Dynamic Selection of Suitable Pivots for Similarity Search in Metric Spaces

Claudia Deco[1], Mariano Salvetti[1], Nora Reyes[2] and Cristina Bender[1]

[1] Facultad de Ciencias Exactas, Ingenieria y Agrimensura,
Universidad Nacional de Rosario, (2000) Rosario, Argentina
deco@fceia.unr.edu.ar, salvettimariano@hotmail.com, bender@fceia.unr.edu.ar
[2] Departamento de Informática,
Universidad Nacional de San Luis, (5700), San Luis, Argentina
nreyes@unsl.edu.ar

Abstract. This paper presents a data structure based on Sparse Spatial Selection (SSS) for similarity searching. An algorithm that tries periodically to adjust pivots to the use of database index is presented. This index is dynamic. In this way, it is possible to improve the amount of discriminations done by the pivots. So, the primary objective of indexes is achieved: to reduce the number of distance function evaluations, as it is showed in the experimentation.

Keywords: Metric databases, dynamic index, Sparse Spatial Selection.

1 Introduction

The digital age creates a growing interest in finding information in large repositories of unstructured data that contain textual data, multimedia, photographs, 3D objects and strings of DNA, among others. In this case, it is more useful a similarity search than an exact search. Similarity search is usually an expensive operation.

The similarity search problem can be formalized through the concept of metric space. Given a set of objects and a distance function between them, which measures how different they are, the objective is to retrieve those objects which are similar to a given one. An index is a structure that allows fast access to objects, and accelerates the retrieval. There are several types of indexes proposed for metric spaces that have differences, such as how they are explored, or how they store the information.

This paper presents an improvement to Sparse Spatial Selection (SSS). This improvement consists on implementing new policies of incoming and outgoing pivots, in order to that the index suits to searches, to dynamic collections, and to secondary memory.

The rest of the paper is structured as follows: Section 2 presents basic concepts. Section 3 describes the problem of pivots selection. Section 4 presents the proposed method, and Section 5 shows experimental results. Finally, conclusions and future lines of work are presented.

2 Basic Concepts

A *metric space* (*X, d*) consists of a universe of valid objects *X* and a *distance function* $d:X\times X\rightarrow\Re^{+}$ defined among them. This function satisfies the properties: strictly positiveness $d(x,y)>0$, symmetry $d(x,y)=(y,x)$, reflexivity $d(x,x)=0$ and triangular inequality $d(x,y)\leq d(x,z)+d(z,y)$. A finite subset *DB* of *X*, with $|DB|=n$, is the set of elements where searches are performed. The definition of the distance function depends on the type of objects. In a vector space, *d* may be a function of Minkowski family: $L_s((x_1, ..., x_k),(y_1, ..., y_k))=(\sum |x_i-y_i|^s)^{1/s}$. Some examples are: Manhattan distance ($p=1$), Euclidean distance ($p=2$), and Chebychev distance ($p=\infty$).

In metric databases queries of interest can be: range search and *k*-nearest neighbors search. In the first, given a query *q* and a radius *r*, objects that are at a distance less than *r* are retrieved: $\{u\in DB / d(u,q)\leq r\}$. In *k nearest neighbors* search, the *k* objects closest to *q* are retrieved, that is: $A\subseteq DB$ such that $|A|=k$ and $\forall u\in A$, $v\in DB-A$, $d(q,u)\leq d(q,v)$. The basic way of implementing these operations is to compare each object in the collection with the query. The problem is that, in general, the evaluation of the distance function has a very high computational cost, so searching in this manner is not efficient when the collection has a large number of elements. Thus, the main goal of most search methods in metric spaces is to reduce the number of distance function evaluations. Building an index, and using the triangular inequality, objects can be discarded without comparing them with the query. There are two types of search methods: *clustering-based* and *pivots-based* [1]. The first one splits the metric space into a set of equivalence regions, each of them represented by a *cluster center*. During searches, whole regions are discarded depending on the distance from the cluster center to the query. *Pivot-based* algorithms select a set of objects in the collection as *pivots*. An index is built by computing distances from each object in the database to each pivot. During the search, distances from the query *q* to each pivot are computed, and then some objects of the collection can be discarded using the triangular inequality and the distances precomputed during the index building phase. Some pivot-based methods are: *Burkhard-Keller-Tree* [2], *Fixed-Queries Tree* [3], *Fixed-Height FQT* [3], *Fixed-Queries Array* [4], *Vantage Point Tree* [5], *Approximating and Eliminating Search Algorithm* [6], *Linear AESA* [7] y *SSS* [8,9].

This paper presents an improving to the Sparse Spatial Selection (SSS) method, implementing new policies for selecting incoming and outgoing pivots from the index. The proposed method is dynamic, because the collection can be initially empty, or can increase or decrease with time. Also, this proposal generates a number of pivots based on the intrinsic dimensionality of the space.

3 Related Work on Pivots Selection

Pivots selection affects the efficiency of the search method in the metric space, and the location of each pivot with respect to the others determines the ability to exclude elements of the index without directly comparing them with the query. Most search pivots-based methods select pivots randomly. Also, there are no guidelines to determine the optimal number of pivots, parameter which depends on the specific

collection. Several heuristics have been proposed for the selection of pivots, as if the distance function is continuous or discrete. In [7] pivots are objects that maximize the sum of distances among them. In [10] a criterion for comparing the efficiency of two sets of pivots of the same size is presented. Several selection strategies based on an efficiency criterion to determine whether a given set of pivots is more efficient than another are also presented. The conclusion is that good pivots are objects far away among them and to the rest of the objects, although this does not ensure that they are always good pivots.

In [8] the Sparse Spatial Selection (SSS) which dynamically selects a set of pivots well distributed throughout the metric space is presented. It is based on the idea that, if pivots are dispersed in the space, they will be able to discard more objects during the search. To achieve this, when an object is inserted into the database, it is selected as a new pivot if it is far enough from the other pivots. A pivot is considered to be far enough from another pivot if it is at a distance greater than or equal to $M \times \alpha$. M is the maximum distance between any two objects. α is a constant parameter that influences the number of selected pivots and its takes optimal experimental values around 0.4.

In all of the analyzed techniques for selecting pivots, the number of pivots must be fixed in advance. In [10] experimental results show that the optimal number of pivots depends on the metric space, and this number has great importance in the method efficiency. Because of this, SSS is important in order to adjust the number of pivots as well as possible. To improve SSS, we propose that the index suits to searches, after the index was adapted to the metric space.

4 Proposed Method

We present a new indexing and similarity searching method based on dynamic selection of pivots. The proposed method is *dynamic*, because it could be applied to an initially empty database that grows over time. The method is *adaptive*, because it is not necessary to preset the number of pivots to be used because the algorithm selects pivots as necessary to self-adapt it to space complexity.

In the construction of the index, SSS is applied to select the initial pivots. Then, as time passes and searches are performed, we apply new policies for selecting pivots in order to eliminate those least discriminating pivots from the index, and to select objects as candidate pivots to put them into the index. In this way, we can adapt dynamically the index to searches performed during a given time.

4.1 Initial construction of the index and growth of the collection

Let (X,d) be a metric space, where $DB \subset X$ is the database. Let M be the maximum distance between objects ($M=\max\{d(x,y) \mid x,y \in X\}$), and α a value between 0.35 and 0.40 [8]. The collection of elements is initially empty.

The first object x_1 inserted into the database, is the first pivot p_1. When the second (or new) object is inserted in the database, its distance to all pivots that are already in the index is calculated. If these distances are all greater than or equal to $M \times \alpha$, this

object is added to the set of pivots. That is, an object of the collection will be a pivot if it is more than a fraction of the maximum distance of all pivots. Thus, the set of pivots does not have to be selected randomly, because pivots are chosen as the database grows. Then, distances from the new pivot against to all database objects are calculated and stored. Also, the value of M is updated. The pseudo code is:

```
Input: (X,d), DB ,M , α
Output: Pivots
0   Pivots ← {x1}
1   FOR ALL x1 ∈ DB DO
2           IF(∀p ∈ Pivots, d(x1, p) >= M×α)
3               THEN Pivots ← Pivots ∪ {x1}
4           END IF
5           Recalculate the value of M, and update.
6   END FOR ALL
```

Pivots that were selected for the initial index are far apart (over $M \times \alpha$). For many authors, this is a desirable feature of the set of pivots. The number of pivots depends on the initial dimensionality of the metric space. When the construction begins, the number of pivots should grow rapidly in the index, but this number will be stabilized when the database grows.

4.2 Exchange of Pivots in the Index

Given a query *(q,r)*, distances of *q* toward all pivots is calculated. Applying triangular inequality, all elements $x_i \in DB$ such that $|d(x_i,p_j)-d(p_j,q)|>r$ for some pivot p_j are discarded. Not discarded objects form the list of candidates, $\{u_1,u_2,\ldots u_n\} \subset DB$, and should be compared directly with the query. From this list of candidates, the statistics on the elements of the database that are part of search results is increased in a unit. If $\max_{1\le j\le k} |d(q,p_j)-d(e,p_j)|>r$, then the element e is outside the query range. So, the pivot p_j discriminates the object $e \in DB$. With searches, statistics of discrimination of each pivot are stored. These statistics are calculated when search results are obtained.

Selection policy for the Outgoing Pivot. When a pivot is a "bad pivot"? In a query, at most n elements (i.e., all elements of the database) can be eliminated. This elimination would have split these n discriminations between k pivots. And in a query, it is possible to know which pivot discriminates each element $e \in DB$.

Taking into account the objects discriminated by the various pivots, and a set of B queries, we define the percentage of discrimination for a pivot p_i as $[\%Disc(p_i)]=Disc(p_i)/(B \times n)$, where $Disc(p_i)$ is the amount of items that pivot p_i discriminates and $(B \times n)$ represents the total of possible discriminations. Then, p_i is a "bad pivot" when $[\%Disc(p_i)] < 1/k$ where $1/k$ is an experimental threshold, which is proposed as a constant that depends on the number of pivots in the index. If $[\%Disc(p_i)] < 1/k$, we say that p_i is very little relevant to discriminate, at least in light of the B searches made to the database. Then, it is selected as a victim, and it could be replaced in a future. So, the selection policy of a victim pivot within the index is to choose the "less discriminating pivot". After B searches the pivot with the lower

percentage of discrimination is determined. If it is less than a threshold of tolerance with value *T*, it is replaced. Then, $T = 1 / (1.1 \times k)$ represents a 10% of tolerance, used to stabilize the algorithm. This parameter has been evaluated experimentally. Recalling that *k* is the current number of pivots in the index, this constant of tolerance is used in the next situation:

```
0   IF ( min_{1≤j≤k} discrimine [j] < 1 / (1.1 × k) ) THEN {
1           OutgoingPivot ← GetPivot();
2           ChangePivot();
3           GenerateIndex();
4   }
```

In line 0, it is evaluated whether the proportion of *j*-th discrimination is less than the tolerance threshold. If so, this pivot is defined as the "least discriminating pivot" leaving it available for pivots exchanging (line 2). When a pivot is replaced, a whole column of the distance matrix between the incoming pivot, which is returned from *GetPivot*() method, and all elements of the database are recalculated (*GenerateIndex*() method). The complexity of changing a pivot is $n \times \Theta(distance\ function)$. If discrimination percentage is not less than *T*, nothing is done.

Selection policy for the Incoming Pivot. To choose which object becomes a pivot, the policy is to propose "the candidate pivot". The approach is similar than to select a victim pivot. The idea is to use statistical data of database elements obtained from queries. An object $e \in DB$ will be a candidate pivot if it is most of the times in the list of candidates, because an element that was often part of the list of candidates is difficult to discriminate with the current pivots. This implies that if this element is selected as pivot, in future searches it will improve the percentage of discrimination around the region that surrounds it. Another benefit is that it adapts automatically pivots to the region where the majority of searches are made. This transforms the index into a dynamic structure, achieving its main objective: to reduce distance computations in searches. Then, the element that most often was a candidate will be taken as the incoming pivot.

The following pseudo code shows the implementation of the policy for selecting incoming pivot (*GetPivot*() method). In this method the account of times that *i*-th element was part of the list of candidates is used. Thus, the element that most often was taken as a candidate is chosen as the candidate pivot to enter to the index.

```
      Input: DB, Stats
      Output: Stats, an array with the time they get the items
in the database in the list of candidates.
      0 Candidate = NULL; maxCurrentStats = 0;
      1 FOREACH ( e on Database)) DO
      2     IF (Stats(e) > maxCurrentStats)
      4           THEN Candidate = e;
      5        maxCurrentStats = Stats(e);
      6     END IF
      7 END FOREACH
      8 RETURN Candidate
```

In line 0, variable values are initialized. In line 2 the statistical value of each element *e* is compared with the current maximum value, which represents the element

that most often was the candidate and which will be taken as incoming pivot. If the statistic value of *e* exceeds the current one, in line 5 *e* is selected as the future incoming pivot, and the iteration continues.

5 Experimental Results

For experimentation several sets of synthetic random points in vector spaces of dimension 8, 10, 12 and 14 are used. The Euclidean distance function is used. The number of distance evaluations required to answer a query using the proposed policies, how the index adjusted itself as queries are processed, and how database size affects the index performance, was analyzed. The database contains 100,000 objects, and range query retrieved the 0.02% elements from the database.

Number of Pivots in the Index. Our proposal creates a dynamic amount of pivots, depending on the space dimensionality, and not on the number of objects in the database. For experiments α=0.5 was set, in order to achieve a uniform and distant distribution of pivots in the space. This value of α was chosen from experimental results showed later (Figures 5 and 6). Table 1 and Figure 1 show the number of pivots depending on the collection size, as the number of elements is growing.

Table 1: Number of pivots selected in vector spaces of dimension 8, 10, 12 and 14.

	n, size of the collection (x 10^3)									
dim	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
8	11	12	12	12	13	13	14	14	15	16
10	13	18	20	20	21	21	21	21	21	22
12	25	28	28	31	32	34	34	34	34	34
14	38	47	53	60	60	61	63	66	69	69

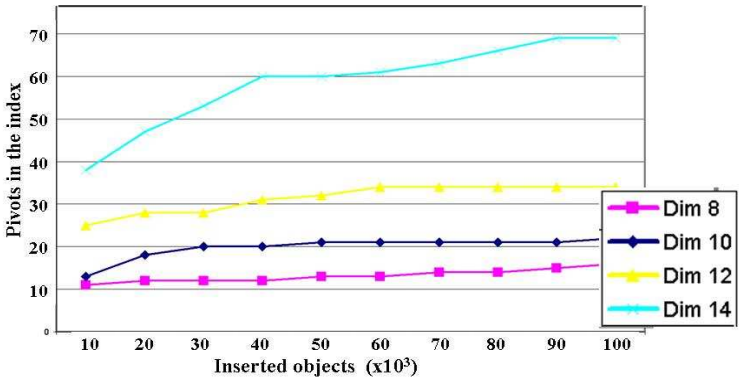

Fig. 1: Number of pivots selected in vector spaces of dimension 8, 10, 12 and 14.

As it is noted, the number of pivots grows very quickly with the insertions of the first objects in the collection, and then continues to grow but in a slower degree until

it get to stabilize. So, with few elements inserted, number of pivots depends on number of elements in the database. Already with a considerable amount of elements inside, the set of current pivots covers all the metric space. In addition, number of pivots in the index increases as the size of the space increases. So, the number of pivots in the index depends on the complexity of the collection of objects.

Search efficiency. To analyze the efficiency of the index for searching, a database with 10,000 objects, 1,000 queries and dimensionalities 8, 10, 12 and 14, are used. 20 periods were run, and information from all periods was averaged. For each dimension, the amount of distance functions evaluated (#DE), the amount of pivots used in the index (#P), and the amount of discriminations carried out (#DR), were recorded, assessing the proposed method against SSS.

Table 2: Efficiency in synthetic metric spaces with different methods.

	***dim* = 8**			***dim* = 10**			***dim* = 12**			***dim* = 14**		
Method	**#P**	**#DE**	**#DR**	**#P**	**#DE**	**#DR**	**#P**	**#DE**	**#DR**	**#P**	**#DE**	**#DR**
SSS	17	17994	6141634	24	26391	6393097	34	35721	6623490	60	62976	6915951
Proposal	15	15737	6394202	23	24380	6657667	33	34686	6717067	44	45683	7025895

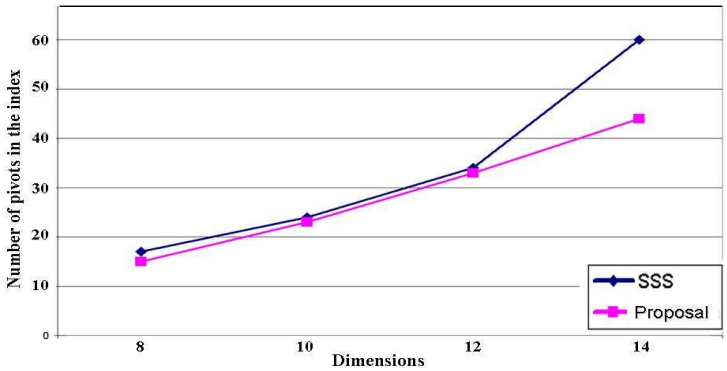

Fig. 2: Number of pivots, depending on dimensions 8, 10, 12 and 14.

Table 2 shows that the number of pivots used with our proposal is always lower than in the implementation of SSS, highlighting a marked difference in *dim*=14, with 16 pivots less. In the remaining dimensions, the difference is little, but it remains at most 2 pivots less in our favour. In Figure 2, it can be noted that our proposal has a minor number of pivots. This is an important result, because the strategy of pivots selection of SSS presents a similar efficiency to other more complexes methods and the number of pivots that it selects is close to the optimal number for other strategies.

Figure 3 shows the amount of distance evaluations. The number of evaluations for our proposal always remains below, and, in general, with a uniform linear growth when the size increases. Except in *dim*=14, where SSS shows a slight growth with the amount of reviews, with a difference of about 17,000 reviews, in other dimensions the difference never exceeds 3,000.

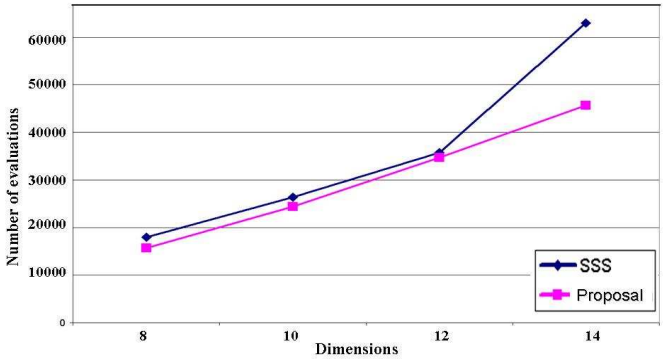

Fig. 3: Distance functions evaluated, depending on dimensions 8, 10, 12 and 14.

As results exposed in [8], the number of evaluations of the distance function in SSS is always around to the best result obtained with pivot selection techniques and strategies proposed in [10]. In conclusion, the proposal here presented, makes a number of distance evaluations similar to the best results obtained in previous works, even using a smaller number of pivots, which clearly implies space savings.

Figure 4 shows that the use of our proposal in searches obtains a greater number of discriminations by pivots, in all dimensions where it was experienced.

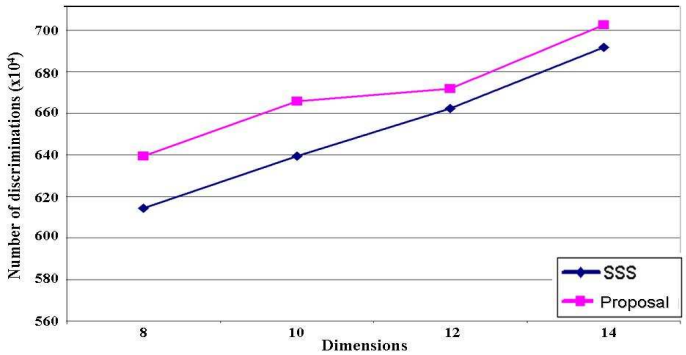

Fig. 4: Discrimination carried out, depending of dimension 8, 10, 12 and 14.

In the first two dimensions, there is a major difference between numbers of discriminations made. A great performance with the selected pivots with our selection policies, in contrast with pivots selected using pure SSS, is showed. This is because, with the time, the proposal makes an adjustment of pivots, and they make better discrimination, reducing the amount of computations of the distance function at query time. Thus, it shows that both selection policies of pivots (incoming and outgoing) are good, and that maintains the dynamism of the index.

About parameter α. The value of parameter α determines the number of pivots. Values between 0.35 and 0.40, depending on the dimensionality of the collection, are recommended [8]. Here we decided to use values of α from 0.32 to 0.54, in order to

evaluate the number of pivots in the index, since a rise in α represents a reduction in the number of pivots and this is noted better in spaces of higher dimension. In dimensions 2, 8, 10, 12 and 14, α varies in the range shown above.

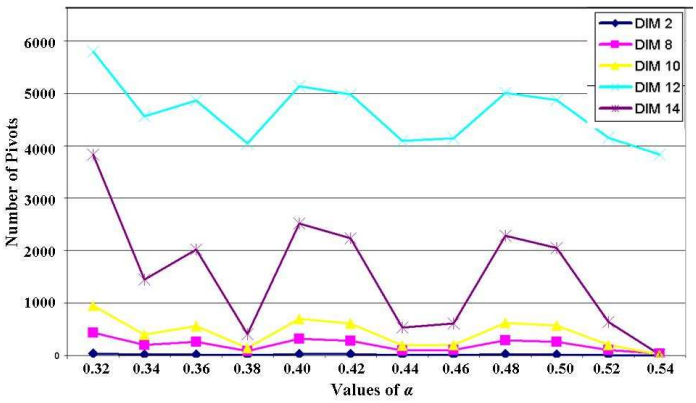

Fig. 5: Number of pivots selected by the parameter α.

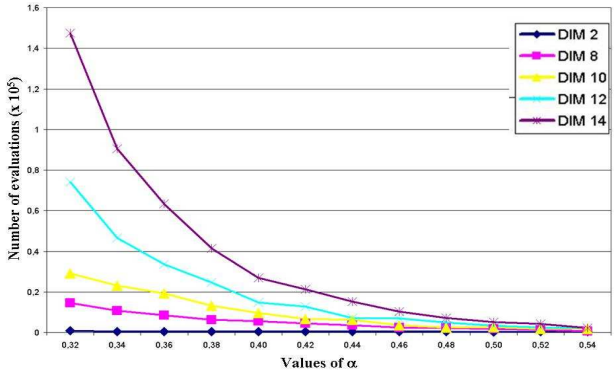

Fig. 6: Evaluations of the distance function according to the parameter α.

As shown in Figure 5 for all dimensions, the number of pivots varies with α, with some local maximum and minimum and large amplitude in greater dimensions. Then the observed value 0.50 decreased the number of pivots, as expected. Figure 6 shows the number of distance evaluations, varying α, in order to analyze the impact of our proposal in searches. As seen, in all dimensions there is a consistent behaviour, which gradually begins to decrease when α increases. This is because when distance between pivots increases, the required distance function evaluations decrease. This value has a uniform behaviour because values of 20 periods were averaged. The early periods have largest number of assessments and with the passing of periods, pivots were adapting to searches. In addition, it achieves more discrimination when α increases, because with the passing of periods, in our proposal, pivots are adjusted and searches are improved, discriminating more elements and decreasing the computations of distance functions.

5 Conclusions

This paper presents a new indexing and similarity search method based on dynamic selection of pivots. One of its most important features is that it uses SSS for the initial selection of pivots, because it is an adaptive strategy that chooses pivots that are well distributed in the space to achieve greater efficiency. In addition, two new selection policies of pivots are added, in order to the index suites itself to searches when it is adapted to the metric space. The proposed structure automatically adjusts to the region where most of searches are made, to reduce the amount of distance computations during searches. This is done using the policy of *'the most candidate*' for the incoming pivot selection, and the policy of *'the least discriminates'* for the outgoing pivot selection. Future work is to evaluate the performance of this proposal with real metric spaces, such as collections of texts or images. Moreover, from the results of experiments, to implement algorithms that work with indexes in secondary memory can be proposed. An improvement to selection policies would be to use a data warehouse for training the index with historical search data.

References

1. Chávez E., Navarro G., Baeza-Yates R., Marroquín J. L.: Searching in Metric Spaces. ACM Computing Surveys. 33(3), pp 273--321. (2001).
2. Burkhard W. A., Keller R. M.: Some approaches to best-match file searching. Communications of the ACM, 16(4):230-236.(1973).
3. Baeza-Yates R. A., Cunto W., Manber U., Wu S.: Proximity matching using fixed-queries trees. In M. Crochemore and D. Gusfield, editors, Proc. of the 5th Annual Symposium on Combinatorial Pattern Matching, LNCS 807, pages 198-212. (1994).
4. Chavez E., Navarro G., Marroquin A.: Fixed queries array: a fast and economical data structure for proximity searching. Multimedia Tools and Applications (MTAP), 14(2):113-135. (2001).
5. Yianilos P.: Excluded middle vantage point forests for nearest neighbor search. In: 6th DIMACS Implementation Challenge: Near Neighbour searches ALENEX'99. (1999)
6. Vidal E. An algorithm for finding nearest neighbor in (approximately) constant average time. Pattern Recognition Letters 4, 145-157 (1986).
7. 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. In: Pattern Recognition Letters, 15:9-17(1994).
8. Pedreira O., Brisaboa N.R.: Spatial Selection of Sparse Pivots for similarity search in metris Spaces. In: 33nd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'07), LNCS vol: 4362, pp. 434--445. Springer (2007).
9. Pedreira O., Fariña A., Brisaboa N.R. and Reyes N.: Similarity search using sparse pivots for efficient multimedia information retrieval. In: The Second IEEE International Workshop on Multimedia Information Processing and Retrieval, pp. 881--888. (2006).
10. Bustos B., Navarro G., Chávez E.: Pivot selection techniques for proximity search in metric spaces. In: XXI Conference of the Chilean Computer Science Society, pp. 33-44. IEEE Computer Science Press. (2001).