Spatial Selection of Sparse Pivots for Similarity Search in Metric Spaces [⋆]

Oscar Pedreira and Nieves R. Brisaboa

Database Laboratory, Facultade de Informatica
University of A Coruña
Campus de Elviña s/n, 15071 A Coruña, Spain
{opedreira,brisaboa}@udc.es

Abstract. Similarity search is a necessary operation for applications dealing with unstructured data sources. In this paper we present a pivot-based method useful, not only to obtain a good pivot selection without specifying in advance the number of pivots, but also to obtain an insight in the complexity of the metric space. Sparse Spatial Selection (SSS) adapts itself to the dimensionality of the metric space, is dynamic, and it is suitable for secondary memory storage. In this paper we provide experimental results that confirm the advantages of the method with several metric spaces. Moreover, we explain how SSS can be easily parallelized. Finally, in this paper we conceptualize Nested Metric Spaces, and we prove that, in some applications areas, objects can be grouped in different clusters with different associated metric spaces, all of them nested into the general metric space that explains the distances among clusters.

Key words: Similarity search, metric spaces, pivot selection

1 Introduction

Similarity search has become a very important operation in applications that deal with unstructured data sources. The computational cost of the algorithms that determine the similarity between two objects makes similarity search an expensive operation. This fact has motivated the development of many research works aiming to do efficient similarity search over large collections of data.

The similarity search problem can be formally defined through the concept of *metric space*. A *metric space* $(\mathbb{X}, d)$ is composed of a universe of valid objects $\mathbb{X}$ and a *distance function* $d : \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}^+$ defined among them. This function holds several properties: strictly positiveness ($d(x, y) > 0$ and if $d(x, y) = 0$ then $x = y$), symmetry ($d(x, y) = d(y, x)$), and the triangle inequality ($d(x, z) \leq d(x, y) + d(y, z)$). The finite subset $\mathbb{U} \subseteq \mathbb{X}$ with size $n = |\mathbb{U}|$, represents the collection of objects where searches are performed. A k-dimensional vector space

[⋆] This work has been partially supported by CYTED VII.J (RITOS2), MCYT (PGE and FEDER) grants TIC2003-06593 and TIN2006-15071-C03-03, and Xunta de Galicia grant PGIDIT05SIN10502PR is a particular case of metric space in which every object is represented by a vector of k real coordinates. The dimensionality of a vector space is clearly k, the number of components of each vector. Although general metric spaces do not have an explicit dimensionality, we can talk about their *intrinsic dimensionality*, following the idea presented in [1] were it is defined as $\mu^2/2\sigma^2$ (being μ and σ^2 the mean and variance of d respectively). The higher the dimensionality the more difficult the search.

The definition of the distance function d depends on the type of the objects we are managing. For example, in the case of a vector space, d could be a distance function of the family L_s, defined as $L_s(x, y) = (\sum_{1 \leq i \leq k} |x_i - y_i|^s)^{\frac{1}{s}}$. For instance, L_1 is known as *Manhattan distance*, L_2 is the *Euclidean distance*, and $L_\infty = max_{1 \leq i \leq k} |x_i - y_i|$ is the *maximum distance.*

There are three main queries of interest in a metric space: *i) range search*, that retrieves all the objects $u \in \mathbb{U}$ within a radius r of the query q, that is: $\{u \in \mathbb{U} \;/\; d(q, u) \leq r\}$; *ii) nearest neighbor search*, that retrieves the most similar object to the query q, that is: $\{u \in \mathbb{U} \;/\; \forall\, v \in \mathbb{U},\; d(q, u) \leq d(q, v)\}$; and *iii) k-nearest neighbors search*, retrieving the set $A \subseteq \mathbb{U}$ such that $|A| = k$ and $\forall\, u \in A,\; v \in \mathbb{U} - A,\; d(q, u) \leq d(q, v)$. The range query is the most used, and the others can be implemented in terms of it [1]. In any case, the distance function is the unique information that can be used in order to perform searches. The naive way of implementing those operations is to compare all the objects in the collection against the query. The problem is that the evaluation of the distance function is very expensive, and therefore searches become inefficient if the collection has a high number of elements. Thus, reducing the number of evaluations of the distance function is the main goal of the methods for similarity search in metric spaces.

The existing techniques differ usually in some features. Some of them allow only discrete (and not continuous) distances. There are also static methods, where the index has to be build on the whole collection, and dynamic techniques where the index is built as elements are added to an initially empty collection. Other important factor is the possibility of storing the index efficiently into secondary storage, and the number of I/O operations needed to access it. In general, the applicability and efficiency of a method depends on this issues.

Search methods can be classified into two types [1]: *clustering*-based and *pivot-based* techniques. *Clustering-based* techniques split the metric space into a set of equivalence regions each of them represented by a *cluster center*. During searches, whole regions can be discarded depending on the distance from their cluster center to the query. But the technique we present here is *pivot-based*, therefore a more detailed explanation about *pivot-based* methods will be provided later.

We have developed *Sparse Spatial Selection* (SSS), a new pivot-based technique. SSS is a dynamic method since the collection can be initially empty and/or grow later. It works with continuous distance functions and is suitable for secondary memory storage. The main contribution of SSS is the use of a new pivot selection strategy. This strategy generates a number of pivots that depends on the intrinsic dimensionality of the space (something interesting from both the theoretical and practical points of view). Moreover, SSS can be easily parallelized as we show in this paper. On the other hand SSS can be extended to deal with more complex metric spaces where the distances among subsets of objects depend on specific dimensions that are not relevant for other set of objects. That is, in some applications areas, objects can be grouped in different clusters with different associated metric spaces, all of them nested into the general metric space that explains the distances among clusters. To deal with these complex spaces we propose the extension of SSS becoming Sparse Spatial Selection for Nested Metric Spaces (SSSNMS).

The rest of the paper is structured as follows: Section 2 describes the pivot-selection problem and its importance for the efficiency of pivot-based methods. Then, *Sparse Spatial Selection* is described in Sec. 3. In Sec. 4 we present and discuss the experimental results we obtained in the tests. Section 5 describes the parallelization of the algorithm and the concept of *nested metric spaces*. Finally, Sec. 6 shows our conclusions and future lines of work.

2 Previous Work on Pivot Selection

Pivot-based search techniques choose a subset of the objects in the collection that are used as *pivots*. An index is build by computing the distances from each pivot to each object in the database. Given a query (q, r), the distances from the query q to each pivot are computed, and then some objects of the collection can be directly discarded using the triangle inequality and the distances precomputed during the index building phase. Being $u \in \mathbb{U}$ an object in the collection, we can discard u if $|d(p_i, u) - d(p_i, q)| > r$ for any pivot p_i, since by the triangle inequality, if this condition is true, its distance to q will be $d(u, q) > r$. The objects that can not be discarded by this condition make up the candidate list, and they must be compared against the query. The *total complexity* of the search is the sum of the *internal complexity*, the comparisons of q with each pivot, and the *external complexity*, the comparisons of q with each object in the candidate list. The most well-known pivot-based methods are: *Burkhard-Keller-Tree* (BKT) [2], *Fixed-Queries Tree* (FQT) [3], *Fixed-Height FQT* (FQHT) [4], *Fixed-Queries Array* (FQA) [5], *Vantage Point Tree* (VPT) [6] and their variants [7, 8], *Approximating and Eliminating Search Algorithm* (AESA) [9] and LAESA (*Linear AESA*) [10].

It is well-known that the efficiency of a similarity search method depends on the set of objects chosen as pivots. The number of pivots, their "location" in the metric space and their "location" with respect to the rest of pivots determine actually the capacity of the index to discard elements without comparing them against the query.

Most of the pivot-based search methods choose pivots randomly. Furthermore, there are no guidelines to determine the optimal number of pivots, since this parameter depends on the metric space we are working with. In previous work, some heuristics for pivot selection have been proposed. For example, in

[10] pivots are objects maximizing the sum of distances among them. [6] and [11] propose heuristics to obtain pivots far away from each others. In [12] the importance of the pivot selection strategy was studied in depth, showing empirically how it affects to the performance of a technique.

The main contribution in [12] is a criterion to compare the efficiency of two sets of pivots of the same size. Using that criterion different techniques for pivot selection were proposed and proved. Their results show that the technique called *good pivots* were consistently better than a *random* selection of pivots or the use of *outliers* (objects far away among them and to the rest of the objects).

But the performance obtained by all the techniques depends on the metric space considered. For example the use of *outliers* works very well in metric spaces were the objects are uniformly distributed but it works very bad in real metric spaces [12].

Determining the optimal number of pivots k is an important problem. It is known that the efficiency of the searches depends on this parameter. Moreover, k can vary greatly for different metric spaces. In [12] a brute-force approach to determine the optimal number of pivots is used, because this value has to be fixed.

3 Sparse Spatial Selection (SSS)

Let $(\mathbb{X}, d)$ be a metric space, $\mathbb{U} \subseteq \mathbb{X}$ an object collection, and M the maximum distance between any pair of objects, $M = max\, \{\ d(x,y)\ /\ x, y \in \mathbb{U}\ \}$. The set of pivots contains initially only the first object of the collection. Then, for each element $x_i \in \mathbb{U}$, x_i is chosen as a new pivot if its distance to every pivot in the current set of pivots is equal or greater than $M\alpha$, being α a constant parameter that takes values around 0.4. That is, an object in the collection becomes a new pivot if it is located at more than a fraction of the maximum distance with respect to all the current pivots. The following pseudocode summarizes the pivot selection process:

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

When a new object is inserted in the database, its distance to all the pivots already selected is computed and stored. If its distance to all of them is equal or greater than $M\alpha$, the object is added to the set of pivots. In this case, its distance to every object in the database is computed and stored in the index structure. Thus, the number of pivots does not have to be stated in advance over an initial object collection; it grows at the same time as the intrinsic dimensionality of the collection does. The building of the index is completely dynamic, and the set of pivots adapts appropriately to the new inserted objects (actually the collection could be initially empty, which is interesting in practical applications). Furthermore, it guarantees that even though the collection grows, the pivots will be well distributed over the metric space.

It seems evident that all the selected pivots will not be too close to each other (more that $M\alpha$). This is a desirable characteristic in a set of pivots [12], but there are more advantages. Forcing the distance between two pivots to be greater or equal than $M\alpha$, we ensure that they are well distributed in the whole space. It is important to take into account that our pivots are not very far away from each others neither very far from the rest of objects in the collection (i.e., they are not *outliers*). Our hypothesis is that, being well distributed in the space, when a search is performed our set of pivots will be able to discard more search objects than pivots selected with a different strategy.

Since our method needs only to store the distances between the pivots and the objects in the collection, we can use storage structures suitable for secondary memory. For example, a simple way to store in secondary memory the distances is the use B-trees. We could create a B-tree for each pivot to store its distances to all the objects in the database. Thus, when a new pivot is added to the structure, a new B-tree is created for it. This implementation is suitable for the dynamic nature of the method.

3.1 The parameter α and the number of pivots

Although in our method it is not necessary to state in advance the number of pivots to use, we have to set the value of α. This value determines the number of pivots. It is clear that the bigger the value of α, the smaller the number of pivots that can be "placed" into the space. However, α must always take values between 0.35 and 0.40, depending on the intrinsic dimensionality of the space. That is, the optimal results in SSS are always obtained when α is set to those values and in general a higher α works better when the intrinsic dimensionality is higher.

Figure 1 shows the number of evaluations of the distance function in terms of α for vector spaces of dimensionality 8, 10, 12, and 14. In this figure we can see that the best result is always obtained for values of α that range from 0.35 and 0.40, and that the efficiency of the method is virtually the same for all the values of α included in this interval. We can also see that when $\alpha > 0.40$ the number of evaluations of the distance function takes higher values in spaces of high dimensionality. This result is due to the fact that an increase in the value of α implies a reduction of the number of pivots, and that this reduction has a stronger effect in spaces of high dimensionality.

This results show some of the main advantages of our proposal. Our method finds itself the appropriate number of pivots for the intrinsic dimensionality of the metric space, using only the maximum distance between any pair of objects in the collection and the parameter α. In subsection 4.2 we present empirical results about how SSS captures the intrinsic dimensionality of any vector or metric space.

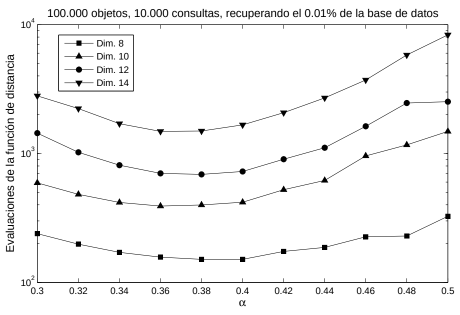

Fig. 1. Number of evaluations of the distance function for several vector spaces, in terms of the value of α.

4 Experimental Results

4.1 Experimental environment

We have tested the algorithm using several collections of data. First we used synthetic sets of random points in vector spaces of dimensionalities $k = 8$, 10, 12 and 14. Each collection has $100,000$ vectors uniformly distributed in an hypercube of side 1. The Euclidean distance was the distance function used with this data sets. Using this collections of data we can study the behavior of the algorithm in spaces of different intrinsic dimensionality. We have also tested the algorithm with real metric spaces. The first one is a collection of $69,069$ words taken from the English dictionary, using the edition distance as the distance function. The second is a collection of $47,000$ images extracted from the NASA photo and video archives, each of them transformed into a 20-dimensional vector, using the Euclidean distance to measure the similarity between them.

4.2 Number of pivots and the intrinsic dimensionality

The concept of intrinsic dimensionality was defined in [1] as $\mu^2/2\sigma^2$ where μ is the average of distances among all the objects in the database and σ^2 is its variance. To show that our method captures the intrinsic dimensionality of the space by itself, we explicitly calculated the intrinsic dimensionality of some experimental vector and metric spaces using this formula, that is we computed $\mu^2/2\sigma^2$. The results are shown in table 1. Each row shows average, variance, intrinsic dimensionality and number of pivots with different α values for each one of the vector or metric spaces used in the experiments.

It is clear the the number of pivots grows as the intrinsic dimensionality grows therefore SSS can be considered as an alternative method to obtain and insight into the dimensionality of a metric space.

On the other hand in Section 3 we emphasized that our method dynamically generates a number of pivots that depends on the dimensionality of the space,

DB	μ	σ^2	$Int.Dimens.$	α	$pivots$	α	$pivots$
English	8.365435	3.8853	135.9486	0.5	108	0.44	205
Spanish	8.281279	3.4975	119.9314	0.5	64	0.44	124
k=8	1.0484	0.0618	0.0339	0.5	18	0.38	68
k=10	1.0493	0.1002	0.0551	0.5	25	0.38	126
k= 12	1.0552	0.1598	0.0889	0.5	43	0.38	258

Table 1. Intrinsic dimensionality and Number of pivots

and not on the number of elements in the database. Table 2 shows the number of pivots selected in several test collections of different size. The number of objects selected as pivots increases as the dimensionality of the vector space does. This result also shows that the number of pivots depends on the intrinsic dimensionality of the metric space. In all the test spaces the number of pivots grows quickly with the first objects of the database. Then that number grows much more slowly until it becomes stable. Obviously, when the collection has few elements, the number of pivots depends on its size. However, when the collection reaches a given size no more pivots will be selected even if new objects are inserted in the database. This happens because the current set of pivots covers all the space and captures its dimensionality. With this results we can conclude that the number of pivots generated depends on the intrinsic dimensionality of the space, and not on the size of the collection.

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

Table 2. Number of pivots selected in vector spaces of dimensionality k = 8, 10, 12, and 14, against the size of the collection

4.3 Search efficiency

In this section we show the results obtained in the tests performed to evaluate the efficiency of the algorithm in the search operation. The first set of tests used the four vector spaces. For each of them, we got the mean number of evaluations of the distance function over $10,000$ queries. The mean number of elements retrieved in each of them is the 0.01% of the database. In order to evaluate the behavior of the algorithm, we compared the results with those obtained with the pivot selection techniques proposed in [12].

Table 3 shows the minimum number of evaluations of d we have obtained with each pivot selection strategy, and the number of pivots used. We can observe that the number of evaluations of the distance function obtained with our method is always around the best result obtained with the strategies proposed in [12]. This results show that SSS has an efficiency similar to that of other more complex techniques. In the results of our tests we can also see that the number of pivots that our method generates is very similar to the optimum number of pivots of other pivot selection techniques.

Method	$k = 8$		$k = 10$		$k = 12$		$k = 14$	
	pivots	eval. d	pivots	eval. d	pivots	eval. d	pivots	eval. d
Random	85	213	190	468	460	998	1000	2077
Selection	85	204	200	446	360	986	800	2038
Incremental	65	157	150	335	300	714	600	1458
Loc. Opt. A	70	155	150	333	300	708	600	1448
Loc. Opt. B	60	157	150	369	300	881	760	1930
SSS	57	151	148	389	258	689	598	1452

Table 3. Minimum number of evaluations of d with different pivot selection strategies in vector spaces

Table 4 shows another interesting result obtained in this tests: the average and the standard deviation of the number of evaluations of d with SSS and a random pivot selection. In this table we can see that, in addition to perform less evaluations on average, SSS has also a lower standard deviation, something very important for practical purposes.

Method	$k = 8$		$k = 10$		$k = 12$		$k = 14$	
	μ	σ	μ	σ	μ	σ	μ	σ
Random	224	53	581	166	1046	316	2087	622
SSS	151	33	390	101	689	193	1452	399

Table 4. Average and standard deviation of the number of evaluations of d

Table 5 shows the minimum number of evaluations of d we have reached with our pivot selection technique and the ones proposed in [12], for the collection of words taken from the English dictionary. We have used the edit distance as the distance function. We used a 10% of the database as queries and a query range $r = 2$, that retrieves around the 0.02% of the collection. In this case, the result obtained with our technique is better than the obtained with any other one. As happened with vector spaces, the number of pivots that our method generates is similar to the optimum number of pivots used by other strategies, that have got this number by trial and error.

Method	pivots	eval. d
Random	200	443
Good pivots	200	389
Outliers	200	429
SSS	205	370

Table 5. Minimum number of evaluations of d in a collection of words

Finally, Fig. 2 shows the results of the experiments with the collection of images from NASA archives. As in the previous experiments, the 10% of the objects were used as queries. In this case, the query range retrieves on average the 0.10% of the database. In the Fig. 2 we can see that SSS performs always better than a random pivot selection. However, in this case our strategy has not a better behavior than *Incremental* [12], that performs 220 evaluations with 60 pivots, whereas our better result is 255 evaluations with 77 pivots.

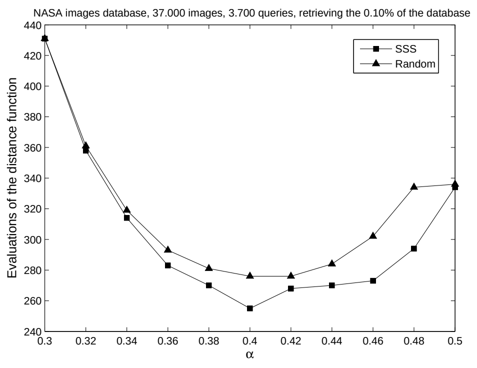

Fig. 2. Number of evaluations of d for the collection of NASA images

5 Extensions of SSS

The ideas and results presented in previous sections show that SSS has many desirable characteristics that make it a good choice from a practical point of view. SSS has an efficiency similar (or even better) to that of other existing techniques. It is a dynamic method able to adapt itself to the growth of the collection in an efficient way. The index construction process is simpler than in other methods and the index structure can be easily stored in secondary memory. In this section we first describe how the method can be parallelized. Then we present the concept of *Nested Metric Spaces* and how the performance of SSS can get reduced in such collections.

5.1 Parallel processing

The structure of the index constructed with SSS makes it very easy to execute the search operation in a parallel architecture. As mentioned in previous sections, the index stores the distances from every object in the collection to each pivot, so we can see the index structure as a table with a column for each pivot and a row for each object in the collection. Suppose that the collection has n objects and that we can use p nodes (processors) to execute the search operation. In this situation we can store $\frac{n}{p}$ rows in each node. During the search operation each node will have to process only the rows stored in it, making the search much more efficient. Once the search is completed in each node, a union operation has to be executed to make up the final candidate list with the candidates from each node. Other option is each node to process its own candidate list and then execute the union operation to make up the final result set. The parallel search is easy to implement and can be very important for application domains in which the search operation has a very high computational cost due to the complexity of the distance function.

5.2 Nested metric spaces

In all the experiments mentioned in Sec. 4, SSS always performed better than a random pivot selection. Of course this is the result we hoped. However, during the test phase of our research we found a test collection where *Random* is always more efficient than SSS and most of the other pivot-based techniques. The collection has $100,000$ vectors of 112 coordinates each of them corresponding to a color image. The search for an explanation of this (very) strange result led us to the concept of *Nested Metric Spaces*.

Our hypothesis is that, in some metric spaces, the objects in the collection can be grouped in different clusters or subspaces. Different dimensions explain the difference between each pair of objects in each of this subspaces nested into a more general one. Figure 3 illustrates this idea. The general metric space of this picture has a main dimension along the horizontal axis and another two dimensions in other directions. In the figure we can see that there are big subspaces in which two objects are equal to each other according to the main dimension X but different according to the own dimensions (Y and Z respectively). And this is the reason because a random set of pivots performs better than SSS in a space like this. The maximum distance M is given by the main dimension X, so if previous pivots p_1, p_2, p_3 were already selected, no more pivots can be placed in any of the X subspaces. However, a random set of pivots has good opportunities to place some pivots in the subspaces since they have a big number of elements.

Thus, two objects in a subspace will be far away from each other according to a random set of pivots, but very close according to the pivots p_1, p_2, p_3 selected by SSS.

Instead of seeing this result as a bad characteristic of SSS, it has broadened our research line in the search of refinements of SSS able to deal with this complex cases. Our idea is Sparse Spatial Selection for Nested Metric Spaces (SSSNMS), a new approach that tries to solve this problem. The goal of this method is to identify the subspaces and apply SSS in each of them. In a first phase, SSS is applied to the whole collection with a high value of α. Thus we have few pivots and the distances from each object in the database to each of them. The idea is to use this information to identify the subspaces searching for objects close to each other according to the pivots but really far away according to the distance function. In a second phase, SSS is applied in each subspace. Thus, each subspaces as its own set of pivots, able to differentiate the objects placed in it. Of course we have now more pivots than applying only SSS. However, during the search operation, the pivots of each subspace are used only to do the search in that subspace if necessary.

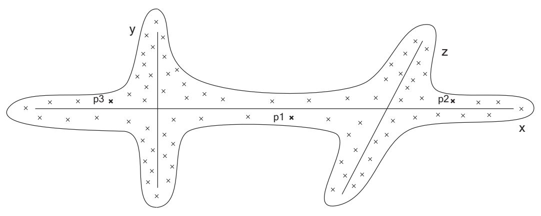

Fig. 3. Subspaces nested into a general metric space

6 Conclusions

In this paper we presented SSS and some variations of this method. The main contribution of our method is the pivot selection strategy. The goal pursued by this pivot selection technique is to generate a set of pivots well distributed over the whole space. Furthermore, the index is completely dynamic (supporting insertions in the collection) and the index structure can be efficiently stored in secondary memory.

Our experimental results show that the method generates a number of pivots that depends on the intrinsic dimensionality of the metric space, and not on the number of elements of the collection. This fact also makes our method a useful way to obtain an insight of the complexity of the space. This number of pivots is very similar to the optimum number for other strategies. This makes it unnecessary to state in advance the number of pivots needed for the index structure, something that no method has considered until now. The number of pivots selected is adapted to the space complexity, avoiding to select unnecessary pivots that could reduce the search efficiency. The efficiency of our method in vector and metric spaces is at least as good as the best obtained in previous works [12].

We have also described the way to parallelize the algorithm, something important in some application domains. Finally we presented the concept of *Nested Metric Spaces*, its importance in the performance of the search operation and a way to deal with it.

Our work line still maintains opened some questions that will be addressed in the future. First, we are evaluating the behavior of SSS with other real metric spaces, like collections of text documents or music. We are currently working in the parallel implementation of the method. We are also working in refinements of the pivot selection strategy able to deal with the nested metric spaces problem.

References

1. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Computing Surveys **33**(3) (2001) 273–321
2. Burkhard, W.A., Keller, R.M.: Some approaches to best-match file searching. Communications of the ACM **16**(4) (1973) 230–236
3. Baeza-Yates, R., Cunto, W., Manber, U., Wu, S.: Proximity matching using fixed-queries trees. In: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, Springer-Verlag (1994) 198–212
4. Baeza-Yates, R.: Searching: an algorithmic tour. Encyclopedia of Computer Science and Technology **37** (1997) 331–359
5. Chávez, E., Marroquín, J.L., Navarro, G.: Overcoming the curse of dimensionality. In: European Workshop on Content-based Multimedia Indexing (CBMI'99). (1999) 57–64
6. Yianilos, P.: Data structures and algorithms for nearest-neighbor search in general metric spaces. In: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete Algorithms, ACM Press (1993) 311–321
7. Bozkaya, T., Ozsoyoglu, M.: Distance-based indexing for high-dimensional metric spaces. In: Proceedings of the ACM International Conference on Management of Data (SIGMOD 1997), ACM Press (1997) 357–368
8. Yianilos, P.: Excluded middle vantage point forests for nearest neighbor search. In: Proceedings of the 6th DIMACS Implementation Challenge: Near neighbor searches (ALENEX 1999), Baltimore, Maryland, USA (1999)
9. Vidal, E.: An algorithm for finding nearest neighbors in (aproximately) constant average time. Pattern Recognition Letters **4** (1986) 145–157
10. Micó, L., Oncina, J., Vidal, R.E.: A new version of the nearest-neighbor approximating and eliminating search (aesa) with linear pre-processing time and memory requirements. Pattern Recognition Letters **15** (1994) 9–17
11. Brin, S.: Near neighbor search in large metric spaces. In: 21st conference on Very Large Databases. (1995)
12. Bustos, B., Navarro, G., Chávez, E.: Pivot selection techniques for proximity search in metric spaces. In: SCCC 2001, Proceedings of the XXI Conference of the Chilean Computer Science Society, IEEE Computer Science Press (2001) 33–40