Combining Methods for Searches in Nested Metric Spaces

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

[1] Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Universidad Nacional de Rosario
Rosario, Argentina
hugogercek@gmail.com, {deco, bender}@fceia.unr.edu.ar, salvettimariano@hotmail.com
[2] Departamento de Informática. Universidad Nacional de San Luis
San Luis, Argentina
nreyes@unsl.edu.ar

Abstract. Most search methods in metric spaces assume that the topology of the object collection is reasonably regular. However, there exist nested metric spaces, where objects in the collection can be grouped into clusters or subspaces, in such a way that different dimensions or variables explain the differences between objects inside each subspace. This paper proposes a two levels index to solve search problems in spaces with this topology. The idea is to have a first level with a list of clusters, which are identified and sorted using Sparse Spatial Selection (SSS) and Lists of Clusters techniques, and a second level having an index for each dense cluster, based on pivot selection, using SSS. It is also proposed for future work to adjust the second level indexes through dynamic pivots selection to adapt the pivots according to the searches performed in the database.

Keywords: metric spaces, pivots selection, similarity search

1 Introduction

With the evolution of information technology and communications have emerged repositories of unstructured information, with types of data such as free text, images, audio and video. This scenario requires more general models, such as metric databases, and tools for efficient searches on these data types. In unstructured data repositories it is more useful a similarity search than an exact search. The similarity search problem can be formalized through the concept of metric space: given a set of objects and a distance function between them, which measures how different they are, the objective is to retrieve those objects that are similar to a given one. In order to improve objects retrieval an index can be used, because an index structure allows fast access to objects. Most of the search techniques were developed assuming that the topology of the object collection is reasonably regular, but experiments on spaces where collections of objects can be grouped into subspaces or clusters have shown that they are not so efficient. In [1] a two level structure is proposed: Sparse Spatial

Selection for Nested Metric Spaces (SSSNMS), which is the first that consider this type of spaces.

This paper presents a new version of this structure for indexing and similarity searching with two levels of indexes. At the first level, clusters are identified with Sparse Spatial Selection (SSS) and are sorted into a List of Clusters (LC) [2]. At the second level, based on a measure of density, the clusters that are considered highly populated with pivots are indexed also using SSS.

The rest of the paper is organized as follows: Section 2 presents basic concepts. Section 3 discusses related work. Section 4 presents the proposed method. Finally, conclusions are presented.

2 Basic Concepts

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

In general metric spaces it can be translated the concept of "dimensionality", even if the objects are not assumed to have coordinates [3]. One easy characterization of the intrinsic dimensionality is obtained from the histogram of distances. An *easy* instance will have a small mean distance value and large standard deviation, while a *difficult* instance will be the converse, a large mean distance value and small standard deviation.

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

building phase. Some pivot-based methods are: *Burkhard-Keller-Tree* [4], *Fixed-Queries Tree* [5], *Fixed-Height FQT* [5], *Fixed-Queries Array* [6], *Vantage Point Tree* [7], *Approximating and Eliminating Search Algorithm* [8], *Linear AESA* [9] y *SSS* [1].

3 Related Work

Pivots selection affects the efficiency of the search method in the metric space, and the location of each pivot with respect to the others determines the ability to exclude elements of the index without directly comparing them with the query. Most search pivots-based methods select pivots randomly. Also, there are no guidelines to determine the optimal number of pivots, parameter which depends on the specific collection. Several heuristics have been proposed for the selection of pivots. In [9] pivots are objects that maximize the sum of distances among them. In [10] a criterion for comparing the efficiency of two sets of pivots of the same size is presented. Several selection strategies based on an efficiency criterion to determine whether a given set of pivots is more efficient than another are also presented. The conclusion is that good pivots are objects far away among them and to the rest of the objects, although this does not ensure that they are always good pivots.

In [1] the Sparse Spatial Selection (SSS), which dynamically selects a set of pivots well distributed throughout the metric space, is presented. It is based on the idea that, if pivots are dispersed in the space, they will be able to discard more objects during the search. To achieve this, when an object is inserted into the database, it is selected as a new pivot if it is far enough from the other pivots. A pivot is considered to be far enough from another pivot if it is at a distance greater than or equal to $M^*\alpha$. M is the maximum distance between any two objects. α is a constant parameter that influences the number of selected pivots and its takes optimal experimental values around 0.4.

In all of the analyzed techniques for selecting pivots, the number of pivots must be fixed in advance. In [10] experimental results show that the optimal number of pivots depends on the metric space, and this number has great importance in the method efficiency. Because of this, SSS is important in order to adjust the number of pivots as well as possible. In [11] an improved SSS is presented, where the index suits to searches, after the index was adapted to the metric space, using a dynamic selection of pivots. The initial index is built using SSS and it is "updated" during searches.

Another improvement to SSS is the SSS-Tree [12] that uses trees and the best properties of clustering techniques. Its main feature is that cluster centers are selected using SSS, so the number of clusters in each node depends on the complexity of the subspace associated with it.

Since the indexes lose their efficiency as the intrinsic dimension of data increases, in [2] an index called List of Clusters (LC), based on the compact partition of the data set, is presented. It is shown that the LC is very resistant to the intrinsic dimensionality of the data set. In addition, due how the List of Clusters is built, a special order to its members is given: clusters in previous positions have priority over subsequent clusters, when they contain elements that are located in regions of intersection. Each cluster in the list, which is a subspace of center c and radius r_c, is

called ball. In the LC, the first center chosen has precedence over the later in the case of overlapping balls. That is, all elements that fall under the ball of the first center are stored in that cluster even though they might be in others. Given a query (q, r) the idea is to use this feature to inspect the LC for those clusters in which the ball has query intersection, and stop the search when the query ball is completely contained within this cluster.

In [13] is presented the Sparse Spatial Selection for Nested Metric Spaces (SSSNMS) as a new approach to solve problems of indexing and searching in nested metric spaces. In this type of spaces, objects in the collection can be grouped into different clusters or subspaces. Each of these subspaces is nested within a more general one. The aim of this method is to identify subspaces and apply SSS in each of them. For this, the index constructed by SSSNMS is structured in two levels: first level selects a set of reference with SSS and it is used as centers of clusters to create a Voronoi partition. In the second level, those clusters, that are considered dense, are indexed using SSS pivots in each of them. Given a query (q, r), it is compared against all cluster centers of the first level. Those clusters $C_i=(c_i,r_c)$ for which $d(q,c_i)-r_c>r$ are directly discarded from the result set as the intersection of each cluster with the result set is empty. If the not discarded cluster does not have an associated table of distances from their objects to the pivots, the query is directly compared against all objects in the cluster. If the not discarded cluster has an associated table of distances, the query is compared against pivots and this table is processed in order to eliminate as many objects as possible. Objects that cannot be discarded are directly compared against the query.

In this paper, we analyze the problem of searching in nested metric spaces, and we propose a new index structure that has as main objective to minimize the search time. For this, we use SSS and Lists of Clusters. The proposal is presented in the next section.

4 Proposed Method

Most index structures and their search methods were built to work on collections of data where the spatial distribution is fairly regular. For example, SSS belongs to the family of indexes that get good performance in regular spaces, but its performance is not the best in irregular collections. Moreover, the SSSNMS proposal yields better results in nested metric spaces.

In this paper, we analyze the problem of search in such spaces, and we propose a new index structure that has as main objective to minimize search time. For this, we propose to use SSS to identify clusters nested in the general metric space, obtaining the centers of the clusters to ensure a good coverage of general space. Each cluster remains ordered in a List of Clusters. By using this order during a search, if the query ball is totally contained within a cluster, we can omit inspecting the following clusters. This structure provides high resistance to the intrinsic dimensionality of data. Subspaces considered highly populated are indexed using pivots, based on a measure of density that is presented later, in order to get a good coverage of each subspace.

Therefore, our structure has two levels: a List of Clusters that identifies and maintains an order of each nested subspace in the general metric space, and a pivot index built using SSS for each subspace that we consider dense. This structure is dynamic and adaptive at the same time. Dynamic because it can start with an empty collection to which objects will be added. It is adaptive because it allows to adapt itself to the complexity of space. This is, a priori we do not assume anything about the number of clusters needed, and their characteristics, nor on the number of pivots for each dense subspace.

4.1 Construction of the Index

The efficiency of similarity search methods depends on the set chosen as a reference, where *reference* means a pivot, for pivot-based index, or a cluster center, for clustering-based index.

The structure proposed in this paper has two levels. The first level uses SSS to identify subspaces, and to build a List of Clusters where the centers are well distributed (because of the use of SSS). Also, the order of the clusters will optimize the search (because of the use of LC). The second level uses SSS to obtain pivots in each cluster, acquiring an index where the references are well distributed.

Let (X,d) be a metric space, where $U \subseteq X$ is the database. Let M be the maximum distance between objects ($M=max\{d(x,y)/x,y \in U\}$).

The index is built as follows:

Level One: List of Clusters with SSS. In this first level, nested subspaces in the general metric space are identified and indexed. SSS is used to obtain well distributed centers of the clusters. Each cluster is maintained in a List of Cluster to obtain and preserve an order.

Given a center $c \in U$ and a radius r_c, we define the ball (c,r_c) as the subset of elements of X which are at a maximum distance r_c from center c; and where $r_c < M*\alpha$. Experimentally, in [1] it is shown that the optimal value for α must be in the range [0.35, 0.4] as the dimensionality of the collection.

We define: $I_{U,c,rc}=\{u \in U,\ 0<d(c,u) \leq r_c\}$ as the bucket of internal elements that remain inside the ball of center c; and $E_{U,c,rc}=\{u \in U,\ d(c,u)>r_c\}$ as the other elements (external).

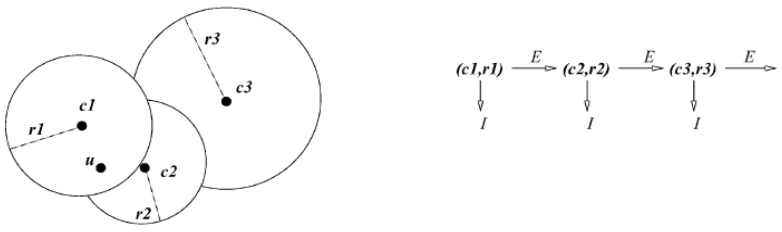

Fig. 1. Clusters representation: $< (c_1, r_1, I_1), (c_2, r_2, I_2), (c_3, r_3, I_3) >$ [14].

The main idea, after selecting the first center, is to go on by selecting the SSS centers iteratively on each set E and get a list of triples (c_i,r_i,I_i) (center, radius,

bucket), where each element represents a cluster . The data structure obtained seems to be symmetric, but it is not. The first center chosen has precedence over the later in the case of overlapping balls, as shown in Figure 1. All items that remain inside the ball of the first center (c_1 in the figure) are stored in the bucket I_1 although that might be within the buckets of subsequent centers (c_2 and c_3 in the figure). The figure shows how the data structure can be viewed as a list, where clusters in previous positions have a preference when it comes to contain elements that are located in regions of intersection on the following clusters.

This structure is dynamic. This allows us to start with an empty collection of elements. But if the initial collection is not empty, an algorithm of "bulk loading" to identify clusters can be applied. This is, to apply a variant of the SSS on the initial set U to obtain only a group of representative elements and the distance between them. For each representative element, distances between it and the others representative elements are averaged, and then they are ranked according to this distance from highest to lowest, and finally their appearances are removed from U. This sorted list is added at the beginning. This ensures that the first items examined by the algorithm will be distant, and therefore should belong to different clusters and so would get a better representation of nested subspaces in the general metric space.

The pseudo code of the algorithm of construction of this index level is as follows:

```
Build_Index(U,L,B)
   for each u_i ∈ U do
      if canBeCenter(u_i,L)   //If distance between u_i and each center is ≥ M*α
         setRadio(r_i,M,α)     //Computes radius r_i which depends on M and α.
         insertAtEndOfL((u_i,r_i,{}),L)   //Inserts triplet ((u_i,r_i,{})
                                          at the end of the List of Clusters
         updateM(M)           //Updates value of M.
      else if isInSomeBallCj(u_i,L,(c_j,r_j,I_j))  //If the element u_i
                            belongs to any ball (c_j, r_j) of L, returns the first
                            triplet (c_j,r_j,I_j) of L that satisfies this condition.
         updateI((c_j,r_j,I_j),u_i))  //Adds element u_i to I_j.
         updateL((c_j,r_j,I_j),L))  //Updates the ball with center c_j of L
         updateM(M)           // Updates value of M.
      else
         updateR(u_i,R))     //Adds u_i to the list L of elements to reconsider.
   reconsider(R,L,B)         //Reconsiders the no indexed elements.
```

This algorithm receives as parameters the set of elements U (preprocessed or not) to index, the List of Clusters L empty, and an empty set B of elements that do not belong to any subspace, but will be indexed with SSS.

If the input is not preprocessed, and the loop *for* is considered as successive insertions for each $u_i \in U$, we would be under the assumption that it starts with an empty database that grows as elements are inserted into it. In the last line of pseudo code, the list R has two types of elements: those who should belong to some subspace of the List of Cluster L but because of the order in which the centers were chosen they do not fall into any ball (c_j,r_j); or elements that are outside from any ball of the List of Clusters L. The method *reconsider(R,L,B)* takes into account these two options: those

elements that should belong to some subspace of L but that could not be observed at first are added to the respective cluster (i.e. the first from the list L if this element "falls" in more than one); and those elements that do not belong to any subspace of L are stored in a bag B of elements. Each element of B is indexed in the usual way with SSS, using each center c of L as a pivot.

r_c is the radius of the cluster of center c. Each radio is static, i.e. once chosen it cannot change, because if so, to update this value the index should be rebuild to keep the properties of the List of Clusters. According to the current values of M, it must be $r_c<M*\alpha$, so centers selection strategies with SSS does not collide with LC properties. That is, a new cluster center is not contained in an existing cluster. Therefore, the radius r_c must be equal to $M*\alpha*\rho$, where $\rho<1$.

Level Two: Choosing pivots on dense subspaces with SSS. When construction is completed the first level of the index, we have: a list of clusters L with elements (c,r,I); and a bag B of elements not contained in any subspace indexed with SSS using the centers c of L as pivots.

The *density* of each cluster is computed as the number of elements of the cluster divided by the maximum distance between them. Those clusters of L, considered dense based on a measure that we will give below, are indexed using SSS, obtaining a reference set consisting of pivots. To compute the density of each cluster can be very costly if the maximum distance between each object is obtained by comparing all the elements of the cluster with the rest. To minimize this cost in construction time we get an approximation of the maximum distance. To do this, an object of the cluster is chosen at random and is compared against all other objects in the cluster. Its further object is compared against all other objects in the cluster to obtain it further object too. After repeating this process a few iterations, we get an approximation of the maximum distance (if it is not the current maximum distance).

We consider that the cluster C_i has high density if $density(Ci)>\mu+2\sigma$, where μ and σ are the mean and standard deviation of the density of all clusters. For each dense cluster, a set of objects is obtained with SSS to be used as pivots, and its table of distances for each object of the cluster to each pivot is computed and stored.

In this second step, the index stores information about the dense subspaces. An element u is chosen as pivot of the subspace with center c_i if the distance of u to each pivot of the subspace is larger than $M_i*\beta$, where M_i is the maximum distance between each pair of objects in the cluster of center c_i and β is a constant value near 0.4 as it is shown in [1].

4.2 Searches

Given a query *(q, r)*, q is compared against all the centers of clusters, following the order in the list of clusters L, until the end of the list or can be stopped if the ball is completely contained in one of the clusters. Each cluster that has not been discarded (i.e., clusters with which it has intersection) is a candidate cluster and should be reviewed. If it reaches the end of the list without the query ball has been completely contained in a cluster, distances to all centers of clusters have been calculated, and therefore they are used to discard some elements of the bag B by filtering the

distances to the pivots (centers of the list of clusters). The range search algorithm is presented in the following pseudo code:

```
SearchL(L,(q,r),B, K)
if L is empty
   if  B is empty
      return K
   else
      return K ∪ SearchB((q,r),B)
let L = <(c, rc, I):L'>
distQC  ← d(q,c)
if distQC ≤ r                    //Query ball contains center c
   if distQC + r ≤ rc            //Query ball is inside the cluster
      return Pivotsearch(I,(q,r)) ∪ K ∪{c}
   else // Query ball contains the cluster or Query ball intersects the cluster
      K' ← Pivotsearch(I,(q,r)) ∪ K ∪ {c}
      return SearchL(L',(q,r),B,K')
else                             //Query ball does not contain the center c
   if distQC + r ≤ rc            //Query ball is inside the cluster
      return Pivotsearch(I,(q,r)) ∪ K
   else if distQC > r + rc       //Query ball is outside the cluster
      return SearchL(L',(q,r),B,K)
   else                          //Query ball intersects the cluster
      K' ← Pivotsearch(I,(q,r)) ∪ K
      return SearchL(L',(q,r),B,K')
```

The list is iterated and the relationship between each cluster and the query is established based on the distance between the query and the center of the cluster. The recursive function *SearchL* has four parameters: the list of clusters L, the query (q, r), the bag of elements B and the list K of candidates (which must be empty to start).

The function *Pivotsearch* gets the list of candidates for each cluster, using the pivots themselves if the cluster is dense and is indexed, and returns all elements of the cluster if it is not dense. Its parameters are: the bucket I of elements of the cluster, which in our case we can think as a reference to index, and the query (q, r). Given the asymmetry of the data set, the search can be pruned if the query ball is totally contained in the ball of center c. In this case, we do not consider the rest of the list. If the end of the list is reached without the query ball has been completely contained in a cluster and the bag of elements B is not empty, the method *SearchB* is responsible for discarding some elements of the bag B by filtering the distances to the pivots (the centers of the list of clusters).

This is an essential feature absent in other algorithms, where the search needs to go into all the partitions that are intercepted by the query ball. In this structure the consideration of relevant partitions can be stopped when the query ball is fully contained on a partition.

The function *Pivotsearch* applies the triangle inequality as follows: given an element e of the index, it can be discarded if $|d(p_i,e)-d(p_i,q)|>r$ for some pivot p_i of

the subspace, since by the triangle inequality if this condition is true, occurs that $d(e,q)>r$.

Finally, once the list of candidates is obtained, the query is compared exhaustively against it, and the distance from centers should not be recalculated since it was previously obtained.

5 Conclusions

This paper presents a new index and similarity search method, which tries to fully exploit the advantages already known from other structures in order to obtain an efficient method for nested metric spaces. We propose a two level structure. A first level where clusters are detected and they kept sorted combining SSS and LC strategies. This allows getting a good coverage of the general metric space and a high resistance to the intrinsic dimensionality of the data set. In the second level each dense subspace is indexed with SSS getting a good coverage of the subspace. In searches, this structure is used first to exclude subspaces, and then to get the list of candidates for each subspace that has not been discarded. With the proposed algorithm, the search can be also stopped when a query ball is completely contained within a cluster from the list, which saves a significant amount of time. It is proposed as future work to use techniques of incoming pivot and outgoing pivot defined in [11], after making a certain number of searches on each subspace, in order to adapt pivots to the searches and to get better performance in future queries. The inclusion of these techniques will enable us to obtain experimental results.

Referencias

1. Pedreira O., Brisaboa N.R.: Spatial Selection of Sparse Pivots for similarity search in metric Spaces. In: 33nd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'07), LNCS vol: 4362, pp. 434-445. Springer (2007).
2. Mamede M.: Recursive Lists of Clusters: A Dynamic Data Structure for Range Queries in Metric Spaces. CITI / Departamento de Informática Faculdade de Ciencias e Tecnologia da UNL. Caparica, Portugal. In Proceedings of ISCIS. 2005, 843-853.
3. Chávez E., Navarro G., Baeza-Yates R., Marroquín J. L.: Searching in Metric Spaces. ACM Computing Surveys. 33(3), pp 273--321. (2001).
4. Burkhard W. A., Keller R. M.: Some approaches to best-match file searching. Communications of the ACM, 16(4): 230-236. (1973).
5. Baeza-Yates R. A., Cunto W., Manber U., Wu S.: Proximity matching using fixed-queries trees. In M. Crochemore and D. Gusfield, editors, Proc. of the 5th Annual Symposium on Combinatorial Pattern Matching, LNCS 807, pages 198-212. (1994).
6. Chavez E., Navarro G., Marroquín A.: Fixed queries array: a fast and economical data structure for proximity searching. Multimedia Tools and Applications (MTAP), 14(2):113-135. (2001).
7. Yianilos P.: Excluded middle vantage point forests for nearest neighbor search. In: 6th DIMACS Implementation Challenge: Near Neighbour searches ALENEX'99. (1999).

8. Vidal E. An algorithm for finding nearest neighbor in (approximately) constant average time. Pattern Recognition Letters 4, 145-157. (1986).
9. Micó L., Oncina J., Vidal R. E.: A new version of the nearest neighbor approximating and eliminating search (AESA) with linear pre-processing time and memory requirements. In: Pattern Recognition Letters, 15:9-17. (1994).
10. Bustos B., Navarro G., Chávez E.: Pivot selection techniques for proximity search in metric spaces. In: XXI Conference of the Chilean Computer Science Society, pp. 33-44. IEEE Computer Science Press. (2001).
11. Salvetti M., Deco C., Reyes N., Bender C.: Adaptive and Dynamic Pivot Selection for Similarity Search. Journal of Information and Data Management, Ed. Sociedade Brasileira de Computação. Vol. 2, No. 1, February 2011, pp. 27–35.
12. Uribe Paredes R., Solar R., Brisaboa N. R., Pedreira O., Seco D.: SSSTree: Búsqueda por Similitud Basada en Clustering con Centros Espacialmente Dispersos. Encuentro Chileno de Computación. Iquique, Chile, Nov. 2007.
13. Brisaboa N. R., Luaces M. R, Pedreira O., Places Á. S., Seco D.: Indexing Dense Nested Metric Spaces for Efficient Similarity Search. In: Proceedings of the 7th International Andrei Ershov Memorial Conference on Perspectives of System Informatics (PSI 2009) - LNCS 5947, Springer, Novosibirsk (Rusia), 2010, pp. 98-109.
14. Chávez E., Navarro G.: A compact space decomposition for effective metric indexing. Pattern Recognition Letters, 26(9):1363–1376, 2005