A Dynamic Pivot Selection Technique for Similarity Search *

Benjamin Bustos
University of Chile
Department of Computer Science
Center for Web Research
Blanco Encalada 2120, 8370459 Santiago, Chile
bebustos@dcc.uchile.cl

Oscar Pedreira, Nieves Brisaboa
University of A Coruña
Databases Laboratory
Campus de Elviña s/n, 15071 A Coruña, Spain
{opedreira,brisaboa}@udc.es

Abstract

All pivot-based algorithms for similarity search use a set of reference points called pivots. The pivot-based search algorithm precomputes some distances to these reference points, which are used to discard objects during a search without comparing them directly with the query. Though most of the algorithms proposed to date select these reference points at random, previous works have shown the importance of intelligently selecting these points for the index performance. However, the proposed pivot selection techniques need to know beforehand the complete database to obtain good results, which inevitably makes the index static. More recent works have addressed this problem, proposing techniques that dynamically select pivots as the database grows. This paper presents a new technique for choosing pivots, that combines the good properties of previous proposals with the recently proposed dynamic selection. The experimental evaluation provided in this paper shows that the new proposed technique outperforms the state-of-art methods for selecting pivots.

1 Introduction

Similarity search has been an active research field during the last years, and has become very important for other areas such as multimedia information retrieval, computational biology, pattern recognition, and many more. Lots of indexing methods based on the properties of metric spaces have been developed to efficiently solve similarity search in the problems that can be formalized through this concept. These methods are usually classified in *pivot-based* (which store precomputed distances to discard objects) or *clustering-based* algorithms (which partition the space into clusters to discard some of them during the search) [3]. The work presented in this paper is focused on the case of pivot-based algorithms. The reader not familiar with these topics can find a good introduction to the field of similarity search in [3] or [7].

Pivot-based algorithms select a set of objects from the collection as pivots, $\{p_1, p_2, \ldots, p_k\}$. To build the index, the distances from every object in the database to each pivot are computed and stored. At query time, the distances from the query object to the pivots are also computed, and the triangle inequality is applied to discard some objects from the result without comparing them with the query [3]. Something that most of the pivot-based algorithms have in common is that they select the pivots at random. However, it has been shown that the specific set of pivots has an important influence in the index performance [2, 7]. Although we are referring to pivot selection, this problem also applies to clustering-based algorithms, since the selection of cluster centers is also usually done at random. The position of the pivots in the space and their position with respect to each other determine the capacity of the index to discard objects without comparing them with the query. Other important problem is how to determine the optimal number of pivots. We could think that the higher the number of pivots, the more efficient the search. But the query has to be compared with both the pivots (internal complexity) and the objects that can not be discarded (external complexity), so we have to reach a good trade-off between the number of pivots and their power of discrimination.

Some of the first works that addressed the problem of pivot selection [4, 6, 1] proposed several heuristics based on two ideas: (a) good pivots are far away from the rest of objects in the metric space (i.e., good pivots are *outliers*), and (*b*) good pivots are also far away from each other [7]. For example, [4] proposes to choose as pivots objects that maximize the distances between pivots already chosen. The

*Funded by Millennium Nucleus Center for Web Research, Grant P04-067-F, Mideplan, Chile, and FONDECYT Project 11070037 (first author). Funded in part by MEC refs. TIN2006-15071-C03-03 and AP-2006-03214 (FPU Program for Oscar Pedreira), and Xunta de Galicia refs. PGIDIT05-SIN-10502PR and 2006/4 (second and third authors).

heuristic provided by [6] is based on the second moment of the distance distribution, and also selects objects that are far away from each other. In [1], a greedy strategy is proposed to also select far away objects (although this strategy was not initially proposed to select pivots, but split points).

The new approach we present in this paper is based in the techniques presented by [2] and [5], two recent and relevant proposals for the problem of pivot selection. Our proposal combines the good properties of both of them and dynamically selects more efficient pivot sets:

(a) [2] addressed the problem of pivot selection in a systematic and formal way. An important contribution of [2] is an efficiency criterion to compare the efficiency of two set of pivots of the same size. In addition, [2] proposes three techniques for pivot selection based in this criterion: *selection*, *incremental* and *local optimum*. An important characteristic of these techniques is that they work well in a wide variety of cases, and they are the first based on a formal theory. Although with these techniques it is possible to obtain a very efficient set of pivots, all of them involve a high computational cost and the number of pivots to select has to be stated in advance. Therefore, this number has to be computed by trial and error on the complete collection of data, which requires the database to be complete before the index construction. In addition, if a significant amount of objects is later inserted in the database, the index performance can be reduced. This makes the index static.

(b) *Sparse Spatial Selection* (SSS) [5] is a recently proposed pivot selection technique which dynamically selects an efficient set of pivots adapted to the complexity of the database. In SSS, when an object is inserted in the database, it is added to the set of pivots if it is far away enough from the pivots already selected. Following this procedure, the database can be initially empty and grow later, since the index is built as the database grows. As we will see in Section 2, the condition used to select a new pivot ensures that the pivots are well distributed in the space, which improves the index performance.

In this paper, we propose a new dynamic pivot selection strategy which permits the database to be initially empty and grow later, since the index is built as the database grows conserving its efficiency. The experimental results obtained, which outperform previous state-of-the-art approaches, are the result of the combination of the good properties of the techniques presented in [2] and [5]. The idea is to combine the criteria for pivot selection of both techniques. Given an object selected as a pivot with SSS, we add it to the set of pivots if it contributes to the efficiency of the set less than the worst current pivot. If it contributes more than the worst, the new object replaces the worst current pivot. Thus, the set of pivots is also selected dynamically but remains smaller and more efficient than applying only SSS.

The rest of the paper is organized as follows: Section 2 presents with more detail the strategies proposed in [2] and [5], since they are the base of the ones proposed in Section 3. Section 4 shows and discusses the experimental evaluation and, finally, Section 5 presents the conclusions of the paper and future work.

2 Related work

In this section, we describe with more detail the approaches proposed by [2] and [5], since they are the base for the new dynamic technique proposed in this work. In the rest of the paper we will suppose that $(\mathbb{X}, d)$ is a metric space and $\mathbb{U} \subseteq \mathbb{X}$ is the database or collection of data.

2.1 Pivot selection based on formal pivot efficiency estimation

An important contribution of [2] is a criterion for comparing the efficiency of two sets of pivots of the same size. Let $Pivots = \{p_1, p_2, \ldots, p_k\},\ p_i \in \mathbb{U}$ be a set of pivots. Given a query (q, r), by the triangle inequality we know that $d(p_i, x) \leq d(p_i, q) + d(q, x)$ and $d(p_i, q) \leq d(p_i, x) + d(x, q)$, for all $p_i \in Pivots$. Therefore, the condition to discard the object $x \in \mathbb{U}$ from the result set without comparing it with the query is:

$$|d(p_i, x) - d(p_i, q)| > r \tag{1}$$

for some pivot p_i. As explained in [2], a new space $\mathbb{P}$ (the *pivot space*) can be defined as the space of tuples of distances between objects $x \in \mathbb{X}$ and the pivots. Given an object $x \in \mathbb{U}$, its corresponding element in $\mathbb{P}$ is $[x] = (d(x, p_1), d(x, p_2), \ldots, d(x, p_k))$. Defining the distance D between tuples as $D_{\{p_1,p_2,\ldots,p_k\}}([x], [y]) = max_{1 \leq i \leq k} |d(x, p_i) - d(y, p_i)|$, $(\mathbb{P}, D)$ is a metric space too. Therefore, given a query (q, r) and an object $x \in \mathbb{U}$, the discard condition (1) is equivalent to:

$$D_{\{p_1,p_2,\ldots,p_k\}}([q], [x]) > r \tag{2}$$

The best set of pivots is the one which makes the probability of this equation as high as possible, since it will be the one which discards more objects from the result. One way to do this is to maximize the mean of the distance distribution of D, denoted μ_D. Thus, the

set of pivots $Pivots = \{p_1, p_2, \ldots, p_k\}$ is more efficient than $Pivots' = \{p'_1, p'_2, \ldots, p'_k\}$ if $\mu_{\{p_1,p_2,\ldots,p_k\}} > \mu_{\{p'_1,p'_2,\ldots,p'_k\}}$. Computing this values with all the database involves a high computational cost, but it can be reduced as explained in [2].

[2] also proposed several techniques for pivot selection based in the previous criterion. The most evident is *Selection*, which evaluates the criterion on N sets of pivots selected at random and returns the one that obtains the best result. *Incremental* is a more elaborated strategy that incrementally builds the set of pivots. A first pivot p_1 is selected from the data set which it alone has the better value of μ_D. Then another pivot p_2 is selected from the data set in such a way that $\{p_1, p_2\}$ has the better value of μ_D, considering p_1 fixed, and so on until completing the set of pivots. *Local Optimum* follows a different idea: it starts with a set of pivots selected at random, and in each iteration the pivot contributing less to μ_D (called *victim*) is replaced by a better pivot. This idea is combined in this work with SSS to build a more efficient while dynamic selection technique.

2.2 Sparse Spatial Selection (SSS)

Sparse Spatial Selection (SSS) [5] dynamically selects a set of pivots well distributed in the space, and adapted to the complexity of the collection. When a new object is inserted in the database, SSS selects it as a new pivot if it is far away enough from the pivots already selected. We consider that the new object is far enough if its distance to any already selected pivot is greater than or equal to $M\alpha$, being M the maximum distance between any two objects from the collection, and α a constant parameter that usually takes values in the interval $[0.35, 0.40]$. Algorithm 1 shows the pseudocode of SSS.

Algorithm 1: Sparse Spatial Selection

Input: $(\mathbb{X}, d)$, $\mathbb{U}$, α, M
Output: $Pivots$

```
Pivots ← {x_1};
foreach x_i ∈ U do
    if ∀ p ∈ Pivots, d(x_i, p) ≥ Mα then
        Pivots ← Pivots ∪ {x_i};
return Pivots
```

The parameter α directly affects the number of pivots selected by SSS. Experimental results provided in [5], show that the optimal values of this parameter are in the interval $[0.35, 0.40]$ for a wide variety of metric databases, and that the efficiency of SSS is virtually the same for all the values in this interval. The value of the maximum distance M between any pair of objects in the metric space can be estimated from the definition of the space and the distance function, not being necessary to directly compute it.

The results presented in [5] show that this strategy is more efficient than others previously proposed in most cases. Furthermore, SSS has other important features. SSS is dynamic, this is, the database can be initially empty, and the pivots will be selected when needed as the database grows. SSS is also adaptive, since it is no necessary to state in advance the number of pivots to select. As the database grows, the algorithm determines if the collection has become complex enough to select more pivots or not. Therefore, SSS adapts the index to the intrinsic dimensionality of the metric space [5]. The selection procedure is also more efficient in SSS than in previous techniques.

3 Dynamic pivot selection technique

The proposed dynamic pivot selection technique is based on the method SSS depicted in [5]. However, our method not only adds pivots to the index, but also checks if an already selected pivot has become redundant (i.e., its *contribution* is low according to some efficiency criterion), trying to replace it with a better suited pivot. It also checks if a pivot candidate would be redundant with respect to the actual set of pivots, in which case it is discarded immediately. The rest of this section describes in detail the proposed technique for selecting pivots.

3.1 Contribution of a pivot

The contribution of a pivot is computed using the criterion defined in [2]. Suppose that there is a set $Pivots$ of selected pivots and a set $Pairs$ with A pairs of objects randomly selected from $\mathbb{U}$. For each pair of objects $(x, y) \in Pairs$, the algorithm obtains the best pivot $p_{max} \in Pivots$ for that pair of objects; that is, the pivot p_{max} that maximizes the distance of $(x, y) \in Pairs$ in the pivot space ($|d(x, p) - d(y, p)|$). The algorithm computes then the second best pivot, $p_{max2} \in Pivots$ (the one that would maximize the distance of pair (x, y) in the pivot space if p_{max} were removed from $Pivots$). The contribution of p_{max} for the pair (x, y) is defined as:

$$|d(x, p_{max}) - d(y, p_{max})| - |d(x, p_{max2}) - d(y, p_{max2})|,$$

and the contribution of the other pivots in $Pivots$ for this pair is 0. The total contribution of a pivot $p \in Pivots$ is the sum of its contributions for all pairs of objects. Note that a pivot that has contribution equal to 0 does not maximize the distance in the pivot space for any pair of objects, thus this pivot is redundant (at least for those A pairs!). The pivot with smaller contribution is marked as the *victim*, which is the candidate for replacement in case one finds a better pivot.

Algorithm 2: Dynamic pivot selection algorithm

Input: $u \in \mathbb{X}$, $\mathbb{U}$, $Pivots$, α, M, A, c
Output: $Pivots$

```
// Initially, Pivots ← ∅
if ∀p ∈ Pivots, d(u,p) ≥ Mα then
    if |Pivots| < c then
        // The algorithm selects the
           first c pivots as in [5]
        Pivots ← Pivots ∪ {u};
    else
        // Compute the victim from
           Pivots
        (victim, contributionVictim, MaxD, Pairs) ←
        computeVictim(U, Pivots, A);
        // Compute the contribution of
           u (the pivot candidate)
        contributionNew ←
        computeContribution(u, A, MaxD, Pairs);
        // If the contribution is
           positive, decide between
           adding the pivot or
           replacing an old one
        if contributionNew > 0 then
            if contributionNew >
            contributionVictim then
                // Replace victim with
                   new pivot
                Pivots ← (Pivots − victim) ∪ {u};
            else
                // Add pivot to Pivots
                Pivots ← Pivots ∪ {u};
return Pivots
```

3.2 Algorithm for selecting pivots

Algorithm 2 shows the pseudocode of the dynamic selection of pivots. The index contains a set $Pivots$ of pivots (initially, $Pivots$ is empty). When an object $u \in \mathbb{X}$ is inserted into the database, the algorithm checks if u is farther than $M\alpha$ to all pivots. If this is the case, u is a pivot candidate. If the size of $Pivots$ is smaller than some (small) constant c, then u is added to $Pivots$ automatically (the computation of a victim only makes sense for a set $Pivots$ with at least two pivots). Otherwise, the algorithm must decide if it adds u to $Pivots$ or if u will replace some pivot p in $Pivots$.

Next, the algorithm computes the contribution of all pivots in $Pivots$ and chooses a victim according to the criterion defined in Section 3.1. Algorithm 3 shows the pseudocode for the selection of the victim.

Now, the algorithm computes the contribution of the pivot candidate u, using the same rule defined in Section 3.1. For each pair of objects, the algorithm computes the distance of pair (x, y) in the pivot space, using u as pivot. If this distance is greater than the distance obtained with the best pivot from $Pivots$ (stored in $MaxD$), the algorithm adds the corresponding contribution of u for pair (x, y), otherwise it adds nothing. The total contribution of u is the sum of its contribution for all pairs of objects. Algorithm 4 shows the pseudocode for this routine. If the contribution of u is 0, it is immediatly discarded as pivot.

Finally, the algorithm decides if the new pivot u should be added to $Pivots$ or the victim should be replaced. If the contribution of u is greater than the contribution of the victim, the victim is replaced with u. Otherwise, u is added to the $Pivots$, thus incrementing its size in one.

Note that this dynamic algorithm for selecting pivots ensures that it will select only pivots that are at distance at least $M\alpha$ to the other pivots. Thus, the set of selected pivots holds the same property of those selected using the method described in [5].

3.3 Complexity of the proposed technique

The space and time complexities of the proposed dynamic pivot selection methods are mainly given by function $computeVictim$. Regarding the space complexity, the algorithm needs space for storing the A pairs of objects ($2A$), the contribution of each pivots ($|Pivots|$), and the array $MaxD$ (A), thus the total space complexity is $O(A + |Pivots|)$. Regarding the time complexity, Algorithm 2 computes firstly the distances to the actual pivots ($Pivots$), then it needs to initialize the arrays ($A + |Pivots| + A$), it computes the contribution of each pivots ($A \cdot |Pivots|$), and finally computes the victim (A). Function $computeContribution$ performs only $O(A)$ extra instructions, thus the time complexity of Algorithm 2 is $O(A \cdot |Pivots|)$. If one considers a fixed set $\mathbb{U}$ that must be indexed, a loose upper bound of the total time complexity for selecting the pivots is $O(A \cdot |Pivots| \cdot |\mathbb{U}|)$, because the $O(A \cdot |Pivots|)$ operations are performed only when an object from $|\mathbb{U}|$ is sufficiently far away from the previously selected pivots.

It must be noticed that the time complexity of Algorithm 3 could be improved by reusing most of the pair of distances and by storing the corresponding computed distances (e.g., by changing only one pair of objects on each call of this routine). With this approach, the new time complexity of Algorithm 3 (and therefore of Algorithm 2) is $O(|Pivots|)$ (instead of $O(A \cdot |Pivots|)$). However, as the A pairs of objects are now not randomly selected on each iteration of Algorithm 3, it is possible that the quality of the estimation of the contribution of each pivot decreases. We

will present some experimental results with this improved method (Dynamic-LCC from "lower construction cost") in the experimental evaluation.

4 Experimental evaluation

4.1 Experimental environment

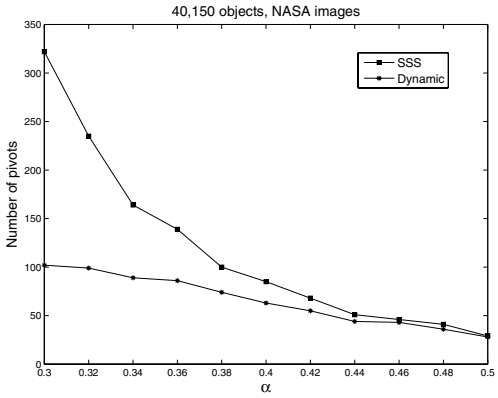
In the experimental evaluation we have used test collections representing real-world problems of similarity search. We have tested our proposal with two collections of images: the first one contains $40,150$ images (NASA) extracted from the NASA video and image archives, each of them represented by a feature vector of 20 components; the second collection contains $112,682$ color images (COLOR), each of them represented by a feature vector of 112 components. The Euclidean distance was used to compare objects in both of them. We also used a collection of $81,061$ words (SPANISH) from the Spanish dictionary with the edit distance.

As [5] shows that SSS is as efficient or more than any state-of-art approach, we only compare the dynamic technique we propose with SSS. In each experiment we ran each algorithm for values of α between 0.30 and 0.50. When comparing the efficiency of the index with the two pivot selection techniques, $1,000$ queries were executed, retrieving the 0.01% of the database in each of them in the case of images, and using a query range of 2 in the case of words, and showing the average result of all queries. As parameters, we used $c = 5$ and $A = 5.000$ (or $A = |\mathbb{U}|$ if the size of $\mathbb{U}$ was smaller than $5,000$) for all experiments.

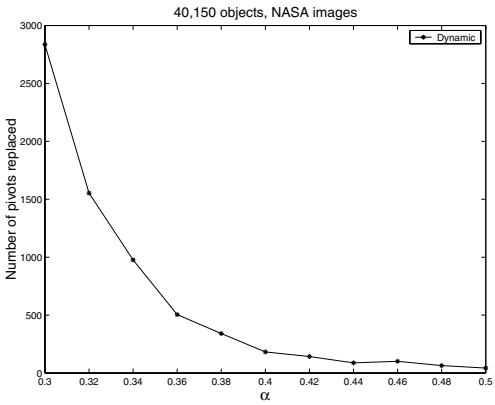
4.2 Size of the set of pivots

One of the important characteristics we depicted about Dynamic, is that its policy of replacement when a pivot becomes redundant, gives as a result smaller and more efficient sets of pivots. This subsection is devoted to the size of the pivot set obtained with SSS and Dynamic. In our experiments with the collections NASA, COLOR and SPANISH we compared the number of pivots obtained with each technique, and the number of replacements of pivots in the case of Dynamic.

Figures 1, 2 and 3 show the results. As we can see, the experimental results confirm our hypothesis since SSS also selects more pivots than Dynamic for all values of α in both NASA, COLOR and SPANISH. These results also show how Dynamic replaces more pivots for smaller values of α. Since SSS selects more and more pivots when α is smaller, the probability of having redundant pivots is higher and therefore Dynamic replaces most of them. This is the reason why the difference in the number of pivots selected



(a) Number of pivots in terms of α



(b) Number of pivots replaced in terms of α

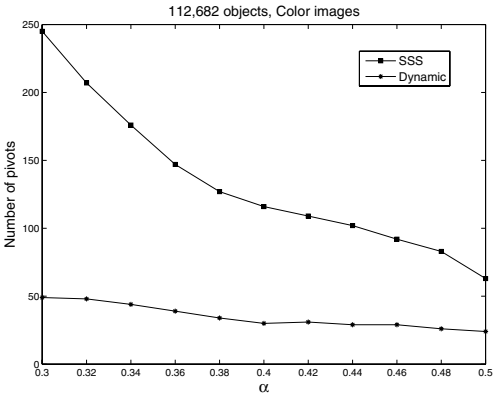
Figure 1. Size of the set of pivots with the NASA dataset

by SSS and Dynamic is higher for small values of α, and almost the same in the case of NASA experiments for values of α around 0.5.

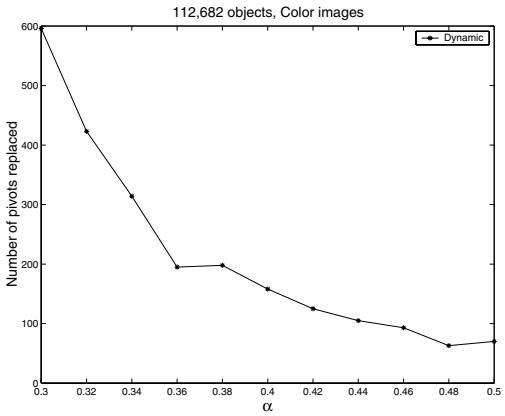
4.3 Search efficiency

Our second hypothesis was that our pivot replacement policy also makes the set of pivots more efficient. In Figure 4, we can observe the search performance in collections NASA and COLOR with both SSS and Dynamic. The results show that Dynamic is clearly more efficient than SSS in the collections of images. Figure 5 shows the results obtained in the SPANISH dataset. In this case the more efficient technique depends on the value of α, although there is no much difference between the two techniques. However, note that Dynamic is always better for similar sizes of the set of pivots, which means that the selected pivots with this method are better ones.

The dynamic method maintains the efficient pivots se-

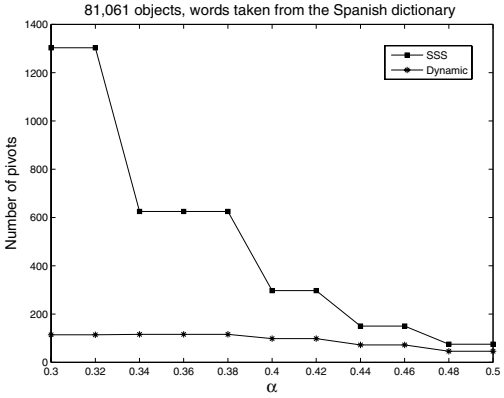


(a) Number of pivots in terms of α

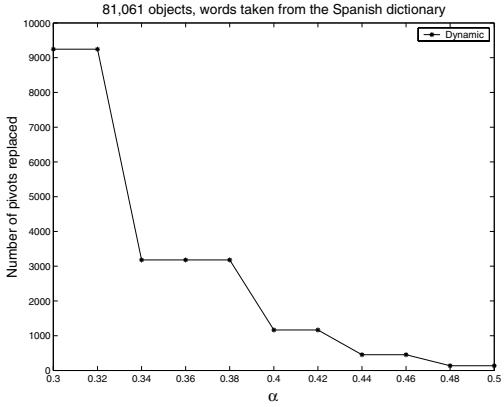


(b) Number of pivots replaced in terms of α

Figure 2. Size of the set of pivots with the COLOR dataset



(a) Number of pivots in terms of α



(b) Number of pivots replaced in terms of α

Figure 3. Size of the set of pivots with the SPANISH dataset

lected by SSS in the index, but it discards the unuseful ones. Thus, the internal complexity of the search is reduced since the query has to be compared with less pivots. This makes Dynamic to obtain better results than SSS even in the case of the COLOR dataset, in which SSS presented some performance problems.

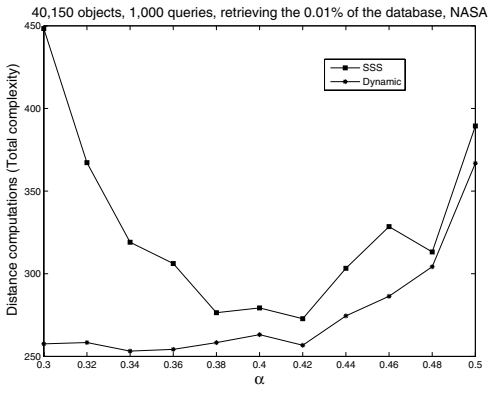
4.4 Index construction

In this section, we compare the cost of index construction with SSS and Dynamic. Dynamic has to perform additional distance computations each time an object becomes a pivot candidate to determine its contrubution to the efficiency of the set of pivots, and thus if it is selected as a pivot or not. Figure 6 shows the cost of index construction in the collections of images NASA and COLOR. In both cases SSS has a significantly smaller cost of construction (and therefore, smaller average cost of insertion). The index created with Dynamic is smaller than the one created with SSS, and this implies that the comparison of all the objects in the database against a new pivot is performed less times. This causes the difference between SSS and Dynamic to be smaller in COLOR than in NASA, because the size of COLOR is more than two times the size of NASA.

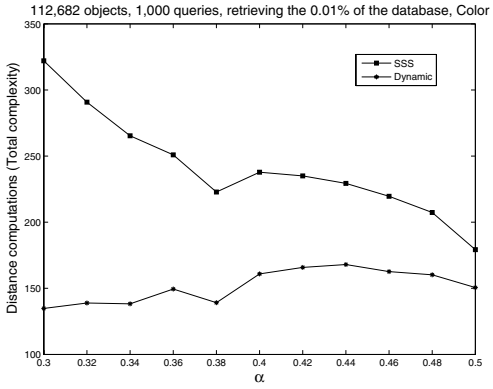
We also compared the cost of index construction with Dynamic and Dynamic-LCC (see Section 3.3). In Figure 7 (for space reasons we only show results for COLOR) we observe that Dynamic-LCC is not as good as Dynamic in terms of search efficiency, but a significant improvement is obtained in terms of index construction. As we already stated in Section 3.3, this was the expected result.

5 Conclusions and future work

We have presented a novel method for dynamically selecting pivots as objects are inserted in a metric database. Our techniqueis based on the method proposed in [5], but it replaces previously selected pivots that became redundant



(a) Distance computations in NASA



(b) Distance computations in COLOR

Figure 4. Evaluation of the index performance with SSS and Dynamic in collections of images

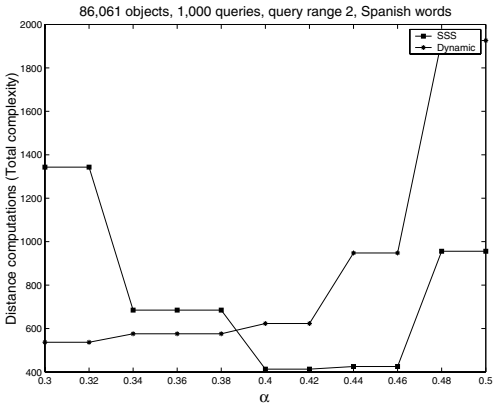


Figure 5. Evaluation of the index performance with SSS and Dynamic in collections of words

or unuseful compared to other selected pivots. The method use the efficiency criterion proposed in [2] to determine the contribution of each selected pivot. If a newly inserted object in the database is a pivot candidate (i.e., it is significantly far away from previously selected pivots), the selection algorithm computes its contribution and decides if it should be added to the actual set of pivots, or if it should replace the actual worst pivot in the pivot set (the one with smaller contribution).

The experimental evaluation showed that our method improves significantly over previously proposed method for different metric databases. The new selection technique is consistently better than the SSS algorithm for a wide range of values of α, and produces smaller and better sets of pivots, thus not only improving the efficiency of similarity queries but also the space used by the pivot-based index. As the value of α gets smaller, the dynamic selection replaces more pivots instead of adding them, thus mantaining a low internal complexity while keeping good pivots.

Future work involves improving the algorithm Dynamic-LCC and testing other policies for pivot replacement. Also, it would be interesting to study if inserted objects that were not considered pivot candidates (because they were not far away from previously selected pivots) could become good pivot candidates after a pivot is replaced. This could imply running several passes over the dataset, until a stopping condition (e.g., no new pivot candidates were found) holds.

References

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

[2] B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003. Elsevier.

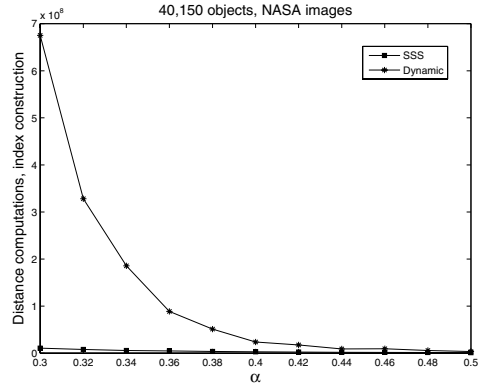
[3] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33:273–321, 2001.

[4] 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.

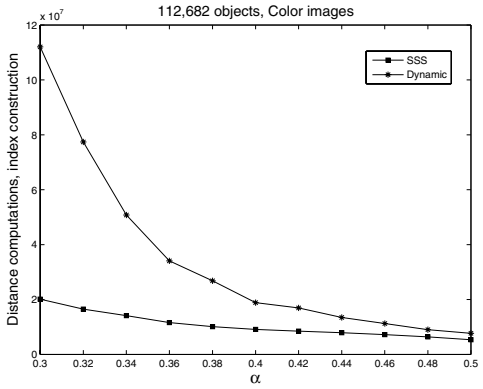
[5] O. Pedreira and N. R. Brisaboa. Spatial selection of sparse pivots for similarity search in metric spaces. In *SOFSEM 2007: 33rd Conference on Current Trends in Theory and Practice of Computer Science*, LNCS (4362), pages 434–445, 2007.

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

[7] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity search. The metric space approach*, volume 32. Springer, 2006.



(a) Cost of index construction in NASA



(b) Cost of index construction in COLOR

Figure 6. Evaluation of the cost of index construction with SSS and Dynamic in collections of images

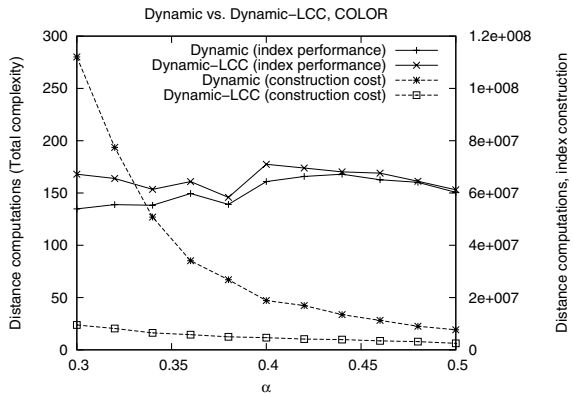


Figure 7. Comparison of Dynamic and Dynamic-LCC in efficiency and index construction

Algorithm 3: Function $computeVictim$

Input: $\mathbb{U}$, $Pivots$, A
Output: $victim$, $contributionVictim$, $MaxD$, $Pairs$

$Pairs \leftarrow \emptyset$;
// Select randomly A pairs of objects from $\mathbb{U}$
for i *from* 1 *to* A **do** $Pairs \leftarrow$ $Pairs \cup$ random pair of objects $(x, y) \in \mathbb{U} \times \mathbb{U}$;
// Initialize array with contribution of each pivot
for i *from* 1 *to* $|Pivots|$ **do** $contribution[i] \leftarrow 0$;
// Initialize array with distances in pivot space
for i *from* 1 *to* A **do** $MaxD[i] \leftarrow 0$;
// Compute contributions
for i *from* 1 *to* A **do**
  $(x, y) \leftarrow i^{th}$ pair of objects in $Pairs$;
  // Compute best pivot for row of (x, y)
  $MaxD[i] \leftarrow \max_{j=1}^{|Pivots|} |d(x, p_j) - d(y, p_j)|$;
  $indexMax \leftarrow$ $\arg\max_{j=1}^{|Pivots|} |d(x, p_j) - d(y, p_j)|$;
  // Compute second best pivot for row of (x, y)
  $max2 \leftarrow$ $\max_{j=1,\ j \neq indexMax}^{|Pivots|} |d(x, p_j) - d(y, p_j)|$;
  // Add contribution for best pivot of row
  $contribution[indexMax] \leftarrow$ $contribution[indexMax] + MaxD[i] - max2$;
// Compute victim and its contribution
$victim \leftarrow \arg\min_{i=1}^{|Pivots|} contribution[i]$;
$contributionVictim \leftarrow \min_{i=1}^{|Pivots|} contribution[i]$;
return $(victim,\ contributionVictim,\ MaxD,\ Pairs)$

Algorithm 4: Function $computeContribution$

Input: $u \in \mathbb{X}$, A, $MaxD$, $Pairs$
Output: $contributionNew$

$contributionNew \leftarrow 0$;
for i *from* 1 *to* A **do**
  $(x, y) \leftarrow i^{th}$ pair of objects in $Pairs$;
  $diff \leftarrow |d(x, u) - d(y, u)|$;
  if $diff > MaxD[i]$ **then**
    $contributionNew \leftarrow$ $contributionNew + diff - MaxD[i]$;
return $contributionNew$