Solving All-k-Nearest Neighbor Problem without an Index

Edgar Chávez, Verónica Ludueña, and Nora Reyes

Departamento de Informática, Universidad Nacional de San Luis,
San Luis, Argentina
{vlud,nreyes}@unsl.edu.ar
Centro de Investigación Científica y de Educación Superior de Ensenada, México
elchavez@cicese.mx

Abstract. Among the similarity queries in metric spaces, there are one that obtains the k-nearest neighbors of all the elements in the database (*All-k-*NN). One way to solve it is the naïve one: comparing each object in the database with all the other ones and returning the k elements nearest to it (k-NN). Another way to do this is by preprocessing the database to build an index, and then searching on this index for the k-NN of each element of the dataset. Answering to the *All-k-*NN problem allows to build the k-Nearest Neighbor graph (kNNG). Given an object collection of a metric space, the Nearest Neighbor Graph (NNG) associates each node with its closest neighbor under the given metric. If we link each object to their k nearest neighbors, we obtain the k Nearest Neighbor Graph (kNNG).The kNNG can be considered an index for a database, which is quite efficient and can allow improvements.
In this work, we propose a new technique to solve the *All-k-*NN problem which do not use any index to obtain the k-NN of each element. This approach solves the problem avoiding as many comparisons as possible, only comparing some database elements and taking advantage of the distance function properties. Its total cost is significantly lower than that of the naïve solution.

1 Introduction

Similarity search has become a very important operation in applications that deal with unstructured data sources. It has applications in a large number of fields. Some examples are non-traditional databases; machine learning and classification; information retrieval; image quantization and compression; computational biology; text searching; and function prediction; between others. All those applications can be formalized with the *metric space model* [6]. A metric space is composed by a universe of objects $\mathbb{U}$ and a distance function d. The distance function gives us a dissimilarity criterion to compare objects from $\mathbb{U}$. A database is a subset $S \subseteq \mathbb{U}$.

In the metric space model the similarity queries in S of any $q \in \mathbb{U}$ are usually of two types:

- *range query*: given $r \in \mathbb{R}^+$ it retrieves all the elements in S within distance r to q ($R(q, r)$), and
- *k-nearest neighbor*: given $k \in \mathbb{N}$ it retrieves the k closest elements to q in S-$\{q\}$ (k-$NN(q)$).

The k-NN(q) query is a building block for a large number of problems in a wide number of application areas. For instance, in pattern classification, the nearest-neighbor rule can be implemented with 1-NN(q) [8].

Formally, the Nearest Neighbor Graph (NNG) is a graph whose vertex set is S and with one edge from u to v whenever v is the nearest neighbor of u in S. The problem of determining the nearest neighbor of any element is often called the all-nearest neighbor problem. It could be generalized to obtain the k-NN of *all* elements of database: the *All-*k-NN problem. It is a useful operation for batch-based processing of a large distributed point dataset. Hence, it will be our focus.

Performing similarity queries on a database using a sequential scan can become impractical, either because of the database size or the cost of distance evaluations. As usually, the computation of distances represents the most significant cost in this type of searches. Hence, it is customary to use this cost as the complexity measure. For general metric spaces, there exist several methods to preprocess the database in order to reduce the number of distance evaluations [6], and then by performing n k-NN queries, avoiding the exhaustive search.

However, when the database is very large or the distance is very costly, building an index, and then performing a k-NN query for each database element could be too expensive. Therefore, in this work, we present a new method to solve the *All-*k-NN problem, which will allow us to effectively compute the kNNG. It is important to mention that this technique does not use an index to solve the problem. The cost of our approach is significantly lower than n^2, which is the number of distance calculations used by the naïve solution. Our proposal computes some distances between database objects and ingeniously takes advantage of the properties that distance function satisfies.

This paper is organized as follows: Section 2 presents a brief description of some useful concepts. Section 3 introduces our proposal, and Section 4 contains the empirical evaluation of our proposed solution. Finally, in Section 5 we conclude and discuss about possible extensions for our work.

2 Previous Concepts

In this section, we briefly state the problem in a more formal way to continue the discussion. A metric space is composed of a universe of objects $\mathbb{U}$, and a distance function $d : \mathbb{U} \times \mathbb{U} \to \mathbb{R}^+$, such that for any $x, y, z \in \mathbb{U}$, $d(x, y) > 0$ (strict positiveness), $d(x, y) = 0 \iff x = y$ (reflexity), $d(x, y) = d(y, x)$ (symmetry), and obeying the triangle inequality: $d(x, z) + d(z, y) \geq d(x, y)$. The smaller the distance between two objects is, the more *similar* they are. We have a finite database S, which is a subset of $\mathbb{U}$ and can be preprocessed. Later, given a new object q from $\mathbb{U}$ used as a query, we must retrieve all elements in S close to q, using as few distance computations as possible. In the metric space model, similarity queries are usually of two types. For a given database S with size $|S| = n$, $q \in \mathbb{U}$ and $r \in \mathbb{R}^+$: $(q, r) = \{x \in S \mid d(q, x) \leq r\}$ is known as a *range query*; and k-NN(q), denotes the *k-nearest neighbors*, formally it retrieves the set $R \subseteq S$ such that $|R| = k$ and $\forall u \in R, v \in S - R$, $d(q, u) \leq d(q, v)$. This last primitive is a fundamental tool in cluster and outlier detection [3, 9],image segmentation [1], query or document recommendation systems [2], VLSI design, spin glass and other physical process simulations [4], pattern recognition [8], and so on.

As it is aforementioned, the distance is considered expensive to compute (think, for instance, in comparing two fingerprints). Thus, the ultimate goal is to build *offline* an index to speed up *online* queries. Differents techniques to solve the problem of similarity queries have arisen to reduce these costs, usually based on data preprocessing. All those structures work based on discarding elements using the triangle inequality, and most of them use the classical divide-and-conquer approach.

A related version of the k-NN problem, perhaps less studied, is the *All-k-*NN problem. That is, if $|S| = n$, we solve the *All-k-*NN problem by efficiently retrieving the k-NN(u_i) for each u_i in S and performing less than $O(n^2)$ distance evaluations. It is a useful operation for batch-based processing of a large distributed point dataset. Consider, for example, a location-based service which recommends each user his or her nearby users, who may the candidates of new friends. Given that locations of users are maintained by the underlying database, we can generate such recommendation lists by issuing an *All-k-*NN query on the database.

Most of the solutions that have been proposed and developed for this problem use indexes. Some of them, for general metric spaces [12, 13], are based on the construction of k-nearest neighbors graphs (kNNG). The kNNG is a weighted directed graph connecting each object from the metric space to its k nearest neighbors; that is, $G(S, E)$ such that $E = \{(u, v), u, v \in S \wedge v \in k\text{-NN}(u)\}$. G connects each element through a set of arcs whose weights are computed according to the distance of the corresponding space. Building the kNNG is a direct generalization of the *all-nearest-neighbor* (*All*-1-NN) problem, which corresponds to the 1NNG construction problem. The kNNG offers an indexing alternative which requires a moderately amount of memory, obtaining reasonably good performance in the search process. In fact, in low-memory scenarios, which only allow small values of k the search performance of kNNG is better than using classical pivot-based indexing alternative. In addition, graph-based techniques offer great potential for improvements, ranging from fully dynamic graph-based indexes to specific optimizations for metric space search.

The naïve algorithm for *All-k-*NN calculates the distance function d between each $u_i \in S$ and every element of S, so it has quadratic complexity.

3 Our proposal

As it was already mentioned, performing similarity queries on a database using a sequential scan can become impractical, either because of the size of the database or because of the cost of distance calculations. Under this cost model, the ultimate goal of any similarity search technique is to solve queries doing the fewest number of distance calculations. This objective was taken into mind in this new proposal. It is important to note that our proposal achieves its goal without using any index.

This technique allows to compute the k-nearest neighbors for all the elements from the database S, to compute the kNNG. To do this, some objects $x_i \in S$ are selected, and their distance to each of the other elements y in the database are calculated ($y \in S - \{x_i\}$). Since each selected element x_i knows its distance to the rest of the objects in S, it can be used as support by some elements close to it, to find their k-nearest neighbors.

After computing the $n-1$ distances from x_i to the other $n-1$ objects in S, we obtain the set of distances: $\{d(x_i, x_1), d(x_i, x_2), d(x_i, x_3), \ldots, d(x_i, x_{n-1})\}$. The elements $\{x_1, x_2, x_3, \ldots, x_{n-1}\}$ are sorted by distance to x_i, resulting in the sequence of objects $x_{j_1}, x_{j_2}, \ldots, x_{j_{n-1}}$. The first k elements of this sequence, $x_{j_1}, x_{j_2}, \ldots, x_{j_k}$, are the k objects more similar to x_i, therefore their k-nearest neighbors, and at this time they can be reported.

Next, we calculate the k-NN of the objects $x_{j_1} \in S$ that do not yet have them. For this purpose, we consider to solve the k-NN query by a range query with decreasing radius, and to use the known fact that the search will be more efficient while more quickly we can reduce the search radius. Assuming, furthermore, that very close objects possibly share some of their closest neighbors. That is, it takes the first neighbor of x_i, which is x_{j_1}, and starts the computation of its k-NN by comparing it with the k-NN(x_i)$-\{x_{j_1}\}$. It will be noticed that x_{j_1} already knows one distance; that is, its distance to x_i because of the simmetry of d. In this way, by adding x_i to this group, we get the first k candidates for being its closest neighbors, and an initial covering radius, which is the distance of x_{j_1} to its furthest neighbor:

$$r_k(x_{j_1}) = \max\{d(x_{j_1}, x_i), d(x_{j_1}, x_{j_2}), \ldots, d(x_{j_1}, x_{j_k})\}$$

.

The knowledge acquired by x_i, when compared to all objects in the database, allows to estimate the distances between x_{j_1} and the other elements in S using triangular inequality. This estimate defines a lower bound of the actual distance between x_{j_1} and any other element of S; that is, how close an object x_j could be to x_{j_1}. By knowing the cover radius and the estimate of distances it is possible to discard some objects of S, avoiding its actual comparison with x_{j_1}. If that lower bound is greater than the current radius enclosing the k candidates for k-NN(x_{j_1}), that is $|d(x_j, x_i) - d(x_i, x_{j_1})| > r_k(x_{j_1})$, this object will be discarded because it will not be closer to x_{j_1} than the current neighbors.

Otherwise, those objects x_j whose estimated distance is less than the current covering radius; that is, they satisfy $|d(x_j, x_i) - d(x_i, x_{j_1})| < r_k(x_{j_1})$, will be directly compared with x_{j_1}. Then, the elements that could not be discarded, will be sorted from highest to lowest estimate value. If those elements when they are compared with x_{j_1} verify $d(x_j, x_{j_1}) < r_k(x_{j_1})$; that is, their real distance is smaller than the covering radius, will become part of the k-NN(x_{j_1}). Hence, the element whose distance was $r_k(x_{j_1})$, which is the current farthest one from x_{j_1}, will be discarded and it implies that the covering radius $r_k(x_{j_1})$ should be updated. When we update the covering radius, reducing it, the estimates should be rechecked to determine if we can discard some other objects. If the elements that remained as promising ones when they are directly compared with x_{j_1} verify $d(x_j, x_{j_1}) > r_k(x_{j_1})$, they must be discarded. Otherwise, they can contribute to the set k-NN(x_{j_1}) and reduce again the covering radius $r_k(x_{j_1})$.

This process ends when all the objects in S were reviewed or when the next estimate analyzed is greater than the covering radius, because all elements of S from then on will be discarded. This is because they are sorted in ascending order by their estimated distance, there will no longer be items closer to x_{j_1} than neighbors that are already known. At the end of this process, k-NN(x_{j_1}) is reported. Then, the next object is taken in the sequence of x_i neighbors that do not already have their k-NN calculated, and proceed to calculate them in the same way.

As the elements to which k-NN are calculated move away from x_i, the number of objects that can be discarded from the list of promising ones, using triangular inequality, is decreasing. It causes an increase in the number of distances to be calculated. But, as our goal is to reduce this number, at that time we must look for a new element that "helps" in the calculation of the remaining k-NN. A parameter α helps us to decide when an object becomes useful in helping to discard candidates. Each time we finish the process of associating an object with its k-nearest neighbors, we check whether the element x_i used as support still produces a good discard. When an element x_i allows to reject less than α elements, its discard capability is considered insufficient. Therefore, it must be replaced by another element from the database. We select a new x_i randomly between the elements in S that do not have their k-NN already calculated. In that case, this new x_i replace the old one, so the $(n-1)$ distances to the remaining members of S are computed. Hence, we obtain k-NN(x_i), and use x_i to assist in the calculation of k-NN(x_{j_i}) for some x_{j_i} for which x_i allows to discard more than α elements. Now, the process is repeated until all the elements in S have their k-NN.

4 Experimental Results

In order to evaluate our proposal, we perform some experiments over different metric databases. For each database we run the process over different database permutations. Besides, we test different values for the α parameter.

For the empirical evaluation of our proposal we consider a set of real-life metric spaces, all of them with a low intrinsic dimension, and with widely different histograms of distances: ***Strings:*** a dictionary of 69,069 English words. The distance is the *edit distance*, that is, the minimum number of character insertions, deletions and substitutions needed to make two strings equal.
NASA images: a set of 40,150 20-dimensional feature vectors, generated from images downloaded from NASA [1]. The Euclidean distance is used.
Color histograms: a set of 112,682 8-D color histograms (112-dimensional vectors) from an image database [2]. Any quadratic form can be used as a distance, so we chose Euclidean distance.

All these metric databases are available from `www.sisap.org` [10].

Besides, to analyze how the intrinsic dimensionality affects the behavior of our approach, we evaluated it over synthetic metric spaces, where we can control their intrinsic dimensionality. We use collections of 100,000 vectors, uniformly distributed in the unit hypercube, in dimensions 4, 8, 12, 16, 20, 24, 28 and 32. We consider the vectors in these spaces as metric objects. Thus, we do not use explicitly the information of the coordinates of each vector. In these spaces we also use Euclidean distance.

We test different values of the parameter α: $\frac{n}{10}$, $\frac{n}{5}$, $\frac{n}{4}$, $\frac{n}{3}$, and $\frac{n}{2}$). Besides, we evaluate several values of the number k of nearest neighbors to be obtained: 1, 2, 5, 10, 50, and 100. As it is aforementioned, we register the cost of our approach considering the number of calculations of distances done. However, for an easy understanding of the results we show the average number of distances needed by element.

[1] At `http://www.dimacs.rutgers.edu/Challenges/Sixth/software.html`
[2] At `http://www.dbs.informatik.uni-muenchen.de/~seidl/DATA/histo112.112682.gz`

The Figure 2 shows the costs of answering to *All-k-*NN in the space of NASA Images, Strings and Color Histograms, respectively. Each line represents the costs of obtaining k neighbors with a certain α. As it can be seen, better results are obtained when α is smaller (e.g. $\frac{n}{10}$). Possibly, in this case the distinguished objects x_i are not changed quickly and this allows reducing the number of distance calculations. However, as it can be noticed, all our alternatives need to perform much less than n distance evaluations in average by element. Therefore, the speedup obtained respect to the naive solution is very significant.

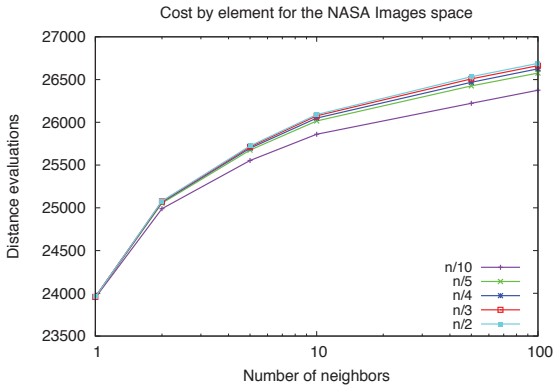
It can be regarded from these results that for higher values of α, for example $\frac{n}{2}$, as it difficult to find elements able to discarding one half of the database for each object whom estimates its distances through it. Higher values of α cause replacing more frequently the distinguished element owing to it does not satisfy the discard quality. Consequently, the new candidate must calculate again its $(n-1)$ distances. Surprisingly, it do not occur in the String space. In this case, all costs are very similar, no matter the value of α.

It is remarkable that, throughout the process up to half of the database, the number of distinguished objects used is approximately the half of the elements that are obtained its k-NN. That is, each distinguished element is actually useful for calculating, on the average, the k-NN for two more objects, and then it loses its discard quality. When the process achieves the last database elements; that is, most of the objects already have their k-NN, the remaining objects to be used as distinguished elements are not so good for discard. Hence, they must be replaced quickly, increasing in consequence the number of distance calculations.

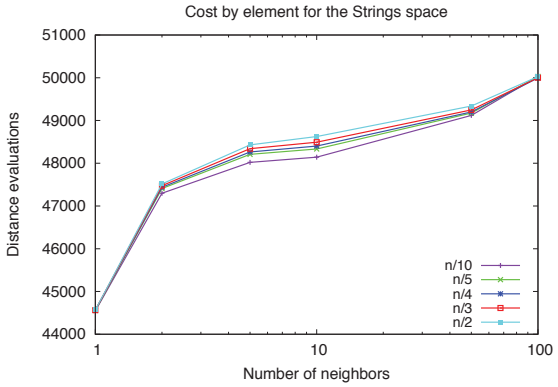
Comparison with others methods: We compare the performance of our technique with respect to several representative indexes: List of Clusters [5], Distal Spatial Appoximation Trees [7], and a generic pivot index [6]. All of them have their implementation available from `www.sisap.org` [10]. As the other methods considered need to build the index to perform the searches, we compute their costs considering both the construction cost of the index and the search cost of all k-NN operations.

The Figure 2(a), Figure 2(b) and Figure 2(c) illustrate the comparison of the costs of our proposal and the other considered indexes. We use LC for the List of Clusters, DiSAT for the Distal Spatial Appoximation Trees, and Piv for the generic pivot index to name the lines of these indexes. Particularly, the values that accompany LC indicate the different cluster sizes used to build the index, and for Piv the number of pivots. We choose to show for each index its better alternatives. For our approach we use the better values of α empirically determined: $\frac{n}{10}$ and $\frac{n}{5}$. We label each α option with the corresponding value.

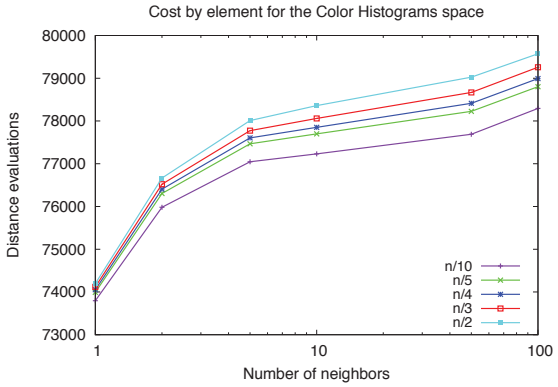
In the Strings space, only the pivot-based index is more expensive than our method, the other indexes beat us for all the k values. We only get closer to LC and DiSAT for $k = 100$. For the Nasa Images space, our technique is surpassed by all indexes and for all the values of k, we consider that it is because this space has the lowest intrinsic dimensionality. Moreover, in the Color Histograms space, we again beat pivot index but only for $k > 10$, and LC and DiSAT always are better than us. It is remarkable that all the Real spaces considered do not have high intrinsic dimensionality [11].



(a) Nasa Images space.

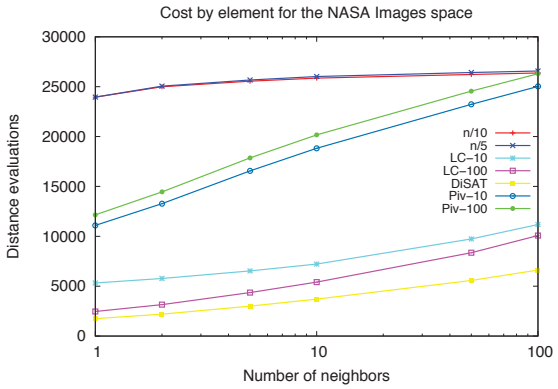


(b) Strings space.

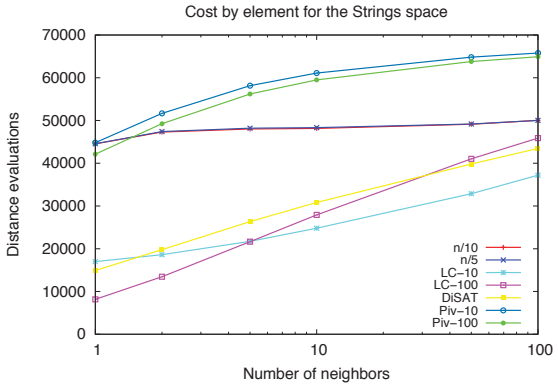


(c) Color histograms space

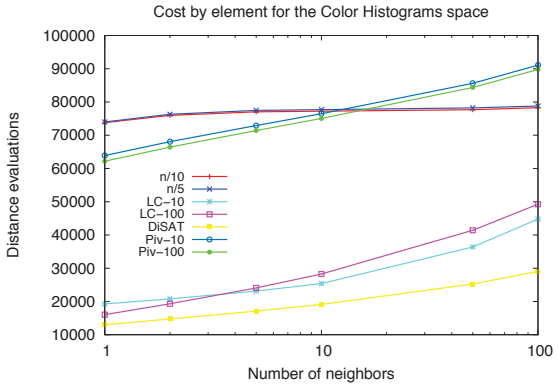
Fig. 1. Costs of *All-k*-NN in Real spaces.



(a) Nasa Images space.



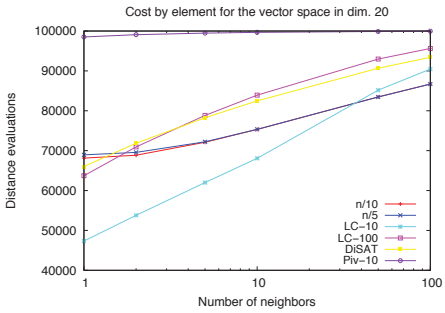
(b) Strings space.



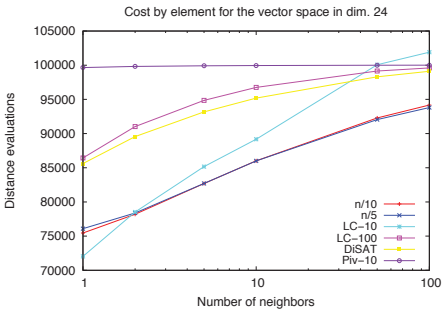
(c) Color histograms space

Fig. 2. Comparison of costs to solve *All-k*-NN on the Real spaces considered.

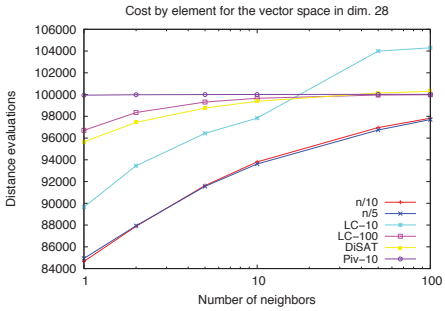
The Figure 3 depicts the same experiments on the synthetic spaces. For lack of space, we only show the results for the vector spaces in higher dimensions; that is, dimension from 20 to 32. We do not show the results for vector spaces in lower dimensions because the behavior of all the methods are similar to those of the already depicted for the real spaces. As it can be observed, as dimension grows our proposal becomes better than more and more of the other alternatives. For example, for dimensions from 24 (Figures 3(b), 3(c), and 3(d)), we can overcome all the other indexes, for almost all number of neighbors.



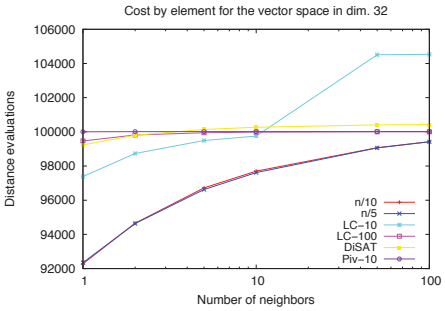
(a) Dimension 20.



(b) Dimension 24.



(c) Dimension 28.



(d) Dimension 32.

Fig. 3. Comparison of costs to obtain *All-*k-NN on synthetic spaces.

Although LC and DiSAT are indexes that have good behavior on medium to high dimensional metric space, they also degrade their performance when dimension is higher than 24. Hence, it is important to remark that our approach seems more resistant to the curse of dimensionality.

5 Conclusions

An extended version of determining the k-NN of an element, is to solve the *All-*k-NN problem. Solving this problem is useful, for example, to build the metric index named the graph of k-nearest neighbors. Until this moment, the only way to solve this problem,

without using an auxiliary index, is via the naïve solution of comparing each database element with each other one. In this paper we propose and tested an alternative approach to solve this problem. Our proposal is very efficient with respect this naïve solution, because it needs to perform much less tan n distance evaluations by element.

Our results are preliminary and encouraging. It can be affirmed that, on low dimensional metric spaces, it may be preferable for solving the *All-*k-NN problem to build an index and then perform the search for the k-NN of each edatabase lement. By the other hand, on medium to high dimensional spaces, the curse of dimensionality affects more clearly the performance of the different indexes than of our proposal. Therefore, as the dimension grows our proposal achieves a better performance than other alternatives that also solve this problem.

As future works, we consider analyzing different options to select the distinguished elements for each process step, and how this selection affects the method performance. Besides, we also plan to study how to take advantage of the available memory space to cache some distances calculated and avoid the recalculation of them.

References

1. Neculai Archip, Robert Rohling, Peter Cooperberg, Hamid Tahmasebpour, and Simon K. Warfield. Spectral clustering algorithms for ultrasound image segmentation. In James S. Duncan and Guido Gerig, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2005*, pages 862–869, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
2. Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. Query clustering for boosting web page ranking. In Jesús Favela, Ernestina Menasalvas, and Edgar Chávez, editors, *Advances in Web Intelligence*, pages 164–175, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
3. M. Brito, E. Chávez, A. Quiroz, and J. Yukich. Connectivity of the mutual k-nearest neighbor graph in clustering and outlier detection. *Statistics & Probability Letters*, 35(4):33–42, 1996.
4. P. Callahan and R. Kosaraju. A decomposition of multidimensional point sets with applications to k nearest neighbors and n body potential fields. *JACM*, 42(1):67–90, 1995.
5. E. Chávez and G. Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363–1376, 2005.
6. E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.
7. Edgar Chávez, Verónica Ludeña, Nora Reyes, and Patricia Roggero. Faster proximity searching with the distal sat. *Information Systems*, 59:15–47, 2016.
8. R. Duda and P. Hart. Pattern classification and scene analysis. *John Wiley & Sons*, 1973.
9. David Eppstein and Jeff Erickson. Iterated nearest neighbors and finding minimal polytopes. In *An Int. Journal of Mathematics and Computer Science*, volume 11–3, pages 321–350, 1994.
10. Karina Figueroa, Gonzalo Navarro, and Edgar Chávez. Metric spaces library, 2007. Available at `http://www.sisap.org/Metric_Space_Library.html`.
11. G. Navarro, R. Paredes, N. Reyes, and C. Bustos. An empirical evaluation of intrinsic dimension estimators. *Information Systems*, 64:206–218, 2017.
12. R. Paredes. *Graphs for Metric Space Searching*. PhD thesis, University of Chile, 2008.
13. R. Paredes, E. Chávez, K. Figueroa, and G. Navarro. Practical construction of k-nearest neighbor graphs in metric spaces. In *Proc. 5th Workshop on Efficient and Experimental Algorithms (WEA)*, LNCS 4007, pages 85–97, 2006.