Spatial Selection of Sparse Pivots for Similarity Search in Metric Spaces *

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

Nora Reyes
Departamento de Informática, Universidad Nacional de San Luis
Ejército 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). The main characteristic of this method is that it 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, without being necessary to specify in advance the number of pivots to use. Furthermore, SSS is dynamic, that is, it is capable to support 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. We also show that the efficiency of our proposal is similar to that of other existing ones over vector spaces, although it is better over general metric spaces.
Keywords: Similarity search, metric spaces, pivot selection, databases, searching, indexing.

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 similar words to a given one to correct edition errors) to big search engines (retrieving relevant documents for a given query). We can find more application 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). Unfortunately, the high computational cost of the functions that measure the distance between two objects makes similarity search a very expensive operation. This fact has motivated the development of many research works aiming to do efficient similarity search over large collections of data.

The problem of *similarity search* 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* (X,*d*) is composed of a universe of valid objects X and a *distance function* $d : X \times X \rightarrow \mathbb{R}^+$ 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 $U \subseteq X$ with size $n = |U|$, 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 $L_s(x,y) = (\sum_{1\leq i\leq k} |x_i - y_i|)^{1/s}$. For example, 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*. In a collection of words, we can use edit distance $d(x,y)$, obtained as the number of insertions, deletions or modifications of characters need to transform a string *x* into other string *y*.

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 search methods is worse in metric spaces with a higher intrinsic dimensionality [8].

There are three main queries of interest for a collection of objects in a metric space: *i) range search*, that retrieves all the objects $u \in U$ within a radius *r* of the query *q*, that is: $\{u \in U \wedge d(q, u) \leq r\}$; *ii) nearest neighbour search*, that retrieves the most similar object to the query *q*, that is: $\{u \in U \,/\, \forall\, v \in U,\ d(q,u) \leq d(q,v)\}$; and *iii) k-nearest neighbours search*, a generalization of the nearest neighbour search, retrieving the set $A \subseteq U$ such that $|A| = k$ and $\forall\, u \in A, v \in U - A, d(q,u) \leq d(q,v)$. The range query is the most used, and the others can be implemented in terms of it [8]. In any case, the distance function is the unique information that can be used in the search operation. Thus, the basic way of implementing these 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 usually differ in some features. Some of them only support discrete distance functions (e.g. the edit distance), while others where developed to work with

* Partially supported by Xunta de Galicia grants 2006/4 and PGIDIT05SIN10502PR, and Ministerio de Educación y Ciencia (PGE y FEDER) grant TIN2006-15071-C03-03

continuous distance functions. This is an important issue that restricts their application in some domains. The distinction between static and dynamic methods is very important. With static methods the index has to be built over the whole collection, while dynamic methods allow insert/delete operations in the database and build the index as the database grows. Other important factor is the possibility of storing the index efficiently in secondary storage, and the number of I/O operations needed to access it. In general, the applicability and efficiency of the method depends on these issues.

Search methods can be classified into two types [8]: *clustering*-based and *pivot*-based techniques. *Pivot*-based search methods choose a subset of the objects in the collection that are used as *pivots.* The index stores the distances from each pivot to each object in the collection in adequate data structures. 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 $x \in$ U an object in the collection, we can discard x if $|d(p_i,u) - d(p_i,q)| > r$ for any pivot p_i. Some representative examples of pivot-based methods are: *Burkhard-Keller-Tree* (BKT) [5], *Fixed-Queries Tree* (FQT) [2], *Fixed-Height FQT* (FQHT) [1], *Fixed-Queries Array* (FQA) [7], *Vantage Point Tree* (VPT) [14] and its variants [3] and [15], *Approximating and Eliminating Search Algorithm* (AESA) [13] and LAESA (*Linear AESA*) [10].

Clustering-based techniques split the metric space into a set of clusters each represented by a cluster center. Given a query, whole regions can be discarded from the search result using the distance from their center to the query and the triangle inequality. The most important *clustering*-based techniques are: *Bisector Trees* (BST) [9], *Generalized-Hyperplane Tree* (GHT) [12], *Geometric Near neighbor Access Tree* (GNAT) [4] and *Spatial Approximation Tree* (SAT) [11]. More details on the problem of searching metric spaces and complete descriptions of all the algorithms enumerated here can be found in [8] and [16].

In this paper we present *Sparse Spatial Selection* (SSS), a new pivot-based technique that permits to deal with dynamic collections and continuous distance functions. It is also well-suited for its use in secondary memory. It is a dynamic method since the collection can be initially empty and/or grow as more elements are added to the database. The main contribution of SSS is the use of a new pivot selection strategy. This strategy selects a number of pivots that depends on the intrinsic dimensionality of the space and not on the size of the collection (which is interesting both from theoretical and practical issues). Moreover, this pivot selection strategy is dynamic and is able to maintain the efficiency of the index during searches, as 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 presented in Section 3. In Section 4 we present and discuss the experimental results obtained in our experiments. 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 at random. 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 work, some heuristics for pivot selection have been proposed. For example, in [10] pivots are objects maximizing the sum of distances between them. [14] and [4] propose heuristics to obtain pivots far away from each other. In [6] 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 [6] is a criterion to compare the efficiency of two sets of pivots of the same size. Let $\{p_1, p_2, \ldots, p_k\}$ be a set of pivots, with $p_i \in$ U. Given an object $u \in$ U, a tuple that contains all the distances from u to each pivot is denoted as $[u] = (d(u,p_1), d(u,p_2), \ldots, d(u,p_k))$. Thus there is a space P = $\{[u] \mid u \in X\}$, that is also a vector space $\mathbb{R}^k$. Over the elements of this space the distance function $D_{\{p1, p2, \ldots, pk\}}([x],[y]) = \max_{\{1 \le i \le k\}} | d(x,p_i) - d(y, p_i)|$ can be defined. Then, we have a metric space (P, L_∞). Under this conditions, given a query q and a radius r, the condition to discard $u \in$ U can be seen as $|d(p_i,u) - d(p_i, q)| > r$ for some pivot p_i, in $D_{\{p1, p2, \ldots, pk\}}([q], [u]) > r$. Since the more objects a set of pivots can discard, the better it is, then a set of pivots will be better than others if it increases the probability of $D_{\{p1, p2, \ldots, pk\}}([q], [u]) > r$. Being μ_D the mean of the distribution D, that probability increases when μ_D is maximized. Therefore, authors' criterion establishes that the set of pivots $\{p_1, p_2, \ldots, p_k\}$ is better than the set of pivots $\{p_1', p_2', \ldots, p_k'\}$ if $\mu_{\{p1, p2, \ldots, pk\}} > \mu_{\{p1', p2', \ldots, pk'\}}$.

In [6] several selection strategies based on the previous efficiency criterion were proposed: *i) Random*, that chooses the set of pivots randomly; *ii) Selection*, that selects N random sets of pivots and finally chooses the one maximizing μ_D; *iii) 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 *iv) 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. The performance obtained by many of these techniques depends on the metric space considered. Their conclusions show that, in general, good pivots should be far from each other and also from the rest of objects in the database. Unfortunately, this second condition does not always lead to obtain good pivots.

Determining the optimal number of pivots (the value k) is an important problem. However, it is known that the efficiency of the searches depends on that parameter. Moreover, k can vary greatly for different metric spaces. All the pivot selection techniques shown use a precomputed fixed value. In [6] a brute-force approach to determine the optimal number of pivots is used. Results confirm that this number depends greatly on the metric space, and affects the efficiency of the methods. Therefore, adjusting it as well as possible is an interesting problem.

3. OUR PROPOSAL: SPARSE SPATIAL SELECTION

In this section we present a new pivot-based search method called *Sparse Spatial Selection* (SSS). It is an efficient method with some important advantages over the previous work. It is a dynamic method since the database can be initially empty and grow/decrease later as objects are inserted/deleted. It is adaptive since the method adapts itself the index to the complexity of the collection as the database grows. The main contribution of SSS is the pivot selection strategy, which selects dynamically a set of pivots well-distributed in the metric space, a property that permits to discard more objects from the result when solving a search.

3.1. Pivot selection strategy

Let (X,d) be a metric space, $U \subseteq X$ an object collection, and M the maximum distance between any pair of objects, $M = \max \{d(x,y) / x, y \in U\}$. First, the set of pivots is initialized with the first object of the collection. Then, for each element $x_i \in U$, x_i is chosen as a new pivot iff its distance to any pivot in the current set of pivots is equal or greater than $M\alpha$, being α is a constant which optimal values are around 0.4 (as shown later). 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 as a new pivot if it is located further than a half of the maximum distance from the current pivots. Next pseudocode summarizes the process of pivot selection:

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

It is evident that all the selected pivots will be far from each other (more that $M\alpha$), a desirable characteristic in a set of pivots [6]. However, our selection strategy has even more advantages. Forcing the distance between two pivots to be greater or equal than $M\alpha$, we ensure that they are well distributed in the space. It is important to take into account that our pivots are not very far from others neither very far from the rest of objects in the collection. These are two important conditions that, as shown in previous works, good pivots have to maintain. Our hypothesis is that, being well distributed in the space, the set of pivots will be able to discard more objects in the search.

Being dynamic and adaptive is other good feature of our pivot selection technique. The algorithm adapts itself the set of pivots to the growth 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. Therefore, the set of pivots maintains its good distribution in the metric space. Actually the collection could be initially empty, which is interesting in a practical application of this method. In this way, the growth of the collection does not imply a decrease in the performance of the algorithm.

3.2. Intrinsic dimensionality of the metric space

In a vector space, the dimensionality can be seen as the number of elements of each vector. Although general metric spaces do not have an explicit dimensionality, following the same idea that in vector spaces, we can talk about their *intrinsic dimensionality*. These interesting concepts allow us to distinguish between low and high dimensionality values. The efficiency of similarity search algorithms is worse with a high intrinsic dimensionality. For instance, any search method will behave better in a vector space with dimensionality 10 than in other with dimensionality 15. This fact has been empirically proved in previous research. In [8] this concept and its influence in the efficiency of search methods were analyzed. That paper proposed also a way to obtain an approximation of the intrinsic dimensionality as: $\rho = \mu / 2\sigma^2$ where μ and σ^2 are respectively, the mean and variance of the histogram of distances among points in the metric space.

Another important feature in our method is that the number of pivots chosen does not depend on the size of the collection, but in the intrinsic dimensionality of the metric space. For example, assuming $\alpha = 0.5$, in a 1-dimensional space we could only need two pivots, in a 2-dimensional space we could have at most 4 pivots, etc. This question has never been taken into account in previous search techniques. Therefore, another advantage of our pivot selection method is that the number of pivots generated is adapted to the intrinsic dimensionality of the space (even if it is unknown). When we are dealing with spaces whose intrinsic dimensionality cannot be approximated these results of special interest (it helps to determine the number of pivots). When the number of pivots is too small, they could not be enough to cover all the dimensions of the space, what could lead to less efficient searches. As our proposal generates a number of pivots depending on the dimensionality of the space, the number of chosen pivots should grow quickly when the first objects of the collection are processed. Then, the number of pivots should become stable when the number of processed elements is high. In Section 4 we provide empirical results that confirm this hypothesis.

3.3. The parameter α

Although in our method it is not necessary to state in advance the number of pivots to use, as in [6], we have to set in advance the value of α. This value in turn conditions the number of pivots. However, α must always take values between 0.35 and 0.40, depending on the dimensionality of the space. Figure 1 shows the number of evaluations of the distance function in terms of α for vector spaces of dimensionalities 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. We can also see that when $\alpha >$ 0.40 the number of evaluations of the distance function takes higher values in spaces of high dimensionality. This result is due to the fact that an increase in the value of α implies a reduction of the number of pivots, and that this reduction has a stronger effect in spaces of higher dimensionality.

These 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 far away from 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 a similar efficiency 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 complexity of the metric space, using only the maximum distance between any pair of objects in the collection.

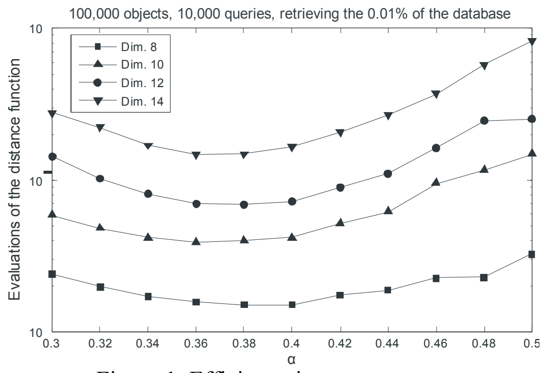


Figure 1. Efficiency in vector spaces

3.4. Index construction and collection growth

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, the distance from every object in the database to the new pivot 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 collection does. This implementation makes it possible the index to be completely dynamic, and the set of pivots to adapt 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.

3.5. Searching

We finally describe how to use the index in the range query operation, since the other query types can be implemented in terms of this one. Being (q,r) a query, the first step consists in computing the distance from q to every pivot in the index. With this information, we can discard every object $x_i \in U$ such that $|d(x_i,p_j) - d(q,p_j)| > r$ for any pivot p_j, since by the triangle inequality $(d(x,y) \leq d(x,z) + d(z,y))$, if this condition is true, its distance to q will be $d(x_i,q) > r$. The objects that are not discarded by this condition make up the candidate list $\{u_1, u_2, ..., u_m\} \subseteq U$ and they must be directly compared against the query.
The complexity of the search operation is measured as the number of evaluations of the distance function. First we have to compare the query q against every pivot. These distance computations constitute the *internal complexity* of the algorithm. Then we have to compare the query q with each object in the candidate list. These distance evaluations constitute the *external complexity* of the algorithm. The *total complexity* is the sum of the internal and external complexities [8].

4. EXPERIMENTAL RESULTS

Our method has been tested with several data collections in different situations. First we used synthetic sets of random points in vector spaces of dimensionalities 8, 10, 12 and 14. Although they are objects of a vector space, this information has not been used in the tests. The advantage of using this data types to test the algorithm is that we can study its behaviour in spaces with different intrinsic dimensionality. The Euclidean distance was the distance function used with these data sets. We also have tested the algorithm with real metric spaces: collections of words extracted from the English and Spanish dictionaries, using the edit distance as distance function.

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 used collections of 1,000,000 of vectors of dimensions 8, 10, 12 and 14. For each vector space we obtained the number of pivots selected in terms of the number of objects inserted in the collection, with α fixed to 0.5.

k	*n*, collections size (× 10^3)									
	100	200	300	400	500	600	700	800	900	1000
8	16	17	19	20	21	22	22	22	22	22
10	20	24	28	29	30	30	30	30	30	30
12	44	50	53	54	55	57	58	58	58	58
14	56	62	69	71	73	79	80	80	82	82

Table 1. Number of pivots selected in vector spaces of dimensionality 8, 10, 12, 14 in terms of the size of the collection

Table 1 shows the results obtained in this experiments. First we can see that 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. Let us take a look now to the number of pivots in terms of the collection size. In all the test spaces the number of pivots grows quickly with the first objects of the database. Then this 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 these 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 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 behaviour of the algorithm, we compared the results with those obtained with the pivot selection techniques proposed in [6].

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
Selection	85	204	200	446	360	986	800	2038
Incremental	65	157	150	335	300	714	600	1458
Loc. Opt. A	70	155	150	333	300	708	600	1448
Loc. Opt. B	60	157	150	369	300	881	760	1930
SSS	57	151	148	389	258	689	598	1452

Table 2. Minimum number of evaluations of *d* with different pivot selection strategies in vector spaces.

Table 2 shows the minimum number of evaluations of *d* (#d) we obtained with each pivot selection strategy, 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 [6]. In some cases we performed less evaluations of *d* with 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 a similar efficiency to that of other methods. In the results of our tests we can also see that the number of pivots that our method selects is very similar to the optimum number of pivots of other pivot selection techniques.

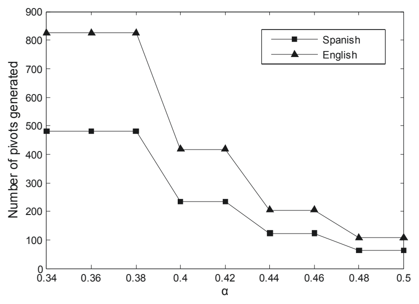
4.3. Search efficiency in metric spaces

In addition to the tests with uniformly distributed vector spaces, we have also compared our method with others using "real" metric spaces. More specifically, we used two collections of words. The first one contains 69,069 words from the English dictionary, and the second contains 51,589 words from the Spanish dictionary. We 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 obtained with our pivot selection technique and the ones proposed in [6], 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 selects is similar to the optimal number of pivots used by other strategies that have got this number by trial and error.

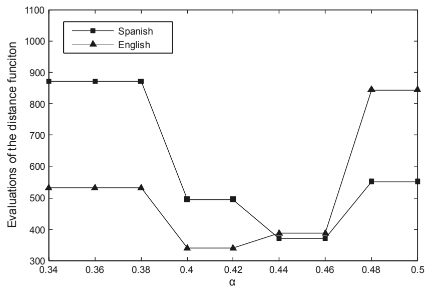
Methods	pivots	eval. d
Random	85	213
Good Pivots	85	204
Outliers	65	157
SSS	57	151

Table 3. Minimum number of evaluations of *d* in a collection of words taken from the English dictionary.

Finally, we ran the same tests with a collection of words 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), we also used a query range $r = 2$ to retrieve around 0.02% of the database for each query. Figure 2 shows the results of this test.



(a) Number of pivots selected in terms of α



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

Figure 2. Minimum number of pivots selected (a) and number of evaluations of the distance function for different values of α (b) in collections of words taken from the English and Spanish dictionaries

In figure 2 we can notice important differences both in 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 the obtained with the English dictionary. However, the optimum number of evaluations of *d* is very similar in both cases, and is 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 the number of words in the Spanish collection. 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 these differences, our method has selected an appropriate number of pivots for each space, obtaining a similar efficiency in searches 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.

5. CONCLUSIONS AND FUTURE WORK

In this paper we propose a new pivot-based method for similarity search in metric spaces. The main characteristics of this method are its efficiency (as we have seen in Section 4), dynamism (which allows the database to be initially empty and grow later) and adaptability (since the method adapts itself the number of pivots and the index to the complexity of the collection). The main contribution of our method is the pivot selection strategy, the responsible of these three important characteristics. In addition, the index structure makes possible its efficient storage in secondary memory.

Our experimental results show that the method selects a number of pivots that depends on the intrinsic dimensionality of the metric space, and not on the number of elements of the collection. In addition, 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 the selection of unnecessary pivots which could reduce the search performance. The efficiency of our method in vector spaces is similar to that obtained in previous works. However, our tests show that our method is more efficient than the existing ones in similarity search over general metric spaces.

6. REFERENCES

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

[2] Ricardo Baeza-Yates, Walter Cunto, Udi Manber, and Sun Wu. Proximity matching using fixed-queries trees. In *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 198-212, Springer-Verlag, 1994.

[3] Tolga Bozkaya and Meral Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD 1997)*, pages 357-368, May 1997.

[4] Sergey Brin. Near neighbor search in large metric spaces. In *21st conference on Very Large Databases (VLDB)*, 1995.

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

[6] Benjamín Bustos, Gonzalo Navarro, and Edgar 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 Computer Science Press, 2001.

[7] Edgar Chávez, José Luis Marroquín, and Gonzalo Navarro. Overcoming the curse of dimensionality. In *European Workshop on Content-based Multimedia Indexing (CBMI'99)*, pages 57-64, 1999.

[8] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273-321, September 2001.

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

[10] Luisa Micó, José Oncina, and R. Enrique 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.

[11] Gonzalo Navarro. Searching in metric spaces by spatial approximation. In *Proceedings of String Processing and Information Retrieval (SPIRE'99)*, pages 141-148. IEEE Computer Science Press, 1999.

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

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

[14] Peter 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.

[15] Peter Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *Proceedings of the 6th DIMACS Implementation Challenge: Near neighbour searches (ALENEX 1999)*, January 1999.

[16] Pavel Zezula, Giuseppe Amato, Vlatislav Dohnal, and Michal Batko. *Similarity search. The metric space approach*, volume 32 of *Advances in Database Systems*. Springer, 2006.