Pivot Selection Techniques for Proximity Searching in Metric Spaces

Benjamin Bustos [a,b,1], Gonzalo Navarro [b,2], Edgar Chávez [c,3]

[a] *Department of Computer and Information Science, University of Konstanz, Universitaetstrasse 10, 78457 Konstanz, Germany.*

[b] *Center for Web Research, Department of Computer Science, University of Chile, Blanco Encalada 2120, Santiago, Chile.*

[c] *Escuela de Ciencias Físico-Matemáticas, Universidad Michoacana, Edificio "B", Ciudad Universitaria, Morelia, Mich. México.*

Abstract

With few exceptions, proximity search algorithms in metric spaces based on the use of *pivots* select them at random among the objects of the metric space. However, it is well known that the way in which the pivots are selected can drastically affect the performance of the algorithm. Between two sets of pivots of the same size, better chosen pivots can largely reduce the search time. Alternatively, a better chosen small set of pivots (requiring much less space) can yield the same efficiency as a larger, randomly chosen, set. We propose an efficiency measure to compare two pivot sets, combined with an optimization technique that allows us to select good sets of pivots. We obtain abundant empirical evidence showing that our technique is effective, and it is the first that we are aware of in producing consistently good results in a wide variety of cases and in being based on a formal theory. We also show that good pivots are outliers, but that selecting outliers does not ensure that good pivots are selected.

Key words: Metric databases, range queries, pivot based indexing, nearest neighbour search.

Email addresses: bustos@informatik.uni-konstanz.de (Benjamin Bustos), gnavarro@dcc.uchile.cl (Gonzalo Navarro), elchavez@fismat.umich.mx (Edgar Chávez).

[1] On leave from the Department of Computer Science, University of Chile. Partially funded by German Science Foundation (DFG) project No. KE 740/6-1 of the strategic research initiative SPP 1041.

[2] Funded by Millenium Nucleus Center for Web Research, Grant P01-029-F, Mideplan, Chile.

[3] Supported in part by CYTED VII.19 RIBIDI Project 58000.

1 Introduction

The concept of "proximity" searching has applications in a vast number of fields, for example: multimedia databases, image quantization and compression, text retrieval, computational biology, function prediction, just to name a few. In pattern recognition, proximity searching is used to implement function approximators, which classify a sample object according to the labeling of its nearest neighbors already classified. On the image processing scope, the detection of similar deformations of organs can be used for medical diagnosis purposes, and there are also applications on forensic investigation (e.g., tool and shoe marks identification).

All those applications have in common that the objects of the database form a *metric space* [7], that is, it is possible to define a positive real-valued function d among the objects, called *distance* or *metric*, that satisfies the properties of *strict positiveness* ($d(x,y) \geq 0$ and $d(x,y) = 0 \Leftrightarrow x = y$), *symmetry* ($d(x,y) = d(y,x)$), and the *triangle inequality* ($d(x,z) \leq d(x,y) + d(y,z)$). For example, a *vector space* $\mathbb{R}^t$ is a particular case of metric space, where the objects are tuples of t real numbers and the distance function belongs to the L_s family, defined as $L_s(\vec{x},\vec{y}) = \left(\sum_{1\leq i\leq t} |x_i - y_i|^s\right)^{1/s}$, $\vec{x},\vec{y} \in \mathbb{R}^t$. For example, L_1 is called *Manhattan distance*, L_2 is the *Euclidean distance* and $L_\infty = \max_{1\leq i\leq t} |x_i - y_i|$ is called the *maximum distance*.

In general, the distance d is considered expensive to compute (e.g., comparing two fingerprints), and in many applications d is so costly that the extra CPU time or even I/O time costs can be neglected. For this reason, in this paper the complexity of the algorithms will be measured as the *number of distance computations performed*. The goal of proximity search algorithms is to build an *index* of the database in advance and later perform proximity queries using this index, avoiding a full scan of the database. Many of these algorithms are based on the use of *pivots* [7,9], which are distinguished objects from the database. These pivots are used, together with the triangle inequality, to filter out objects of the database without measuring their actual distance to the query, hence saving distance computations while answering the query.

Almost all proximity search algorithms based on pivots choose them randomly among the objects of the database. However, it is well known that the way pivots are selected affects the search performance [10,7,8]. Some heuristics to choose pivots better than at random have been presented, but in general all of them try to choose objects that are *far away* from each other. For example, in [10] is proposed to choose objects that maximize the sum of distances between pivots previously chosen (see Section 5.4 for more details), in [13] is proposed an heuristic based on the second moment of the distance distribution which selects objects that are far away, and in [3] is proposed a greedy heuristic to select objects that are the farthest apart (note that this last structure does not

select pivots, but "split points"). However, these heuristics only work in specific metric spaces and have a bad behavior in others. In $\mathbb{R}^t$ with the Euclidean metric, it is shown in [8] that it is possible to find an optimal set of $t+1$ pivots selecting them as the vertices of a sufficiently large regular t-dimensional simplex containing all the objects of the database, but unfortunately this result does not apply to general metric spaces.

In this paper, we present an efficiency criterion to compare two pivot sets, which is based on the distance distribution of the metric space. Then, we present a selection technique based on this criterion to select a good set of pivots. We show empirically that this technique effectively selects good sets of pivots in a variety of synthetic and real-world metric spaces. Our technique is the first we are aware of in producing consistently good results in a wide variety of cases and in being based on a formal theory. Also, we show that good pivots have the characteristic to be outliers, that is, good pivots are objects far away from each other and from the rest of the objects of the database, but an outlier does not have always the property to be a good pivot.

The paper is organized as follows: In Section 2 we describe the canonical search algorithm based on pivots. In Section 3 we propose our efficiency estimator. In Section 4 we describe some optimization techniques, based on the efficiency estimator, for selecting good set of pivots. Section 5 presents the experimental results with synthetic and real-world datasets. Finally, we present some conclusions in Section 6.

2 Basic proximity search algorithm using pivots

Let $(\mathbb{X}, d)$ be a metric space, where $\mathbb{X}$ is the universe of valid *objects* and d is the metric of the space, and let $\mathbb{U} \subseteq \mathbb{X}$ be the set of objects or database, $|\mathbb{U}| = n$. Given a query object $q \in \mathbb{X}$, a *range query* $(q, r)_d$ is defined as the objects in $\mathbb{U}$ that are within distance r to q, that is $(q, r)_d = \{u \in \mathbb{U},\ d(u, q) \leq r\}$.

Given a query $(q, r)_d$ and a set of k pivots $\{p_1, \ldots, p_k\}, p_i \in \mathbb{U}$, by the triangle inequality it follows that $d(p_i, x) \leq d(p_i, q) + d(q, x)$, and also that $d(p_i, q) \leq d(p_i, x) + d(x, q)$ for any $x \in \mathbb{X}$. From both inequalities, it follows that a lower bound on $d(q, x)$ is $d(q, x) \geq |d(p_i, x) - d(p_i, q)|$. The objects $u \in \mathbb{U}$ of interest are those that satisfy $d(q, u) \leq r$, so all the objects that satisfy the *exclusion condition* (1) can be excluded, without actually evaluating $d(q, u)$.

$$|d(p_i, u) - d(p_i, q)| > r \quad \text{for some pivot } p_i \tag{1}$$

The index consists of the kn precomputed distances $d(p_i, u)$ between every pivot and every object of the database. Therefore, at query time it is necessary

to compute the k distances between the pivots and the query q, $d(p_i, q)$, in order to apply the exclusion condition (1). Those distance calculations are known as the *internal complexity* of the algorithm, and this complexity is fixed if there is a fixed number of pivots. The list of objects $\{u_1, \ldots, u_m\} \subseteq \mathbb{U}$ that cannot be discarded with the exclusion condition (1), known as the *object candidate list*, must be checked directly against the query. Those distance calculations $d(u_i, q)$ are known as the *external complexity* of the algorithm.

The total complexity of the search algorithm is the sum of the internal and external complexity, $k + m$. Since one increases and the other decreases (or at least does not increase) with k, it follows that there is an optimum k^* that depends on the tolerance range of the query, r. In practice, however, k^* is so large that one cannot store the k^*n distances, and the index simply uses as many pivots as space permits.

There are many proximity search algorithms in metric spaces that are based on pivots, such as *Burkhard-Keller Tree* [4], *Fixed-Queries Tree* (FQT) [1], *Fixed-Height FQT* (FHQT) [1], *Fixed Queries Array* (FQA) [5], *Vantage Point Tree* [13], *Multi Vantage Point Tree* [2], *Excluded Middle Vantage Point Forest* [14], *AESA* [12], *Linear AESA* (LAESA) [10] and *Spaghettis* [6].

3 Efficiency criterion

Depending on how pivots are selected, they can filter out less or more objects. We define in this section a criterion to tell which from two pivot sets is expected to filter out more and hence reduce the number of distance computations carried out during a range query. Since the internal complexity is fixed, only the external complexity can be reduced, and this is achieved by making the candidate object list as short as possible.

A set of k pivots $\{p_1, p_2, \ldots, p_k\}$, $p_i \in \mathbb{U}$, defines a space $\mathbb{P}$ of distance tuples between pivots and objects from $\mathbb{U}$. The mapping of an object $u \in \mathbb{U}$ to $\mathbb{P}$, which will be denoted $[u]$, is carried out as $[u] = (d(u, p_1), d(u, p_2), \ldots, d(u, p_k))$. Defining the metric $D_{\{p_1,\ldots,p_k\}}([x], [y]) = \max_{1 \le i \le k} |d(x, p_i) - d(y, p_i)|$, it follows that $(\mathbb{P}, D)$ is a metric space, which turns out to be $(\mathbb{R}^k, L_\infty)$. Given a range query (q, r), the exclusion condition (1) in the original space $\mathbb{U}$ becomes (2) for the new metric space $(\mathbb{P}, D)$.

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

To achieve a candidate object list as short as possible, the probability of (2) should be as high as possible. One way to do this is to maximize the mean of the distance distribution of D, which will be denoted μ_D. Hence, we will say

that $\{p_1, \ldots, p_k\}$ is a better set of pivots than $\{p'_1, \ldots, p'_k\}$ when:

$$\mu_{D_{\{p_1,\ldots,p_k\}}} > \mu_{D_{\{p'_1,\ldots,p'_k\}}} \tag{3}$$

Another possibility for maximizing the probability of (2) is trying to reduce the variance of the distribution of D at the same time μ_D is maximized. However, we will show in Section 5.3 that in practice this approach does not work as well as just maximizing μ_D.

An estimation of the value of μ_D is obtained in the following way: A pairs of objects $\{(a_1, a'_1), (a_2, a'_2), \ldots, (a_A, a'_A)\}$ from $\mathbb{U}$ are chosen at random. All the pairs of objects are mapped to space $\mathbb{P}$, obtaining the set $\{D_1, D_2, \ldots, D_A\}$ of distances D between every pair of objects. The value of μ_D is estimated as $\mu_D = \frac{1}{A} \sum_{1 \leq i \leq A} D_i$. It follows that $2k$ distance computations are needed to compute distance D for each pair of objects using k pivots. Therefore, $2kA$ distance computations are needed to estimate μ_D.

4 Pivot selection techniques

Now we present three pivot selection techniques based on the efficiency criterion (3). Each technique has a cost measured in number of distance computations at index construction time. As we do more work in optimizing the pivots, better pivots are obtained. When comparing two techniques, we give them the same amount of work to spend. We describe the optimization cost of each technique.

These selection techniques can be directly adapted to work with algorithms that use a fixed number of pivots, such as *FHQT*, *FQA*, *LAESA* and *Spaghettis*. They can also be adapted to the other pivot based algorithms.

i. *Selection of N random groups.*

   N groups of k pivots are chosen at random among the objects of $\mathbb{U}$, and μ_D is calculated for each of this groups of pivots. The group that has the maximum μ_D value is selected.

   Optimization cost: Since the value of μ_D is estimated N times, the total optimization cost is $2kAN$ distance computations.

ii. *Incremental selection.*

   A pivot p_1 is selected from a sample of N objects of $\mathbb{U}$, such that that pivot alone has the maximum μ_D value. Then, a second pivot p_2 is chosen from another sample of N objects of $\mathbb{U}$, such that $\{p_1, p_2\}$ has the maximum μ_D value, considering p_1 fixed. The third pivot p_3 is chosen from another sample of N objects of $\mathbb{U}$, such that $\{p_1, p_2, p_3\}$ has the maximum μ_D value, considering p_1 and p_2 fixed. The process is repeated

until k pivots have been chosen.

Optimization cost: If the distances $D_{\{p_1,\ldots,p_{i-1}\}}([a_r],[a'_r])$, $1 \leq r \leq A$, are kept in an array, it is not necessary to redo all the distance computations to estimate μ_D when the i^{th} pivot is added. It is enough to calculate $D_{\{p_i\}}([a_r],[a'_r])$, $1 \leq r \leq A$, because it follows that $D_{\{p_1,\ldots,p_i\}}([a_r],[a'_r]) = \max(D_{\{p_1,\ldots,p_{i-1}\}}([a_r],[a'_r]), D_{\{p_i\}}([a_r],[a'_r]))$. Therefore, only $2NA$ distance computations are needed to estimate μ_D when a new pivot is added. Since the process is repeated k times, the total optimization cost is $2kAN$ distance computations.

iii. *Local optimum selection.*

A group of k pivots is chosen at random among the objects of the database. The matrix $M(r,j) = D_{p_j}([a_r],[a'_r])$, $1 \leq r \leq A$, $1 \leq j \leq k$, is calculated using the A pairs of objects. It follows that $D([a_r],[a'_r]) = \max_{1\leq j\leq k} M(r,j)$ for every r, and this can be used to estimate μ_D. Also, it must be kept for each row of M the index of the pivot where the maximum value is, which will be denoted r_{max}, and the second maximum value, denoted r_{max2}. The *contribution contr* of the pivot p_j is the sum over the A rows of how much does p_j help increase the value of $D([a_r],[a'_r])$, that is, $contr = M(r,r_{max}) - M(r,r_{max2})$ if $j = r_{max}$ for that row, and $contr = 0$ otherwise. The pivot whose contribution to the value of μ_D is minimal with respect to the other pivots is marked as the *victim*, and it is replaced, when possible, by a better pivot selected from a sample of X objects of the database. The process is repeated N' times.

Optimization cost: The construction cost of the initial matrix M is $2Ak$ distance computations. The search cost of the victim is 0, because no extra distance computations are needed, all information is in M. Finding a better pivot from the X objects sample costs $2AX$ distance computations, and the process is repeated N' times, so the total optimization cost is $2A(k+N'X)$ distance computations. Considering $kN = k + N'X$, i.e., $N'X = k(N-1)$, the optimization cost is $2AkN$ distance computations.

Note that it is possible to exchange the values of N' and X while maintaining the optimization cost. In the experiments we use two possible value selections: $(N' = k) \wedge (X = N-1)$ (called *local optimum A*) and $(N' = N-1) \wedge (X = k)$ (called *local optimum B*). We also try with another value selection, $N' = X = \sqrt{k(N-1)}$, but the obtained result does not show any improvement on the performance of the algorithm.

5 Experimental results

We have tested the selection techniques on a synthetic set of random points in a k-dimensional vector space treated as a metric space, that is, we have not used the fact that the space has coordinates, but treated the points as abstract objects in an unknown metric space. The advantage of this choice

is that it allows us to control the exact dimensionality we are working with, which is very difficult to do in general metric spaces. The points are uniformly distributed in the unitary cube, our tests use the L_2 (Euclidean) distance, the dimension of the vector space is in the range $2 \leq dim \leq 14$, the database size is $n = 100,000$ (except when otherwise stated) and we perform range queries returning 0.01% of the total database size, taking an average from 10,000 queries. We show the results of the experiments with real-world datasets in Section 5.5.

About the parameters A and N of the optimization cost: Our experiments show that, given an amount of work to spend, it is better to have a high value of A and a low value of N. This indicates that it is worth to make a good estimation of μ_D, while small samples of candidate objects suffice to obtain good sets of pivots. For the experiments in this section, these parameters have fixed values as follows: $A = 100,000$ and $N = 50$.

5.1 Comparison between the selection techniques

Figure 1 shows the comparison between all the selection techniques, when varying the number of pivots and keeping the dimension of the space fixed. These results show that the incremental selection technique and the local optimum A technique obtain the best performance in practice. Local optimum B works well only in spaces with low dimension and with few pivots, obviously influenced by the setting of parameter N. Selection of N random groups shows little improvement over random selection in all cases.

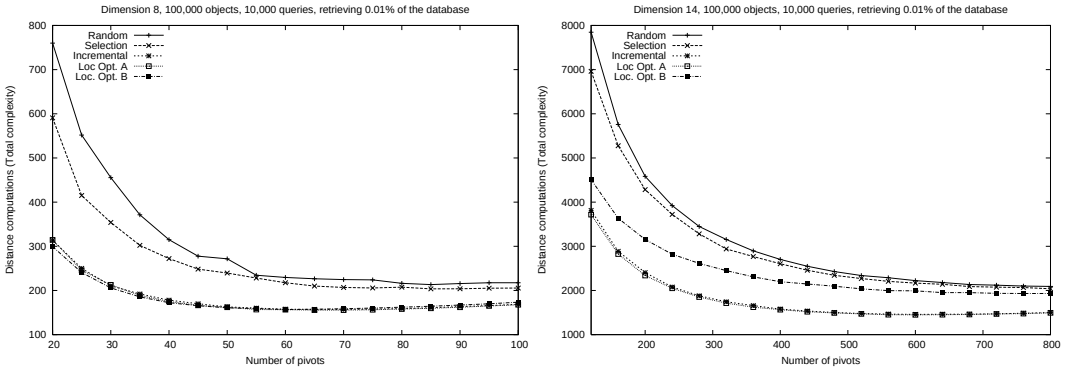

Fig. 1. Comparison between selection techniques in random vector spaces of dimension 8 (left) and dimension 14 (right).

Although the incremental and local optimum A techniques give almost the same efficiency, the incremental selection have some advantages that makes it our favorite pivot selection technique: It is a much simpler technique and it allows us to easily add more pivots to the index. Also, the only way to determine the optimum number of pivots k^*, for a fixed tolerance range, is calculating an average of the total complexity of the algorithm for different

values of k, where k^* is equal to the value of k which minimizes the total complexity. That is, it is worth to add pivots to the index until the total complexity cease to improve. The incremental selection allows us to add more pivots to the index at any time without doing all the optimization work again, if the distances $D_{\{p_1,\ldots,p_k\}}([a_r],[a'_r])$, $\forall r \in 1 \ldots A$ are kept. On the other hand, selection of N random groups and local optimum selection techniques must redo all the optimization work to obtain a new set of pivots, because these techniques cannot take advantage of the work done previously. For this reason, it is much easier to calculate the optimum number of pivots k^* using the incremental selection technique.

5.2 Comparison between random and good pivots

Figure 2 shows a comparison for internal and total complexity (see Section 2) between random and incremental selection when using the optimum number of pivots for each technique. The left plot shows a comparison when varying the dimension of the space. Since k^* is equal to the internal complexity of the algorithm, it follows that not only the optimum number of pivots is lower when using the incremental selection, but so is also the total complexity of the algorithm. The right plot shows a comparison in a vector space of dimension 8 varying the database size. Again we obtain that the optimum number of pivots and the total complexity of the algorithm is lower when using the incremental selection. The results obtained show that the incremental selection technique effectively produces good sets of pivots.

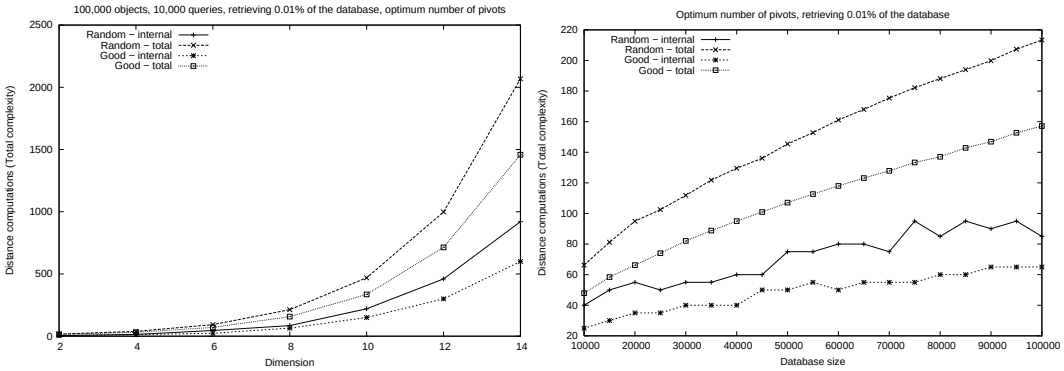

Fig. 2. Comparison between random and good pivots when varying the dimension (left) and the database size (right).

The profit when using k^* pivots with incremental selection seems low in high dimensional spaces. However, consider that much fewer pivots (i.e., less memory) are needed to obtain the same result than with random selection. For example, in a vector space of dimension 14 the optimum number of pivots using random selection is 920, while incremental selection only needs 280 pivots to achieve a better total complexity than random selection in its optimum, hence saving almost 70% of the memory used in the index.

The optimization cost used in this experiments, given by parameters A and N, may seem a little bit high. However, it is possible to obtain good results with a fraction of the used optimization cost. Figure 3 (left) shows the results of an experiment in a uniform vector space of dimension 8, using the optimum number of good pivots and varying parameter A from 100 to 100,000. The results shows that for values higher than 10,000 the improvement is negligible. Even when using a value as low as $A = 100$, we observed an improvement of 12% in the total complexity over random pivots.

5.3 Alternative efficiency estimators

As stated in Section 3, another possibility for maximizing the probability of the exclusion condition (2) is trying to reduce the variance of the distribution of D, σ_D^2, at the same time μ_D is maximized. To accomplish this, we try to maximize the *intrinsic dimension* of the space $\mathbb{P}$, defined in [7] as $\mu_D/2\sigma_D^2$. We also tried another efficiency estimator: to maximize the minimum value of the distance distribution of D. This aims to shift to the right the distance distribution as much as possible.

Figure 3 (right) shows the results of an experiment in a synthetic vector space of dimension 8, comparing the two additional efficiency estimators against the original one. The figure shows that the original estimator selects best sets of pivots compared with the others, which even cannot do better than random selection for more than 40 pivots.

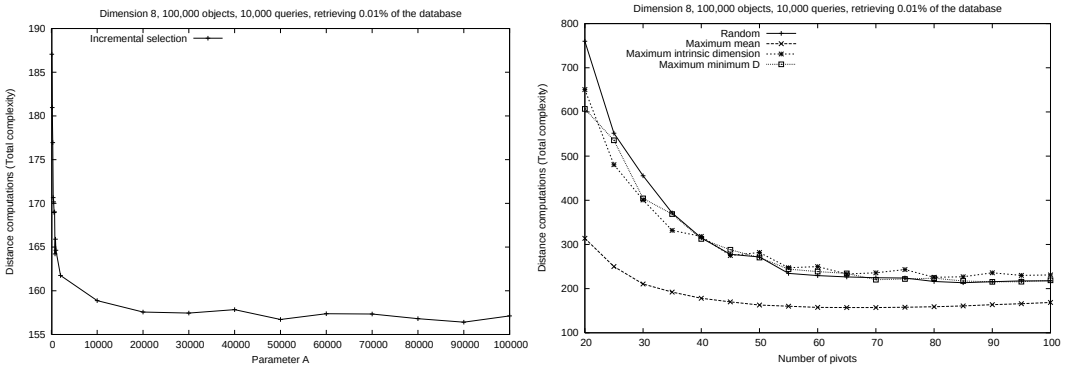

Fig. 3. Efficiency of the selection technique when varying parameter A (left) and comparison between different efficiency estimators (right).

5.4 Properties of a good set of pivots

When studying the characteristics of the good sets of pivots, we found that good pivots are *far away* from each other, i.e., the mean distance between pivots is higher than the mean distance between random objects of the metric

space, and also good pivots are *far away* from the rest of the objects of the metric space. The objects that satisfy these properties are called *outliers*. It is clear that pivots *must be far away from each other*, because two very close pivots give almost the same information for discarding objects. This is in accordance with previous observations [8,13,3].

Then, it can be assumed that good pivots are outliers, so a new selection technique could be as follows: use the same incremental selection method with the new criterion of selecting objects which maximize the sum of the distances between the pivots previously chosen, selecting the first pivot at random. This technique will be called *outliers selection*, and it was already proposed in [10]. It carries out $(i-1)N$ distance computations when the i^{th} pivot is added, where N is the size of the sample of objects from where a new pivot is selected. Hence, the optimization cost of this selection technique is $\frac{k(k-1)}{2}N$.

It is important to note that outliers selection *do not use the efficiency criterion described in section 3*, because this alternative selection technique maximizes the mean distance in the original space and the efficiency criterion maximizes the mean of distance D. These criteria do not always go together.

Figure 4 shows the result obtained when comparing incremental and outliers selection techniques in random vector spaces. The figures show that the outliers selection has slightly better performance than the incremental selection. This result can lead to think that outliers selection is the best pivot selection technique, but in the next section we will see that this assumption is not true for general metric spaces.

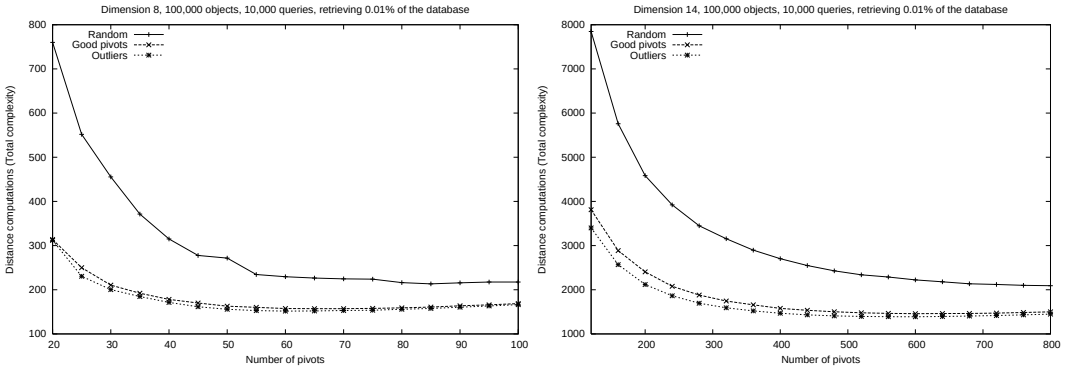

Fig. 4. Comparison between incremental and outliers selection techniques in random vector spaces of dimension 8 (left) and dimension 14 (right).

5.5 Real-world examples

We present four examples of the use of the incremental selection and the outliers selection, where the objects of the metric space are not uniformly dis-

tributed. The incremental selection technique was used to select good pivots. We also tested the local optimum A technique with these databases, obtaining slightly better results compared with the incremental selection, but we prefer to use this technique over local optimum for the reasons stated in Section 5.1.

Figure 5 (left) shows the result of an experiment in a 30-dimensional vector space, where the elements have a Gaussian distribution, that is, the elements form clusters. The space is formed by 100 clusters, each of them centered at a random point of the space, and the variance for each coordinate is 0.001. The result shows that both good pivots and outliers improve the performance of the search algorithm in comparison with random pivots, but good pivots perform better for few pivots.

Figure 5 (right) shows the results of the experiment over a string space, that is, the objects of the database were strings taken from an English dictionary of 69,069 terms, and 10% of the database was used as the query set. The distance function used was the *edit distance* (the minimum number of character insertions, deletions and substitutions to make two strings equal), and the tolerance range was $r = 2$, which retrieves an average of 0.02% of the database size per query. In this case, the incremental selection improves more the performance of the algorithm, with respect to random pivots, compared to the outliers selection.

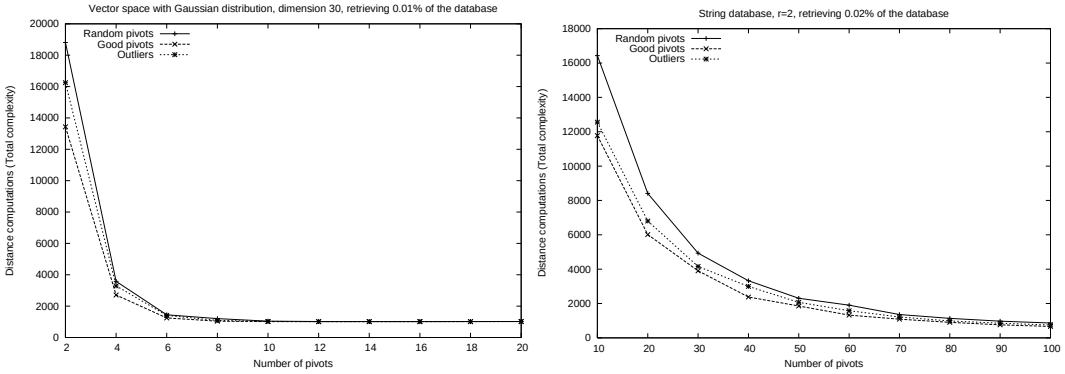

Fig. 5. Experiments with a vector space with Gaussian distribution (left) and a string database (right).

Figure 6 (left) shows the results of the experiment when the objects of the database are a set of 40,700 images from NASA archives [11]. Those images were transformed into 20-dimensional vectors, and 10% of the database was defined as the query set. We used a tolerance range which returns on average 0.10% of the objects of the database per query. The figure shows that for more than 25 pivots the outliers selection technique has worse performance that the random selection, while incremental selections always performs better.

Figure 6 (right) shows the result of the experiment with a database of 112,682 color images, where each image is represented by a feature vector of dimension 112. A 10% of the database was used as the query set. The result shows that

good pivots perform better than random pivots, but outliers perform worse than random pivots. In fact, with less than 40 pivots the results were an order of magnitude worse than with random pivots.

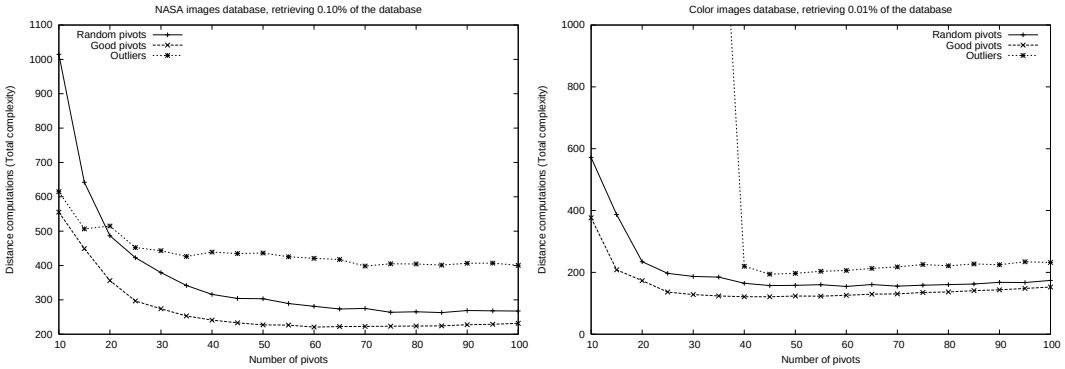

Fig. 6. Experiments with the NASA images (left) and a color image databse (right).

The last two results are in contrast with those obtained on uniformly distributed vector spaces.

6 Conclusions

We have defined an efficiency criterion to compare two sets of pivots, and we have shown experimentally that this criterion consistently selects good sets of pivots in a variety of synthetic and real-world metric spaces, reducing the total complexity of pivot-based proximity searching when answering range queries. Our efficiency criterion is based on a formal theory, that takes in account the distance distribution of the mapped space defined by the selected pivots. We consider this formalism crucial, in contrast to simple heuristics, to consistently obtain good results in a wide scope as the one of metric spaces.

We presented three different pivot selection techniques, which use the efficiency criterion defined, and we showed that the so-called *incremental selection* is the best method in practice. We have found that good pivots have the property to be outliers, but outliers are not necessarily good pivots. It is interesting to note that outliers sets have good performance in uniformly distributed vector spaces, but have bad performance in general metric spaces, even worse than random selection in some cases. This result leads to questioning if it is valid to test pivot selection techniques in uniformly distributed vector spaces.

Acknowledgements. We want to thank Prof. Thomas Seidl from Aachen University for kindly allowing us to use the color image database for experiments.

References

[1] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198–212, 1994.

[2] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 357–368, 1997. Sigmod Record 26(2).

[3] S. Brin. Near neighbor search in large metric spaces. In *Proc. 21st Conference on Very Large Databases (VLDB'95)*, pages 574–584, 1995.

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

[5] E. Chávez, J.Marroquín, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools and Applications (MTAP)*, 14(2):113–135, 2001.

[6] E. Chávez, J. Marroquín, and R. Baeza-Yates. Spaghettis: an array based algorithm for similarity queries in metric spaces. In *Proc. String Processing and Information Retrieval (SPIRE'99)*, pages 38–46. IEEE CS Press, 1999.

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

[8] A. Faragó, T. Linder, and G. Lugosi. Fast nearest-neighbor search in dissimilarity spaces. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(9):957–962, 1993.

[9] R. Santos Filho, A. Traina, C. Traina Jr., and C. Faloutsos. Similarity search without tears: The OMNI family of all-purpose access methods. In *ICDE*, pages 623–630, 2001.

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

[11] Sixth DIMACS Implementation Challenge: Available Software. `http://www.dimacs.rutgers.edu/Challenges/Sixth/software.html`.

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

[13] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311–321, 1993.

[14] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *DIMACS Implementation Challenge, ALENEX'99*, Baltimore, MD, 1999.