathbb{R}^+$: the distance from the center to its furthest object in the cluster. When given a query (q, r), the query object is compared with the cluster centers. For each cluster (c, r_c), the whole cluster can be directly discarded if $d(q, c) - r_c > r$, since in this case, the intersection of the cluster and the result set is empty. Well-known clustering-based methods are BST [10], GHT [11], GNAT [12] and M-Tree [13].

In both cases, the objects that could not be discarded make up the candidate list and have to be directly compared with the query. The search complexity is given by the sum of the comparisons of the query with pivots or centers (internal complexity) and the comparisons with candidate objects (external complexity).

Selection of Reference Objects

Both pivot-based and clustering-based methods use some objects of the collection as references for building the index: pivots in the case of pivot-based methods and cluster centers in the case of clustering-based methods.

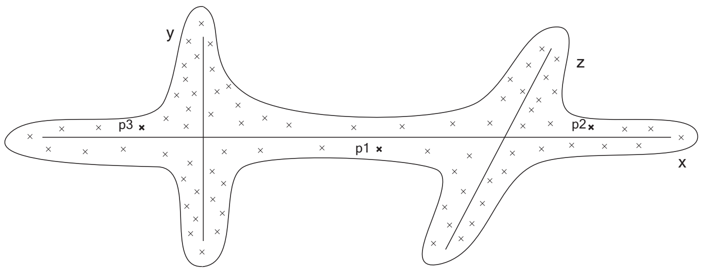

Fig. 1. Dense subspaces nested in a general metric space.

It has been shown that the specific set of objects used as references affects the search performance of the method [4, 12]. The number of references, and their location with respect to other references and to the objects of the database determine the effectiveness of the index for discarding objects. Several techniques have been proposed for selecting references. [9, 12, 7] proposed different techniques for selecting references far away from each other. [4] formally analyzed the problem of reference selection, and proposed a criterion for comparing the effectiveness of two sets of pivots and several techniques based on the iterative optimization of that criterion. [14–16] are also based on defining a criterion for measuring the effectiveness of the set of reference objects and select the references by trying to optimize that criterion.

Sparse Spatial Selection (SSS) [17] selects a set of pivots well distributed in the space. In contrast to previous techniques, SSS is dynamic, this is, the database is initially empty and new references are selected as needed when new objects are inserted in the database. When a new object is inserted, it is selected as a reference if its distance to the other references is greater or equal than $M\alpha$, being M the maximum distance between any two objects, and α a constant that usually takes values around 0.4. The object becomes a reference if it is far enough from the already selected references.

[17] shows that the optimal values of α are in the range $[0.35, 0.40]$ and the search cost is virtually the same for all the values in that interval. Other important feature of SSS is that it is not necessary to specify the number of references to select. SSS selects more references as needed when the database grows, adapting the number of objects to the complexity of the collection. Although it was originally proposed for selecting pivots, it can be applied for selecting cluster centers without changes.

Nested Metric Spaces

Most methods for searching in metric spaces assume that the topology of the collection of objects is reasonably regular. However, some spaces present irregularities that can affect their behavior and degrade the search performance they achieve.

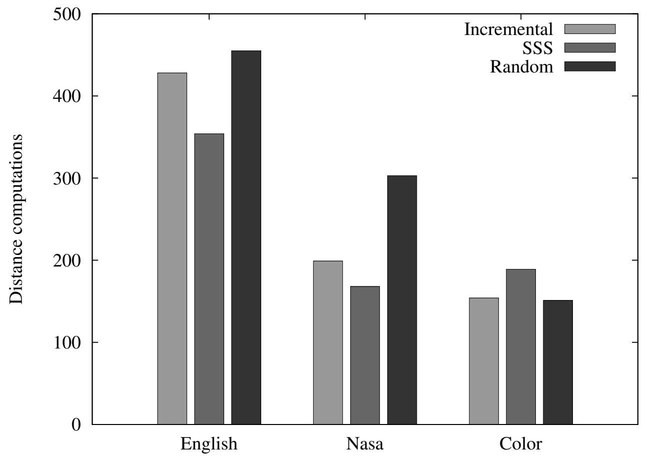

Fig. 2. Average search cost for the collections for English, Nasa, and Color, with Incremental, SSS, and random pivot selection.

In many applications the objects of the collection are grouped in different clusters or subspaces, in such a way that different dimensions or variables explain the differences between objects inside each subspace.

Figure 1 shows an example of this situation. The database is a subset of $\mathbb{R}^3$. This space has three explicit dimensions: the main corresponds to the x axis, and the other two correspond to the y and z axis. In this example, there are two subspaces with a large number of objects along the axes y and z. The objects inside a subspace are almost equal according to the main dimension, but different according to the specific dimensions of the subspace they belong to. In this example, the maximum distance between two objects is given by the main dimension. If working with SSS, after selecting the pivots p_1, p_2, and p_3, no more pivots could be selected. However, a random pivot selection has the chance of putting pivots inside each subspace since they have a large number of objects.

Figure 2 shows the average search cost for solving a query in the collections English, Nasa, and Color (the details of each collection are described in Section 4). SSS gets the best result in English and Nasa. However, in Color it is worse than even a random selection. Most the coordinates of the feature vectors of Color take the value 0 or are very close to 0, so a large number of objects is grouped near the origin of coordinates. The irregularity of this collection makes SSS to obtain this result.

3 Sparse Spatial Selection for Nested Metric Spaces

Sparse Spatial Selection for Nested Metric Spaces (SSS-NMS) is a new method for searching in metric spaces that identifies dense clusters of objects in the collection and adapts the structure of the index to this situation.

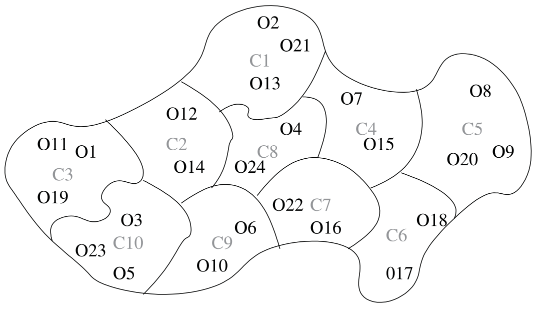

Fig. 3. At the first level, the space is partitioned using reference objects selected with SSS as cluster centers.

The index built by SSS-NMS is structured in two levels. In a first level, SSS-NMS selects a set of reference objects with SSS and uses them as cluster centers to create a Voronoi partition of the space. In a second phase, those clusters considered dense are further indexed by applying SSS in each of them. Following this procedure, SSS-NMS is able to detect complex groups of objects in the database and to index them according to their complexity.

3.1 Construction

The construction of the index is carried out in two phases. Since SSS is used in each of them, we will call α and β the constants that control the selection of reference objects with SSS in each phase respectively.

Voronoi Partition of the Space with SSS

In the first phase SSS is applied to obtain a set of reference objects used as cluster centers $\{c_1, \ldots, c_m\}$. To create a Voronoi partition of the collection, each object is assigned to the cluster corresponding to its nearest cluster center: $C_i = \{x \in U / \forall j \neq i, d(x, c_j) \geq d(x, c_i)\}$, where C_i is the cluster associated to the center c_i (see Figure 3).

The value of α should be small in this first phase. Having few cluster centers could result in dense clusters that contain also objects that do not belong to the real dense cluster of objects. However, a small value of α can also result in empty clusters. Those empty clusters are removed, and their corresponding centers are added to the rest of clusters as any other object of the collection.

Indexing Dense Clusters with SSS

In the second phase, those clusters considered dense are further indexed using a set of references obtained with SSS as pivots. We compute the density of each cluster as the number of elements in the cluster divided by the maximum distance between them:

$$density(C_i) = \frac{|C_i|}{max\{d(x,y)/x, y \in C_i\}}$$

Computing the density of all clusters could be very costly if the maximum distance of each of them is obtained by comparing all the objects of the cluster with all the rest of objects. To avoid this overhead in the construction time, we obtain an approximation of the maximum distance. To do this, an object of the cluster is picked at random and compared with the rest of objects of the cluster. Its furthest object is then compared with all the objects of the cluster, to obtain it furthest object too. After repeating the process for a few iterations, a good approximation of the maximum distance (if not the actual maximum distance) is obtained. This approximation is also used when applying SSS in the first level.

We consider that the cluster C_i has a high density if $density(C_i) > \mu + 2\sigma$, where μ and σ are the mean and the standard deviation of the densities of all clusters. For each dense cluster, a set of objects is obtained with SSS to be used as pivots, and the table of distances from all the objects of the cluster to the pivots is computed and stored. In this second phase the index stores more information for the dense complex subspaces. In this case, the value of β should be around 0.4, as indicated in [17].

3.2 Search

Given a query (q, r), the query object is compared with all the cluster centers of the first level. Those clusters $C_i = (c_i, r_c)$ for which $d(q, c_i) - r_c > r$ are directly discarded from the result set, since the intersection of the cluster and the result set is empty. For the clusters that could not be discarded there are two possibilities. If the cluster does not have associated a table of distances from its objects to pivots, the query has to be directly compared with all the objects of the cluster. If the cluster has associated a table of distances, the query is compared with the pivots and the table is processed to discard as many objects as possible, as described in 2. The objects that can not be discarded are directly compared with the query.

4 Experimental Evaluation

Test Environment

For the experimental evaluation of our method we used three collections of real data available in the Metric Spaces Library [18]: English is a collection of 69,069 words extracted from the English dictionary, and compared using the standard edit distance; Nasa is a collection of 40,150 images extracted from the archives of image and video of the NASA, and represented by feature vectors of dimension 20; Color is a collection of $112,544$ color images represented by feature vectors of dimension 112. Both in Nasa and Color the images are compared using the Euclidean distance of their feature vectors. As usual, the experimental evaluation focused in range search, using the number of distance computations as the measure of the search cost.

Construction and Search

Our initial hypothesis was that, when working with real collections, we can not assume a regular distribution of the objects in the space, and that they are usually grouped in dense subspaces nested into a more general metric space. With these experiments we also obtained the densities of the clusters obtained in the first phase of the construction. Figures 4 and 5 show the histograms of cluster densities for the collections Nasa and Color. As we can see, most of the clusters have more or less the same density, although some of them have a much higher density. The number of clusters with a too high density is small, and these are the ones indexed in the second level of the index. The results are similar for English, and for other values of α (we do not include the graphics for reasons of space).

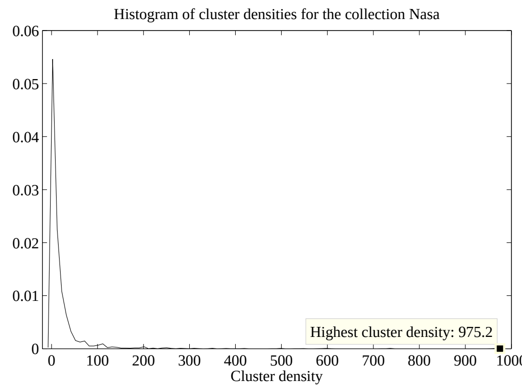

Fig. 4. Histogram of clusters density for Nasa.

The construction of SSS-NMS depends on the parameters α and β, that control the number of reference objects selected by SSS in the first and second levels of the index respectively. Since these parameters affect the structure of the index and the information it stores, they also affect the search performance. In the previous section we indicated that the value of α should be small for selecting a significant number of cluster centers in the first level, and that the value of β should be around 0.4, as indicated in [17].

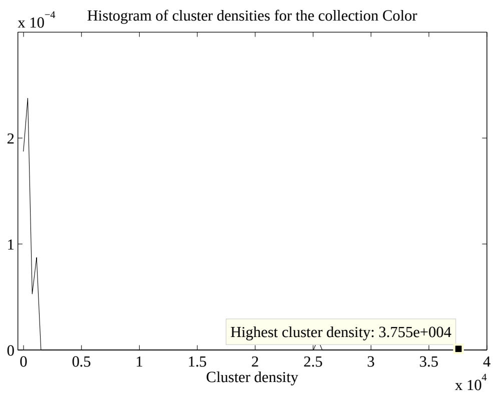

Fig. 5. Histogram of clusters density for Color.

In our first set of experiments we indexed the three collections for values of α between 0.1 and 0.4, and values of β between 0.3 and 0.6. In each case, the 90% of objects of the database were indexed, and the other 10% were used as queries. The search radius for English was $r = 2$, and for Nasa and Color we used search radius that retrieve a 0.01% of the objects of the collection. Figure 6, 7, and 8 show the results we obtained. As we can see in these results, the optimal search cost is obtained for $\alpha = 0.2$ and $\beta = 0.5$. Although in Nasa the best result is obtained for $\alpha = 0.4$, the difference with the result obtained for $\alpha = 0.2$ is small. These results are consistent with [17], since the optimal β for each space is always around 0.4, and the search cost for all those values is virtually the same.

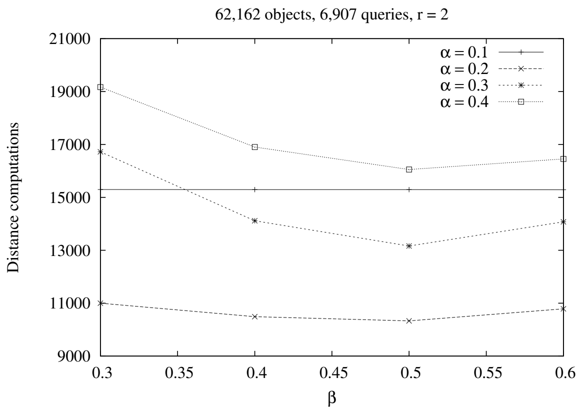

Fig. 6. Search cost in terms of α and β for English.

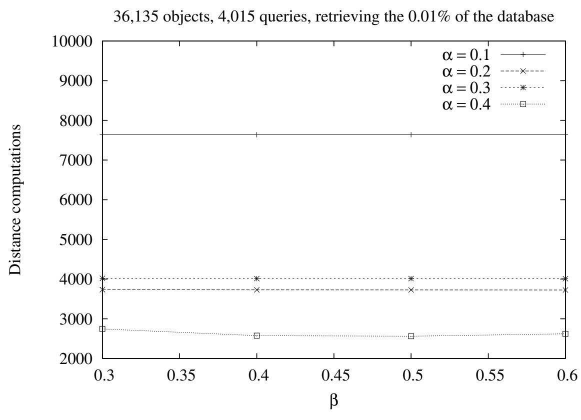

Fig. 7. Search cost in terms of α and β for Nasa.

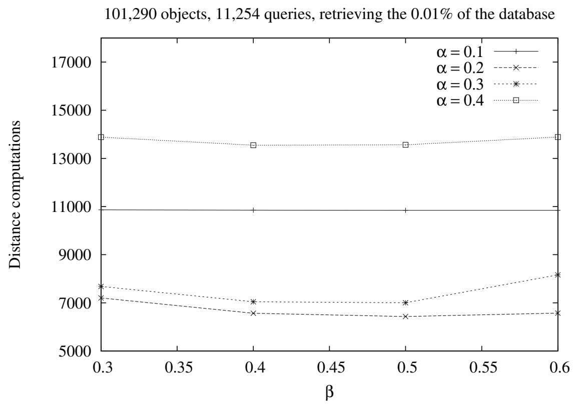

Fig. 8. Search cost in terms of α and β for Color.

Comparison

We evaluated the search efficiency of SSS-NMS by comparing it with existing state-of-art methods. Particularly, we compared it with SSS [17], Incremental [4] and LAESA with random pivots [9]. All of them are pivot-based methods. Comparing SSS-NMS with clustering-based methods would not make sense, since it uses a cluster approach at a first level, to which it adds more information in the second level.

Pivot-based algorithms achieve better results as more space for storing the index is given to them. However, this is only possible for small collections, since in large collections their optimal result can require an index of several gigabytes. Thus, the comparison was carried out configuring the methods to use the same amount of space for storing the index.

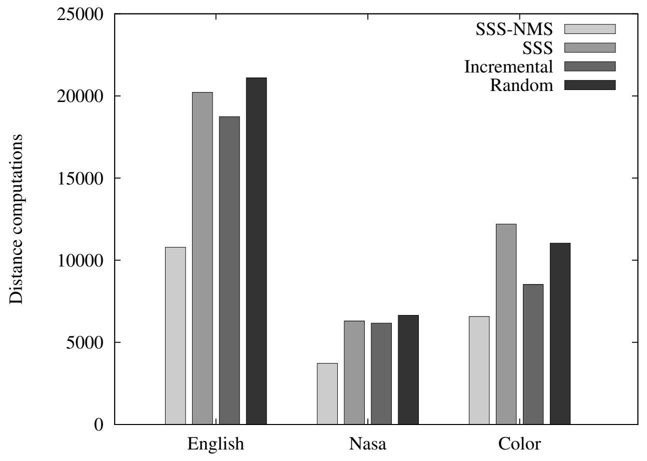

Fig. 9. Comparison of SSS-NMS with LAESA with random pivots and SSS pivots.

Again, 90% of the objects of each collection were indexed and 10% were used as queries, retrieving an average of 0.01% of objects of the database for each query in the case of Nasa and Color, and using a search radius $r = 2$ for English.

Figure 9 shows the results we obtained. For each collection and method we show the average distance computations needed for solving a query. These results show that SSS-NMS is more efficient in terms of distance computations than the other methods when using the same amount of space.

5 Conclusions

In this paper we propose a new method for searching in metric spaces called *Sparse Spatial Selection for Nested Metric Spaces* (SSS-NMS). The main feature of SSS-NMS, is that it assumes that the data does not necessarily have a regular distribution, and it adapts the index structure to the actual distribution of the space. SSS-NMS indexes the dataset in two levels: the first one creates a Voronoi partition of the Space using SSS; in the second level, those clusters considered to have a high density of objects are further indexed with pivots selected with SSS too. Thus, SSS-NMS obtains more information for the complex regions of the space.

The paper also presents experimental results with collections of words and images from the Metric Spaces Library [18], that show the efficiency of SSS-NMS against other methods.

There are still some open questions for future work. As in pivot-based methods is possible to gain efficiency by adding more space, we are studying ways of making possible SSS-NMS to use more space and thus be even more efficient when the characteristics of the application allow that use of additional memory.

References

1. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Computing Surveys **33**(3) (September 2001) 273–321 ACM Press.
2. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity search. The metric space approach. Volume 32 of Advances in Database Systems. Springer (2006)
3. Searcóid, M.O.: Metric Spaces. Springer Undergraduate Mathematics Series. Springer (2007)
4. Bustos, B., Navarro, G., Chávez, E.: Pivot selection techniques for proximity searching in metric spaces. Pattern Recognition Letters **24**(14) (2003) 2357–2366 Elsevier.
5. Burkhard, W.A., Keller, R.M.: Some approaches to best-match file searching. Communications of the ACM **16**(4) (April 1973) 230–236 ACM Press.
6. Baeza-Yates, R., Cunto, W., Manber, U., Wu, S.: Proximity matching using fixed-queries trees. In: Proc. of the 5th Annual Symposium on Combinatorial Pattern Matching (CPM'94), Asilomar, C.A., Springer-Verlag (1994) 198–212
7. Yianilos, P.: Data structures and algorithms for nearest-neighbor search in general metric spaces. In: Proc. of the fourth annual ACM-SIAM Symposium on Discrete Algorithms (SODA'93), ACM Press (1993) 311–321
8. Vidal, E.: An algorithm for finding nearest neighbors in (approximately) constant average time. Pattern Recognition Letters **4** (1986) 145–157 Elsevier.
9. 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 Elsevier.
10. Kalantari, I., McDonald, G.: A data structure and an algorithm for the nearest point problem. IEEE Transactions on Software Engineering **9** (1983) 631–634 IEEE Press.
11. Uhlmann, J.K.: Satisfying general proximity/similarity queries with metric trees. Information Processing Letters **40** (1991) 175–179 Elsevier.
12. Brin, S.: Near neighbor search in large metric spaces. In: Proc. of 21st conference on Very Large Databases (VLDB'95), ACM Press (1995)
13. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Proc. of the 23rd International Conference on Very Large Data Bases (VLDB'97), Athens, Greece, ACM Press (1997) 426–435
14. Vleugels, J., Veltkamp, R.C.: Efficient image retrieval through vantage objects. Pattern Recognition **35**(1) (2002) 69–80 Elsevier.
15. van Leuken, R.H., Veltkamp, R.C., Typke, R.: Selecting vantage objects for similarity indexing. In: Proc. of the 18th International Conference on Pattern Recognition (ICPR 2006), IEEE Press (2006) 453–456
16. Venkateswaran, J., Kahveci, T., Jermaine, C.M., Lachwani, D.: Reference-based indexing for metric spaces with costly distance measures. The VLDB Journal **17**(5) (2008) 1231–1251 Springer.
17. Brisaboa, N.R., Fariña, A., Pedreira, O., Reyes, N.: Similarity search using sparse pivots for efficient multimedia information retrieval. In: Proc. of the 8th IEEE International Symposium on Multimedia (ISM'06), San Diego, California, USA, IEEE Press (2006) 881–888
18. SISAP: Metric spaces library (`http://sisap.org/metric_space_library.html`)