Similarity Search Using Sparse Pivots for Efficient Multimedia Information Retrieval *

Nieves R. Brisaboa, Antonio Fariña, Oscar Pedreira
University of A Coruña
Database Laboratory
Campus de Elviña, 15071 A Coruña, Spain
{*brisaboa, fari, opedreira*}*@udc.es*

Nora Reyes
Universidad Nacional de San Luis
Departamento de Informatica
Ej. de los Andes 950, San Luis, Argentina
nreyes@unsl.edu.ar

Abstract

*Similarity search is a fundamental operation for applications that deal with unstructured data sources. In this paper we propose a new pivot-based method for similarity search, called Sparse Spatial Selection (*SSS*). This method guarantees a good pivot selection more efficiently than other methods previously proposed. In addition,* SSS *adapts itself to the dimensionality of the metric space we are working with, and it is not necessary to specify in advance the number of pivots to extract. Furthermore,* SSS *is dynamic, it supports object insertions in the database efficiently, it can work with both continuous and discrete distance functions, and it is suitable for secondary memory storage. In this work we provide experimental results that confirm the advantages of the method with several vector and metric spaces.*

1 Introduction

Similarity search has become a very important operation in applications that deal with unstructured data sources. For example, multimedia databases manage objects without any kind of structure, such as images, fingerprints or audio clips. 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 simple text editors (finding words similar to a given one to correct edition errors) to big search engines (retrieving relevant documents for a given query). We can find more examples in areas such as computational biology (retrieval of DNA sequences) or pattern recognition (where a pattern can be classified from similar patterns previously classified). The computational cost of the algorithms that determine the similarity between two objects makes similarity search an expensive operation. This fact has motivated the development of many research works aiming to do efficient similarity search over large collections of data.

The similarity search problem can be formally defined through the concept of *metric space*, which provides a formal framework that is independent of the application domain. A *metric space* $(\mathbb{X}, d)$ is composed of a universe of valid objects $\mathbb{X}$ and a *distance function* $d : \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}^+$ defined among them. The distance function determines the similarity or distance between two given objects. This function holds several properties: strictly positiveness ($d(x, y) > 0$ 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 finite subset $\mathbb{U} \subset \mathbb{X}$ with size $n = |\mathbb{U}|$, is called dictionary or database and represents the collection of objects where searches are performed. A k-dimensional vector space is a particular case of metric space in which every object is represented by a vector of k real coordinates. The definition of the distance function depends on the type of the objects we are managing. In a vector space, d could be a distance function of the family L_s, defined as $L_s(x, y) = (\sum_{1 \leq i \leq k} |x_i - y_i|^s)^{\frac{1}{s}}$. For instance, L_1 is known as *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 same idea that in vector spaces. This is a very interesting concept since the efficiency of the search methods is worse in metric spaces with a high intrinsic dimensionality [7].

There are three main queries of interest for a collection of objects in a metric space: *i) range search*, that retrieves all the objects $u \in \mathbb{U}$ within a radius r of the query q, that is: $\{u \in \mathbb{U} \ / \ d(q, u) \leq r\}$; *ii) nearest neighbor search*,

*This work has been partially supported by CYTED VII.J (RITOS2) and, for the first three authors, by por MCYT (PGE and FEDER) grants TIC2003-06593 and TIN2006-15071-C03-03, and Xunta de Galicia grant PGIDIT05SIN10502PR

that retrieves the most similar object to the query q, that is: $\{u \in \mathbb{U} \ / \ \forall \ v \in \mathbb{U},\ d(q,u) \leq d(q,v)\}$; and *iii) k-nearest neighbors search*, a generalization of the nearest neighbor search, retrieving the set $A \subseteq \mathbb{U}$ such that $|A| = k$ and $\forall\ u \in A,\ v \in \mathbb{U} - A,\ d(q,u) \leq d(q,v)$. The range query is the most used, and the others can be implemented in terms of it [7]. In any case, the distance function is the unique information that can be used in order to perform searches. The naive way of implementing those operations is to compare all the objects in the collection against the query.

The problem is that the evaluation of the distance function is very expensive, and therefore searches become inefficient if the collection has a high number of elements. Thus, reducing the number of evaluations of the distance function is the main goal of the methods for similarity search in metric spaces. To do that, they first build indexes over the whole collection. Later, using the triangle inequality, those indexes permit to discard some elements without being necessary to compare them against the query. The techniques that permit to search metric spaces efficiently differ usually in some features. Some of them allow only discrete distance functions, while others where developed to work with continuous distance functions. This is an important issue that restricts their application in some domains. There are also static methods, where the index has to be build on the whole collection, and dynamic techniques, that allow insertions in the database, or even allow the index to be built as elements are added to an initially empty collection. Other important factor is the possibility of storing the index efficiently into secondary storage, and the number of I/O operations needed to access it. In general, the applicability and efficiency of a method depends on this issues.

Search methods can be classified into two types [7]: *pivot*-based and *clustering*-based techniques. *Pivot-based* search techniques choose a subset of the objects in the collection that are used as *pivots*. The index is build by computing the distances from each pivot to each object in the database. Given a query (q, r), the distances from the query q to each pivot are computed, and then some objects of the collection can be directly discarded using the triangle inequality and the distances precomputed during the index building phase. Being $x \in \mathbb{U}$ an object in the collection, we can discard x if $|d(p_i, x) - d(p_i, q)| > r$ for any pivot p_i, since by the triangle inequality, if this condition is true, its distance to q will be $d(x, q) > r$. The objects that can not be discarded by this condition make up the candidate list, and they must be compared against the query. The *total complexity* of the search is the sum of the *internal complexity*, the comparisons of q with each pivot, and the *external complexity*, the comparisons of q with each object in the candidate list. The most well-known pivot-based methods are: *Burkhard-Keller-Tree* (BKT) [4], *Fixed-Queries Tree* (FQT) [2], *Fixed-Height FQT* (FQHT) [1], *Fixed-Queries Array* (FQA) [6], *Vantage Point Tree* (VPT) [13], *Approximating and Eliminating Search Algorithm* (AESA) [12] and LAESA (*Linear AESA*) [9].

Clustering-based techniques split the metric space into a set of equivalence regions each of them represented by a *cluster center*. During searches, whole regions can be discarded depending on the distance from their cluster center to the query. The most important *clustering*-based techniques are: *Bisector Trees* (BST) [8], *Generalized-Hyperplane Tree* (GHT) [11], *Geometric Near-neighbor Access Tree* (GNAT) [3] and *Spatial Approximation Tree* (SAT) [10]. Two good surveys about metric spaces can be found in [7] and [14].

In this paper we present *Sparse Spatial Selection* (SSS), a new pivot-based technique. SSS is a dynamic method since the collection can be initially empty and/or grow later. It works with continuous distance functions and its suitable for secondary memory storage. The main contribution of SSS is the use of a new pivot selection strategy. This strategy generates a number of pivots that depends on the intrinsic dimensionality of the space (something interesting from both the theoretical and practical points of view). Moreover, this pivot selection strategy is dynamic since it adapts itself the index when new objects are added to the collection.

The rest of the paper is structured as follows: Section 2 describes the pivot-selection problem and its importance for the efficiency of pivot-based methods. Then, SSS, the new method proposed in this work, is shown in Section 3. In Section 4 we present and discuss the experimental results we obtained in the tests. Finally, Section 5 shows our conclusions and future lines of work.

2 Previous work on pivot selection

It is well-known that the efficiency of a similarity search method depends on the set of objects chosen as pivots, the number of pivots and their "location" in the metric space.

Most of the pivot-based search methods choose pivots randomly. Furthermore, there are no guidelines to determine the optimal number of pivots, since this parameter depends on the metric space we are working with. In previous works, some heuristics for pivot selection have been proposed. For example, in [9] pivots are objects maximizing the sum of distances among them. [13] and [3] propose heuristics to obtain pivots far away from each others. In [5] the importance of the pivot selection strategy was studied in depth, showing empirically how it affects to the performance of a technique.

The main contribution in [5] is a criterion to compare the efficiency of two sets of pivots of the same size. A set of k pivots $\{p_1, p_2, \ldots, p_k\}$, $p_i \in \mathbb{U}$, defines a space $\mathbb{P}$ of distance tuples between pivots and objects in $\mathbb{U}$. The mapping of an object $u \in \mathbb{U}$ to $\mathbb{P}$, which will be denoted $[u]$, is car-

ried out as $[u] = (d(u,p_1), d(u,p_2), \ldots, d(u,p_k))$. Defining the metric (distance function) $D_{\{p_1,p_2,\ldots,p_k\}}([x],[y]) = max_{1\leq i\leq k}|d(x,p_i) - d(y,p_i)|$, it follows that $(\mathbb{P}, D)$ is a metric space, which turns out to be $(\mathbb{R}^k, L_\infty)$. Given a range query (q,r), the condition to discard an object $u \in \mathbb{U}$, $|d(p_i,u) - d(p_i,q)| > r$, becomes $D_{\{p_1,p_2,\ldots,p_k\}}([q],[u]) > r$ in this new metric space $(\mathbb{P}, D)$. To achieve a candidate object list as short as possible, the probability of $D_{\{p_1,p_2,\ldots,p_k\}}([q],[u]) > r$ should be as high as possible. One way to do this is to maximize the average of the distance distribution of D, which will be denoted μ_D. Hence, we will say that $\{p_1,p_2,\ldots,p_k\}$ is a better set of pivots than $\{p'_1,p'_2,\ldots,p'_k\}$ if $\mu_{\{p_1,p_2,\ldots,p_k\}} > \mu_{\{p'_1,p'_2,\ldots,p'_k\}}$ [5].

In [5] several selection strategies based on the previous efficiency criterion were proposed: *i) Selection*, that gets N random sets of pivots and finally selects the one maximizing μ_D; *ii) Incremental*, in which the next pivot chosen will be that object such that, after adding it to the current set of pivots, maximizes μ_D; and *iii) Local Optimum*, an iterative strategy that, starting with a random set of pivots, in each step replaces by a new object the current pivot which less contributes to μ_D. Their conclusions show that, in general, good pivots should be far from the others and also from the rest of objects in the database (i.e., they should be *outliers*). Unfortunately, that condition does not always lead to obtaining good pivots.

Determining the optimal number of pivots k is an important problem. It is known that the efficiency of the searches depends on this parameter. Moreover, k can vary greatly for different metric spaces. In [5] a brute-force approach to determine the optimal number of pivots is used, because this value has to be fixed. Results confirm that that number depends greatly on the metric space, and affects the efficiency of the methods.

3 Our proposal: Sparse Spatial Selection

3.1 Pivot selection strategy

Let $(\mathbb{X}, d)$ be a metric space, $\mathbb{U} \subset \mathbb{X}$ an object collection, and M the maximum distance between any pair of objects, $M = max\{\, d(x,y) \,/\, x,y \in \mathbb{X} \,\}$[1]. The set of pivots contains initially only the first object of the collection. Then, for each element $x_i \in \mathbb{U}$, x_i is chosen as a new pivot if its distance to every pivot in the current set of pivots is equal or greater than $M\alpha$, being α a constant parameter that takes values around 0.5. That is, an object in the collection becomes a new pivot if it is located at more than a fraction of the maximum distance with respect to all the current pivots. For example, if $\alpha = 0.5$ an object is chosen if it is located further than a half of the maximum distance from the already selected pivots. The following pseudocode summarizes the pivot selection process:

$PIVOTS \leftarrow \{x_1\}$
for all $x_i \in \mathbb{U}$ **do**
  if $\forall\, p \in PIVOTS,\ d(x_i,p) \geq M\alpha$ **then**
    $PIVOTS \leftarrow PIVOTS \cup \{x_i\}$
  end if
end for

It seems evident that all the selected pivots will not be too close to each other. Forcing the distance between two pivots to be greater or equal than $M\alpha$, we ensure that they are well distributed in the whole space. It is important to take into account that our pivots are not very far away from each others neither very far from the rest of objects in the collection (i.e., they are not *outliers*), but they are well distributed covering the whole space. Our hypothesis is that, being well distributed in the space, when a search is performed our set of pivots will be able to discard more objects than pivots selected with a different strategy.

Being dynamic and adaptive is another good feature of our pivot selection technique. The set of pivots adapts itself automatically to the growing of the database. When a new element x_i is added to the database, it is compared against the pivots already selected and it becomes a new pivot if needed. In this way the number of pivots does not depend on the collection's size but on its intrinsic dimensionality. Actually the collection could be initially empty, which is interesting in practical applications.

3.2 The parameter α

Although in our method it is not necessary to state in advance the number of pivots to use, as in [5], we have to set the value of α. This value determines the number of pivots. However, α must always take values between 0.35 and 0.40, depending on the intrinsic dimensionality of the space. Figure 1 shows the number of evaluations of the distance function in terms of α for vector spaces of dimensionality 8, 10, 12, and 14. In this figure we can see that the best result is always obtained for values of α that range from 0.35 to 0.40, and that the efficiency of the method is virtually the same for all the values of α included in this interval.

This results show some of the main advantages of our proposal. Our pivot selection technique is simpler and more efficient than others previously proposed. In addition, our pivots are not too close to each other, but they are not far away from the rest of the objects of the collection. (i.e., our pivots are not *outliers*). However, we have achieved an

[1] M depends on the definition of the metric space, and can be obtained without processing all the objects in the collection. For example, in a vector space it can be inferred from the maximum/minimum values of each component of a vector $v \in \mathbb{X}$.

efficiency similar to that of the existing techniques without having to state in advance the number of pivots to use. Our method finds itself the appropriate number of pivots for the intrinsic dimensionality of the metric space, using only the maximum distance between any pair of objects in the collection and the parameter α.

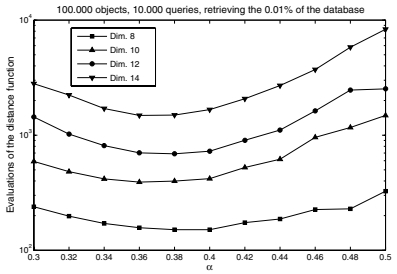


Figure 1. Efficiency in vector spaces.

3.3 Index construction

One of the main advantages of our method is its dynamic nature. Thus, we describe the index construction process assuming that the collection is initially empty. The first object inserted in the database, u_1, becomes the first selected pivot, p_1. When a new object is inserted in the database, its distance to all the pivots already selected is computed and stored. If its distance to all of them is equal or greater than $M\alpha$, the object is added to the set of pivots. In this case, its distance to every object in the database is computed and stored in the index structure. Thus, the number of pivots does not have to be stated in advance over an initial object collection, but it grows at the same time as the intrinsic dimensionality of the collection does. The building of the index is completely dynamic, and the set of pivots adapts appropriately to the new inserted objects. Furthermore, it guarantees that even though the collection grows, the pivots will be well distributed over the metric space.

Since our method needs only to store the distances between the pivots and the objects in the collection, we can use storage structures suitable for secondary memory. For example, a naive way to store in secondary memory the distances is the use B-trees. We could create a B-tree for each pivot to store its distances to all the objects in the database. Thus, when a new pivot is added to the structure, a new B-tree is created for it. This implementation is suitable for the dynamic nature of the method.

4 Experimental results

First we have used synthetic sets of random points in vector spaces of dimensionalities $k = 8$, 10, 12, and 14. We also have tested the algorithm with real metric spaces: collections of words taken from the English and Spanish dictionaries and a set of images from the NASA archives.

4.1 Number of pivots generated in terms of the dimensionality

In Section 3 we emphasized that our method dynamically generates a number of pivots that depends on the dimensionality of the space, and not on the number of elements in the database. To validate this hypothesis we have used collections of $1,000,000$ of vectors of dimensions $k = 8$, 10, 12, and 14. For each vector space we obtained the number of pivots selected when a given number of objects were inserted in the collection, with α fixed to 0.5.

k	n, collection size ($\times 10^3$)					
	100	200	400	600	800	1000
8	16	17	20	22	22	22
10	20	24	29	30	30	30
12	44	50	54	57	58	58
14	56	62	71	79	80	82

Table 1. Number of pivots in vector spaces of dimensionality $k = 8$, 10, 12, and 14.

Table 1 shows the results obtained in this experiments. The number of objects selected as pivots increases as the dimensionality of the vector space does. This result shows that the number of pivots depends on the intrinsic dimensionality of the metric space. In all the test spaces the number of pivots grows quickly with the first objects of the database. Then that number grows much more slowly until it becomes stable. Obviously, when the collection has few elements, the number of pivots depends on its size. However, when the collection reaches a given size no more pivots will be selected even if new objects are inserted in the database. This happens because the current set of pivots covers all the space and captures its dimensionality. With this results we can conclude that the number of pivots generated depends on the intrinsic dimensionality of the space, and not on the size of the collection.

4.2 Search efficiency in vector spaces

In this section we show the results obtained in the tests performed to evaluate the efficiency of the algorithm in the search operation. In the first set of tests we used vector spaces of dimensions $k = 8$, 10, 12, and 14, each of them with $100,000$ vectors uniformly distributed in an hypercube of side 1. We got the mean number of evaluations of the distance function over $10,000$ queries. The mean number of elements retrieved in each of them is the 0.01% of the

database. In order to evaluate the behavior of the algorithm, we compared the results with those obtained with the pivot selection techniques proposed in [5].

Method	$k=8$		$k=10$		$k=12$		$k=14$	
	$\#p$	$\#d$	$\#p$	$\#d$	$\#p$	$\#d$	$\#p$	$\#d$
Random	85	213	190	468	460	998	1000	2077
Select.	85	204	200	446	360	986	800	2038
Incr.	65	157	150	335	300	714	600	1458
Loc. A	70	155	150	333	300	708	600	1448
Loc. B	60	157	150	369	300	881	760	1930
SSS	57	151	148	389	258	689	598	1452

Table 2. Efficiency with different pivot selection strategies in vector spaces

Table 2 shows the minimum number of evaluations of d we have obtained with each pivot selection strategy ($\#d$), and the number of pivots used ($\#p$). We can observe that the number of evaluations of the distance function obtained with our method is always around the best result obtained with the strategies proposed in [5]. In some cases we performed less evaluations of d using less pivots. However, in other cases our method performs more evaluations, although we used less pivots too. This results show that the proposed pivot selection strategy has an efficiency similar to that of other more complex ones. In the results of our tests we can also see that the number of pivots that our method generates is very similar to the optimum number of pivots of other pivot selection techniques.

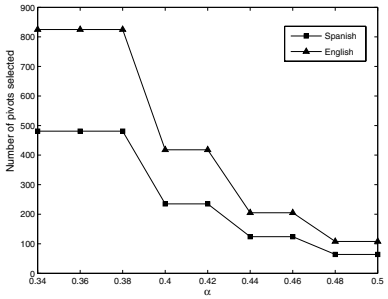
4.3 Search efficiency in metric spaces

In addition to the tests with uniformly distributed vector spaces, we have also compared our method with others using "real" metric spaces. First we used collections of words. The first one contains $69,069$ words taken from the English dictionary, and the second contains $51,589$ words from the Spanish dictionary. We have used the edit distance as the distance function. We used a 10% of the database as queries and a query range $r = 2$, that retrieves around the 0.02% of the collection. Table 3 shows the minimum number of evaluations of d we have reached with our pivot selection technique and the ones proposed in [5], for the collection of words taken from the English dictionary. In this case, the result obtained with our technique its better than the obtained with any other one. As happened with vector spaces, the number of pivots that our method generates is similar to the optimum number of pivots used by other strategies, that have got this number by trial and error.

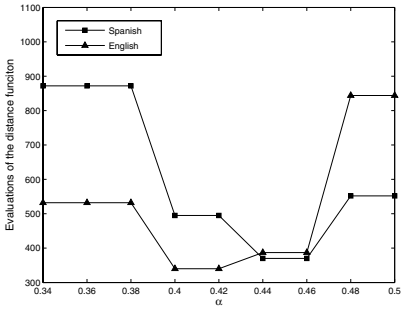
We ran the same tests with a collection of $51,589$ words taken from the Spanish dictionary. As in the case of the English dictionary, approximately the 10% of the database was used as queries ($5,200$ queries) and we also used a query range $r = 2$ to retrieve around the 0.02% of the database for each query. Figure 2 shows the results of this test.

Method	pivots	eval. d
Random	200	443
Good pivots	200	389
Outliers	200	429
SSS	205	370

Table 3. Efficiency in a collection of words



(a) Number of pivots selected in terms of α



(b) Number of evaluations of d in terms of α

Figure 2. Experiments with collections of words

In figure 2 we can notice important differences in both the number of pivots selected and the number of evaluations of the distance function. In the case of the Spanish dictionary, the number of pivots selected is much smaller than that of the English case. However, the optimum number of evaluations of d is very similar for both cases, and its obtained for very similar values of α. Since the two sets are both collections of words this result could seem strange. However, this happens because the two spaces have a different complexity. The number of words in the English collection is a bit bigger than in the Spanish case. In addition, the word length distribution is different in each collection, and this

fact has an important influence in the result of the query. In spite of this differences, our method has selected an appropriate number of pivots for each space, obtaining a similar efficiency in both of them. This result is another evidence of the ability of our proposal to adapt itself to the complexity of the space we are working with.

Figure 3 shows the results of the experiments with a collection of images from NASA archives. The $40,700$ images of the collection were extracted from the photo and video archives of NASA. The images were transformed into 20-dimensional vectors using the Euclidean distance to measure the similarity between them. As in the previous experiments, the 10% of the objects were used as queries. In this case, the query range retrieves on average the 0.10% of the database. In figure 3 we can see that SSS performs always better than a random pivot selection. However, in this case our strategy has not a better behavior than *Incremental* [5], that performs 220 evaluations with 60 pivots, whereas our better result is 255 evaluations with 77 pivots.

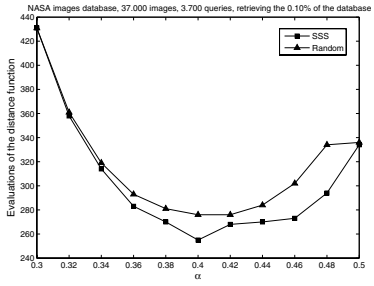


Figure 3. Efficiency with the NASA images.

5 Conclusions

In this paper we proposed a new dynamic pivot based search method. The main contribution of our method is the pivot selection strategy. The goal pursued by our pivot selection technique is to generate a set of pivots well distributed over the whole space. Furthermore, the index structure can be efficiently stored in secondary memory.

Our experimental results show that the method generates a number of pivots that depends on the intrinsic dimensionality of the metric space, and not on the number of elements of the collection. This number of pivots is very similar to the optimum number for other strategies. This makes it unnecessary to state in advance the number of pivots needed for the index structure, something that no method has considered until now. The number of pivots selected is adapted to the space complexity, avoiding to select unnecessary pivots that could reduce the search efficiency. The efficiency of our method in vector and metric spaces is at least as good as the best obtained in previous works [5].

Our work line still maintains opened some questions that will be addressed in the future. First, we are evaluating the behavior of SSS with other real metric spaces, like collections of text documents or music. We are also working in refinements of the pivot selection strategy and in the development of techniques that try to estimate automatically the optimum value of α in terms of the complexity of the space.

References

[1] R. Baeza-Yates. Searching: an algorithmic tour. *Encyclopedia of Computer Science and Technology*, 37:331–359, 1997.

[2] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 198–212. Springer-Verlag, Berlin, 1994.

[3] S. Brin. Near neighbor search in large metric spaces. In *21st conference on Very Large Databases*, 1995.

[4] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, 1973.

[5] B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity search in metric spaces. In *SCCC 2001, Proceedings of the XXI Conference of the Chilean Computer Science Society*, pages 33–40. IEEE CS Press, 2001.

[6] E. Chávez, J. L. Marroquín, and G. Navarro. Overcoming the curse of dimensionality. In *European Workshop on Content-based Multimedia Indexing (CBMI'99)*, pages 57–64, 1999.

[7] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[8] I. Kalantari and G. McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, 9:631–634, 1983.

[9] L. Micó, J. Oncina, and R. E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (aesa) with linear pre-processing time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.

[10] G. Navarro. Searching in metric spaces by spatial approximation. In *Proceedings of String Processing and Information Retrieval (SPIRE'99)*, pages 141–148. IEEE CS Press, 1999.

[11] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.

[12] E. Vidal. An algorithm for finding nearest neighbors in (aproximately) constant average time. *Pattern Recognition Letters*, 4:145–157, 1986.

[13] P. Yianilos. Data structures and algorithms for nearest-neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321. ACM Press, 1993.

[14] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity search. The metric space approach*, volume 32 of *Advances in Database Systems*. Springer, 2006.