Optimal Pivots to Minimize the Index Size for Metric Access Methods

Luis G. Ares, Nieves R. Brisaboa, María F. Esteller, Óscar Pedreira, Ángeles S. Places
Database Laboratory
Universidade da Coruña
A Coruña, Spain
{*lgares,brisaboa,mfesteller,opedreira,asplaces*}*@udc.es*

Abstract

We consider the problem of similarity search in metric spaces with costly distance functions and large databases. There is a trade-off between the amount of information stored in the index and the reduction in the number of comparisons for solving a query. Pivot-based methods clearly outperform clustering-based ones in number of comparisons, but their space requirements are higher and this can prevent their application in real problems. Therefore, several strategies have been proposed that reduce the space needed by pivot-based methods, as BAESA, FQA or KVP. In this paper, we analyze the usefulness of pivots depending on their proximity to the object. As consequence of this analysis, we propose a new pivot-based method that requires an amount of space equal or very close to that needed by clustering-based methods. We provide experimental results that show that our proposal represents a competitive strategy to clustering oriented solutions when using the same amount of memory.

I. Introduction

The main goal of methods for searching in metric spaces is to reduce as much as possible the number of distance evaluations needed to solve a query, since this is the main component of the search cost. However, there are other factors that affect the overall search performance, as the space requirements for storing the index (in either primary or secondary memory) and the extra CPU time needed for loading and processing it. Most research in metric spaces has focused on the number of distance evaluations as a measure of efficiency, but these other factors become important when indexing large collections of data in real systems. This is especially important when searching contents generated by millions of users from the web.

Funded in part by "Ministerio de Educación y Ciencia" (PGE y FEDER) ref. TIN2006-15071-C03-03 and "Xunta de Galicia", ref. 2006/4.

According to [1], metric access methods can be generally classified in two groups: clustering-based and pivot-based methods. Clustering-based methods partition the space and try to discard whole regions during the search. Pivot-based methods store precomputed distances from each object in the database to a set of pivots, and these distances are used during the search to discard as many objects as possible from the result set. In the case of clustering-based methods, the index is usually a list or a tree that represents the partition of the space. They need a very small amount of memory for storing the index, and the extra CPU time is also very small. Pivot-based methods can outperform clustering-based ones in one or two orders of magnitude in terms of distance computations, although they need very significant amounts of space for storing the precomputed distances. Therefore, despite they are very efficient in distance computations, their memory requirements can prevent them to be applied in real applications. The reader not familiar with similarity search in metric spaces can find good introductions in [1] and [2].

Previous works have studied different strategies to reduce the amount of space used by pivot-based methods while trying to conserve their effectiveness for discarding objects from the result set. This can be done mainly by storing less information in the index (bucket and scope coarsening), or storing it with less precision (range coarsening). In both cases, the loss of information implies a loss of effectiveness for discarding objects. FQA [3] and BAESA [4] reduce the size of the index by storing the information with less precision. BAESA divides the range of possible values of the distance in a set of intervals and

stores the interval in which each distance falls using the minimum number of bits necessary. Methods as KVP [5] reduce the size of the index by storing only information they consider promising for the search.

In this paper, we analyze the effectiveness of each pivot for each object in the database and the resulting utility of storing that distance. We also study the effect of the initial pivot set in the index performance. Based on this analysis we propose a new pivot-based method that needs an amount of space very close or even smaller than the used by clustering-based algorithms. Our experimental evaluation shows that our proposal represents a competitive strategy when using the same amount of space.

The rest of the paper is organized as follows: Section II describes previous proposals for reducing the amount of space needed by pivot-based methods, and techniques for the selection of effective pivots. Section III poses the objectives of this work and describes our experimental setup. Section IV presents our analysis of pivot effectiveness. Section V presents the new pivot-based method we propose in this paper and its experimental evaluation. Finally, Section VI presents the conclusions of this work and potential lines of future work.

II. Background and Related Work

Suppose that (X, d) is a metric space, and $U \subseteq X$ a database or collection of objects of size $n = |U|$. Pivot-based methods select a set of m objects $P = \{p_1, \ldots, p_m\}$ from the database to be used as reference objects, called pivots. In the indexing phase, the distances $d(x_i, p_j)$ from the objects in the database to the pivots are computed and stored in an appropriate data structure. When given a query (q, r), the query object is compared with the pivots to obtain the distances $d(q, p_j)$. For each object $x_i \in U$, we can obtain a lower bound of its distance to the query using the triangle inequality. The object is discarded from the result set without comparing it with the query if $d(x_i, q) \geq |d(x_i, p_j) - d(p_j, q)| > r$ for any pivot $p_j \in P$. The search complexity is the sum of the internal complexity (comparisons of the query with the pivots) and the external complexity (comparisons of the query with objects that could not be discarded).

Existing methods differ in the information they store and the data structures they use. Methods like AESA [6] and LAESA [7] use a table to store the distances from objects to pivots, while others as BKT [8], FQT [9], FQA [3] or VPT [10] use tree-like structures.

There are two important issues for pivot-based methods: the reduction of the space requirements of the index, and the selection of the reference objects used as pivots.

A. Reducing the Space Requirements

The main drawback of pivot-based methods is the space and extra CPU time needed for loading and processing the information stored in the index. The goal of reducing the space needed to store the index is to make possible to use it in main memory or even in the cache of the processor, and therefore reducing the extra CPU time needed for processing it too. There are several strategies for reducing the space of this class of methods [1]:

- Range coarsening: The space required by the index can be reduced by storing the distances from objects to pivots with less precision. This is direct when working with discrete distances. For continuous distances, the range of possible values can be divided in several intervals. Instead of storing the exact distance between an object and a pivot, the index can store the interval in which each the distance falls. Since the distances are now less precise, the amount of objects discarded from the result will be smaller. Working with less precise distances coarsens the region of exclusion defined by each pivot for a given query, thus the name of range coarsening.
  This strategy is used by VPT [10] and MVPT [11] and can be easily applied in indexes like FQA [3], a compact representation of FHQT [9]. BAESA [4] is basically an AESA [6] index in which the range of distances is divided in 2^k intervals. Therefore each of the $n \times n$ distances of AESA is stored using k bits.
- Bucket coarsening: This strategy is thought for indexes with a tree-like structure. The idea is to stop creating more subtrees when they have a small enough number of elements. The size of the candidate list increases as this small areas are not further indexed, but the index stores less distances and needs thus less space.
- Scope coarsening: Indexes like AESA [6], LAESA [7], and SSS [12] store all the distances from each object in the database to each pivot. Another option consists in storing the distances from each pivot to a subset of objects in the database, thus coarsening the scope of action of the pivot. Since the index stores less distances, the chances of discarding an object from the result are smaller too.
  KVP [5], [13] is an example of scope coarsening. [5] showed that for a given object in the database, its nearest and furthest pivots are the ones with more chances of discarding it from the result. KVP prioritizes the pivots based on this criterion and, given a limited amount of space, stores only the most promising distances. [13] combines scope and range coarsening and reports KVP to obtain better results than previous methods.

B. Selecting Effective Pivots

Although most methods select the pivots at random, it has been shown that the specific set of pivots affects the search performance [14]. The position of each pivot with respect to each other and to the rest of objects in the database determines the index capacity for discarding objects from the result.

The more the pivots, the more the chances of discarding an object from the result. However, increasing the number of pivots also increases the internal complexity. Therefore, there is an optimal number of pivots that optimizes the trade-off between the internal and external complexities. The number of pivots selected also affects the space requirements of the index and the extra CPU time for processing it. Given two sets of pivots that show the same effectiveness, the smaller one is the best choice.

Previous works have proposed different ways of estimating the effectiveness of a pivot or set of pivots. Works as [15], [7], [10] indicated that good pivots should be far from each other and also far from the rest of objects of the database. Bustos et al. [14] defined a formal criterion for comparing the effectiveness of two set of pivots of the same size.

Despite the good results that can be obtained with these techniques, they are thought towards the selection of a set of pivots that are effective in average for all the objects of the database as a whole. They do not give any guidance on what is the best pivot for an object. Celik [5] showed empirically that given an object of the database and a query, the best pivots for that object are those which are either very near or very far from that object. This criterion is the only one that permits us to have an estimation of what should be the best possible pivot for a given object. However, [13] also mentions that one can come up with distributions for which far and close pivots are not the best choice.

Several techniques have been proposed for pivot selection: MaxMin [16] selects pivots maximizing the minimum distance between them; SFLOO [17] select pivots based on a measure of the loss of precision in the search; Spacing [18] selects pivots with minimum correlation and that maximize the distance between objects in the mapped pivot space; Incremental [14] incrementally selects a set of pivots that maximizes a criterion for comparing two sets of pivots; Maximum Pruning [19] selects pivots that maximize pruning based on a sample of queries.

Sparse Spatial Selection (SSS) [12][20] selects a set of pivots well distributed in the space. An important characteristic of SSS is that the selection is carried out dynamically. This is, when an object is inserted in the database, the algorithm determines if it should be a pivot or not. SSS considers that the new object is far enough if its distance to already selected pivots is greater than $M\alpha$, being M the maximum distance between any two objects and α a constant parameter that controls the density of pivots with which the space is covered. Therefore, the index adapts its structure and information as the database evolves when objects are inserted or removed from the database. In [12] is proven that using SSS, the number of pivots depends on the complexity of the space and not on the number of objects.

III. Objectives of this Work

In this paper, we pursue the following two objectives related to pivot-based methods for similarity search in metric spaces:

- First, to analyze the effectiveness of pivots for each object in the database. We carry out an empirical analysis with different synthetic and real collections of data.
- Second, based on the results of the analysis of the effectiveness of pivots, to reduce as much as possible the number of distances stored in the index for each object of the database.

In our experiments we used several collections of synthetic and real data available in the *Metric Spaces Library* [21]: UV08, UV10, UV12, and UV14 are collections of $100,000$ synthetic vectors with uniform distribution in a hypercube of dimension 8, 10, 12, and 14 respectively. English is a collection of $69,069$ words extracted from the English dictionary, and compared using the standard edit distance; Nasa is a collection of $40,150$ images extracted from the archives of image and video of the NASA, and represented by feature vectors of dimension 20, compared using the Euclidean distance of their feature vectors.

As usual, the experimental evaluation focused in range search. For English we worked with $r = 2$. For vector collections, the search radius was set to retrieve the 0.01% of the database, in average, for each query.

IV. Analysis of Pivot Effectiveness

We consider two issues concerning the selection of the most effective pivots for each object of the database: first, the initial set of pivots from which the most promising ones are selected; second, the effectiveness of the different pivots for each object.

Celik [5] shows that the nearest and furthest pivots of an object are the most promising ones for discarding it in a query evaluation. From this fact we deduce that SSS [12] is an effective way of selecting the pivots, since it guarantees that the pivots will be well distributed in the space. That is, other methods for pivot selection (random

Coll.	Random			SSS (optimal α)			SSS (α = 0.25, 4 dist.)			SSS (α = 0.25, 2 dist.)		
	Piv.	Space	Eval. d	Piv.	Space	Eval. d	Piv.	Space	Eval. d	Piv.	Space	Eval. d
UV08	85	29.1828	211.78	53	18.1963	141.43	494	2.7485	1231.66	494	1.3752	3094.13
UV10	190	65.2321	468.23	176	60.4255	367.14	1461	2.7522	3011.61	1461	1.3789	5891.64
UV12	460	157.9303	998.13	250	85.8317	645.08	4303	2.7630	6803.09	4303	1.3897	9739.49
UV14	1000	343.3266	2077.44	491	168.5734	1381.64	10000	2.7848	14229.20	10000	1.4115	18160.91
English	200	27.5696	443.85	212	45.0553	354.89	3100	1.9089	7931.30	3100	0.9604	12373.45
Nasa	77	10.6143	276.34	55	7.4438	168.62	871	1.1061	1308.28	871	0.5547	1958.34

TABLE I. Search cost with different strategies for selecting the initial set of pivots.

is the obvious case) do not guarantee that each object will have pivots that are really near or far. In the worst case, all pivots can be at a half of the maximum distance from the object.

The next question we have to ask is that, among the pivots selected with SSS, what are the characteristics of those which are most useful for discarding an object from the result? Following the results of Celik [5], we know that those pivots have to be really near or far from the object. But, how many near and far pivots do we need for each object in order to be effective?

Table I shows the number of pivots ("Piv."), space ("Space", in MB), and distance evaluations ("Eval. d") obtained when using:

- SSS, storing all the distances from each object to each pivot ("SSS (optimal α)").
- A random pivot selection, storing all the distances from each object to each pivot ("Random").
- SSS, but storing only the distances to the two nearest and two furthest pivots for each object ("SSS ($\alpha = 0.25, 4\ dist$)").
- SSS, but storing only the distances to the nearest and furthest pivot for each object ("SSS ($\alpha = 0.25, 2\ dist$)").

This is almost a reproduction of the results of Celik [5], but we can observe that, in fact using two pivots, the efficiency is still better than the obtained with clustering-oriented methods, while the space has been dramatically reduced. For instance, in the case of UV12, the number of evaluations when using two pivots is about 15 times the number of comparisons when using all pivots, but the space needed when using all pivots is about 61 times the space when using only two. In the case of UV14, a more complex collection, the difference is bigger: the number of comparisons when using two pivots is about 13 times the number of comparisons when using all of them, but the space when using all the pivots is about 119 times the space needed when using only two. We can observe that in the most complex collections, such as UV14, English, or Nasa, when we pass from 4 to 2 pivots the effectiveness is reduced to a 75% while the space is reduced to a 50%. This is, the loss in effectiveness is not proportional to the gain in space. Therefore, we think that reducing dramatically

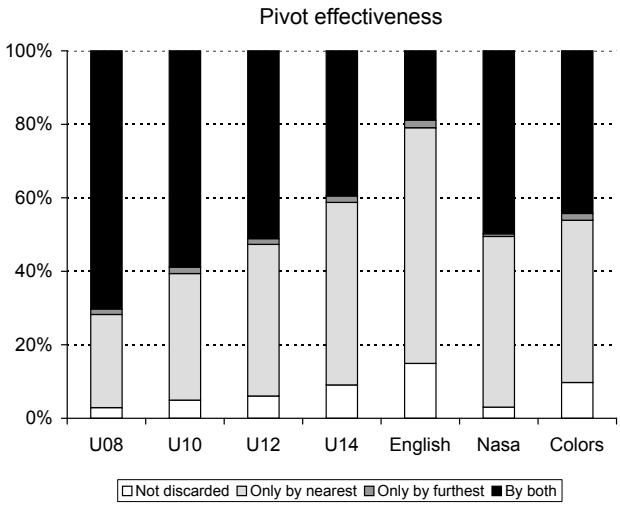

Fig. 1. Percentages of objects discarded by the nearest, furthest or both pivots.

the number of pivots to just 2 is a good idea when we want to reduce the space requirements of the index.

A second question we have to ask is if it is more effective to use the nearest or the furthest pivot for each object. Figure 1 shows for each collection the percentages of objects that could not be discarded ("Not discarded"), objects discarded only by their nearest pivot ("Nearest"), objects discarded only by their furthest pivot ("'Furthest"), and objects that could be discarded by either their nearest or furthest pivot ("Both"). In Figure 1 we can see that most the objects are discarded by the nearest pivot, although a wide percentage can be discarded by both the nearest and furthest pivot too. Only in a few cases the object is discarded by its furthest pivot but not by the nearest.

In order to analyze these data and understand in what cases the object is discarded only by its furthest pivot, we computed the average and standard deviation of the distribution of distances among objects for each collection. Subsequently, for the cases where objects were discarded only by their nearest pivot ("Nearest"), only by their furthest pivot ("Furthest") or both of them ("Both"), we computed the average distance from objects in each group

Collection	μ	σ	Nearest		Furthest		Both	
			$zd_{nearest}$	$zd_{furthest}$	$zd_{nearest}$	$zd_{furthest}$	$zd_{nearest}$	$zd_{furthest}$
UV08	1.5086	0.2452	-4.3989	0.9845	-4.1949	1.3923	-4.3989	1.4331
UV10	1.4032	0.2456	-3.7182	2.2671	-3.5147	2.6743	-3.7182	2.6743
UV12	1.2652	0.2450	-3.0008	3.5298	-2.7151	3.9788	-3.0416	3.9380
UV14	1.1244	0.2469	-2.3669	4.6804	-1.9214	5.0855	-2.4480	5.0450
English	8.3176	2.0260	-2.3335	4.6113	-1.9040	5.1591	-2.2742	4.9617
Nasa	1.2342	0.3424	-2.2611	3.1419	-2.0859	2.9083	-2.1735	2.7623

TABLE II. Standard scores of the distances to nearest and furthest pivots.

to the nearest and furthest pivots. The results are shown in Table II. The distances from the objects to the pivots are shown as standard scores ($z = (d - \mu)/\sigma$) for their better comparison. The nearest and furthest pivots are taken from a set of pivots obtained with SSS and $\alpha = 0.25$.

Lets take English as an example. The average distance between two words is $\mu = 8.31$, and the standard deviation is $\sigma = 2.02$. Take into account that although the distribution of the distances is normal, the standard scores are computed on the distances from the discarded objects and their corresponding pivots. An object is going to be discarded by its nearest pivot when the standard score of its distance to the nearest pivot is about -2.33. However, when the standard score of the distance to the nearest pivot is around or less than -1.9, and the standard score of the distance to the furthest pivot is around 5.1, the object is going to be discarded only by the furthest pivot. In the last two columns we can observe the standard scores to the nearest and furthest pivots for the objects that can be discarded by both of them.

It is curious to observe that, as the complexity of the synthetic collections grows, the furthest object has to be further to have some capacity for discarding an object. In other words, there are few possibilities of the object to be discarded by its furthest pivot. These results are in agreement with the results already shown in Figure 1.

These standard scores give us a threshold for each collection that permits us to decide for each object if the nearest or furthest pivot will be the one with more chances of discarding it.

As a conclusion, the nearest pivot is the one with more capacity of discarding an object, and only in those cases when it is not near enough it is worth to try to use the furthest pivot if it is far enough. The standard score of the distance to the nearest and furthest pivot can give us for each collection a guideline of when to take the furthest pivot. In case of doubt because the distance of both could be over or below these thresholds, the most reliable option is to use the nearest pivot.

V. Minimum-Space Pivot-Based Index

Based on the results of our empirical analysis of pivot effectiveness, we can propose to build an index where for each object we store only the identifier of the pivot that is going to be used, and the distance from the object to that pivot. That pivot will be the nearest pivot, or the furthest in those cases where the distance from the object to the nearest is not small enough and a really far pivot is available. Since this method needs less space for storing the distances from objects to pivots, we can afford having a larger list of pivots that will increase the chances for each object to have a really near pivot.

We could see the resulting index as if we created a Voronoi partition of the space, storing for each object the partition to which it belongs and the distance to the pivot centering of that partition. However, in our case, for the objects that are not near enough to the center of their partition (because they are outliers or they are very close to the limit of two regions), we store information about its furthest pivot instead of about the nearest.

A. Storing Distances with Less Precision

As our goal is to save space, observe that the identifiers of the pivots can be stored with $\log_2(m)$ bits (m = number of pivots), they will require less than a byte in most cases. The distance to the pivot can also be stored with a variable number of bits if we permit to loose some precision.

For the evaluation of our method we will refer to it as Unique Pivot Index (UPI). Table III shows two different configurations of our method. In the first one ("UPI (4 bytes)"), the identifiers of pivots were stored using the minimum number of bits, and the distances were stored using the 4 bytes needed for float precission. In the second ("UPI (2 bytes)"), the identifiers of pivots were also stored using the minimum number of bits, but the distances were stored using 2 bytes, this is, with half precission. For each option we show the space used by the index in MB ("Space") and the number of evaluations of the distance function ("Eval. d"). As we can see in the table, when the distances are stored losing precission, the loss in number of comparisons is almost inexistent, while the space is

reduced to about a half of that needed when using float precission for the distances.

Collection	UPI (4 bytes)		UPI (2 bytes)	
	Space	Eval. d	Space	Eval. d
UV08	0.4311	3094.13	0.2594	3095.72
UV10	0.4348	5891.64	0.2631	5893.84
UV12	0.4456	9739.49	0.2740	9741.91
UV14	0.4673	18160.91	0.2957	18164.03
English	0.3083	12373.45	0.1897	12373.45
Nasa	0.1757	1958.34	0.1068	1958.72

TABLE III. Space and distance evaluations when storing distances with less precision.

The number of bits necessary for storing the distances depends on the range of values returned by the distance function. Therefore, our method can be parameterized to store the distances with a given number of bits depending on the definition of the distance function and the tolerance to the loss of precission.

B. Evaluation

We compared our method with the following methods for evaluating its competitiveness:

- SSS with as much space as needed for its optimal configuration ("SSS (optimal α)").
- KVP storing for each object only the distance to its nearest and furthest pivots ($k = 2$) ("KVP ($k = 2$)").
- List of Clusters, [22] since it is a representative of effective clustering-based methods ("List of Clusters").

Although SSS performs a much smaller number of comparisons, we compared our method with it since it constitutes a good baseline for the number of distance evaluations and space usage.

Table IV shows the results of the comparison. Note that the space used by UPI is the needed to store the table of distances and the needed to store the list of pivots too, that can be larger than in other methods.

Our method obtains better results than KVP in all collections using less space for storing the index. When compared with List of Clusters, our method obtains better results in terms of distance evaluations in all collections of synthetic vectors and in Nasa, using less space than List of Clusters. In the case of English our method needs more evaluations of the distance function, but it needs less space for storing the index. The number of evaluations of the distance function is of course higher than the obtained with SSS, but the reduction in the space needed for the index is very significant. For instance, in the case of UV14, SSS builds an index of about 168 MB, while our method builds an index of about 0.37 MB. These results show that our method is a competitive strategy to previous proposals when using the same amount of space.

VI. Conclusions

In this paper, we consider the problem of reducing as much as possible the space requirements of pivot-based methods for similarity search in metric spaces.

We present an empirical analysis of the effectiveness of the pivots for each object in the database, studying the effect of the selection of the initial set of pivots and the selection of the most promising one for each object. Our results also show that we can store for each object only the distance to its most promising pivot (the nearest or the furthest) reducing the amount of space to that required for clustering-based methods.

Based on the results of this analysis we propose a new pivot-based method that needs an amount of space smaller or very close to that used by clustering-based methods. Our method combines the strategies of scope and range coarsening: we store only the most promising distance for each object, and these distances are stored with half precision. Our experimental evaluation shows that our method is a competitive strategy against previous proposals when compared using the same amount of space.

As future work, we are further studying the parametrization of the number of bits used for storing the distance depending on the effect of the loss of precission in the search cost. We are also working on strategies for increasing the number of pivots without increasing the internal complexity, thus we are working on building an index on the pivots in order to avoid the comparison of the query with all the pivots.

References

[1] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín, "Searching in metric spaces," *ACM Computing Surveys*, vol. 33, no. 3, pp. 273–321, September 2001, aCM Press.

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

[3] E. Chávez, J. L. Marroquín, and G. Navarro, "Fixed queries array: A fast and economical data structure for proximity searching," *Multimedia Tools and Applications*, vol. 14, no. 2, pp. 113–135, 2001, elsevier.

[4] K. Figueroa and K. Fredriksson, "Simple space-time trade-offs for aesa," in *Proc. of Int. Workshop on Experimental Algorithms (WEA'07)*, ser. LNCS(4525). Springer, 2007, pp. 229–241.

[5] C. Celik, "Priority vantage points structures for similarity queries in metric spaces," in *Proc. of EurAsia-ICT 2002: Information and Communication Technology*, ser. LNCS(2510). Springer, 2002.

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

[7] 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*, vol. 15, pp. 9–17, 1994, elsevier.

[8] W. A. Burkhard and R. M. Keller, "Some approaches to best-match file searching," *Communications of the ACM*, vol. 16, no. 4, pp. 230–236, April 1973, aCM Press.

Collection	SSS (optimal α)		KVP (k=2)		UPI		List of Clusters	
	Space	Eval. d	Space	Eval. d	Space	Eval. d	Space	Eval. d
UV08	18.1963	141.43	1.3752	8724.62	0.2594	3095.72	0.3643	6139.21
UV10	60.4255	367.14	1.3789	13063.63	0.2631	5893.84	0.3710	11264.98
UV12	85.8317	645.08	1.3897	18955.94	0.2740	9741.91	0.3710	17273.55
UV14	168.5734	1381.64	1.4115	28502.26	0.2957	18164.03	0.3710	28253.04
English	45.0553	354.89	0.9604	18305.43	0.1897	8872.37	0.2651	7885.79
Nasa	7.4438	168.62	0.5547	2676.93	0.1068	1958.72	0.1427	2027.08

TABLE IV. Comparison with other methods.

[9] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu, "Proximity matching using fixed-queries trees," in *Proc. of the 5th Symposium on Combinatorial Pattern Matching (CPM'94)*, ser. LNCS(807). Springer-Verlag, 1994, pp. 198–212.

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

[11] T. Bozkaya and M. Ozsoyoglu, "Distance-based indexing for high-dimensional metric spaces," in *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD'97)*, A. Press, Ed., 1997, pp. 357–368.

[12] N. R. Brisaboa, A. Fariña, O. Pedreira, and N. Reyes, "Similarity search using sparse pivots for efficient multimedia information retrieval," in *Proc. of the 8th IEEE Int. Symposium on Multimedia (ISM'06)*. IEEE Press, 2006, pp. 881–888.

[13] C. Celik, "Effective use of space for pivot-based metric indexing structures," in *Proc. of Int. Workshop on Similarity Search and Applications (SISAP'08)*. IEEE Press, 2008, pp. 402–409.

[14] B. Bustos, G. Navarro, and E. Chávez, "Pivot selection techniques for proximity searching in metric spaces," *Pattern Recognition Letters*, vol. 24, no. 14, pp. 2357–2366, 2003, elsevier.

[15] S. Brin, "Near neighbor search in large metric spaces," in *Proc. of 21st Conf. on Very Large Databases (VLDB'95)*. ACM Press, 1995, pp. 11–15.

[16] J. Vleugels and R. C. Veltkamp, "Efficient image retrieval through vantage objects," *Pattern Recognition*, vol. 35, no. 1, pp. 69–80, 2002, elsevier.

[17] C. Hennig and L. J. Latecki, "The choice of vantage objects for image retrieval," *Pattern Recognition*, vol. 36(9), pp. 2187–2196, September 2003, elsevier.

[18] R. H. van Leuken, R. C. Veltkamp, and R. Typke, "Selecting vantage objects for similarity indexing," in *Proc. of the 18th International Conference on Pattern Recognition (ICPR'06)*. IEEE Press, 2006, pp. 453–456.

[19] J. Venkateswaran, T. Kahveci, C. M. Jermaine, and D. Lachwani, "Reference-based indexing for metric spaces with costly distance measures," *The VLDB Journal*, vol. 17, no. 5, pp. 1231–1251, 2008, springer.

[20] B. Bustos, O. Pedreira, and N. R. Brisaboa, "A dynamic pivot selection technique for similarity search in metric spaces," in *Proc. of 1st Int. Workshop on Similarity Search and Applications (SISAP'08)*. IEEE Press, 2008, pp. 105–112.

[21] SISAP, "Metric spaces library (`http://sisap.org`)." [Online]. Available: http://sisap.org/Metric_Space_Library.html

[22] E. Chávez and G. Navarro, "A compact space decomposition for effective metric indexing," *Pattern Recognition Letters*, vol. 26, no. 9, pp. 1363–1376, 2005.