- ORIGINAL ARTICLE -

All Near Neighbor Graph Without Searching

Edgar Chávez[1], Verónica Ludueña[2], Nora Reyes[2], and Fernando Kasián[2]

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

Abstract

Given a collection of n objects equipped with a distance function $d(\cdot,\cdot)$, the Nearest Neighbor Graph (NNG) consists in finding the nearest neighbor of each object in the collection. Without an index the total cost of NNG is quadratic. Using an index the cost would be sub-quadratic if the search for individual items is sublinear. Unfortunately, due to the so called *curse of dimensionality* the indexed and the brute force methods are almost equally inefficient. In this paper we present an efficient algorithm to build the Near Neighbor Graph (nNG), that is an approximation of NNG, using only the index construction, without actually searching for objects.

Keywords: Near Neighbor Graph, Proximity Search, Clustering, Metric Indexing

1 Introduction

The nearest neighbor graph (NNG) is used in areas as diverse as machine learning, statistics, optimization and wireless communication networks and genomics. A collection of sites or objects is given, and the objective is to find, for each object the nearest neighbor in the collection. If the objects are points in the plane, there are well known efficient algorithms using, for example, the Voronoi diagram [1]. For collections of points in higher dimensions it is not possible to efficiently build the NNG because the algorithms have an exponential dependence on the dimension of the points. If the objects are more abstract (think for example in strings representing genes in computational biology, documents in a collection, digital images) the problem becomes more difficult to solve. Fortunately there is a well accepted model for proximity problems in general, described below.

Citation: E. Chávez, V. Ludueña, N. Reyes and F. Kasián. *All Near Neighbor Graph Without Searching*. Journal of Computer Science & Technology, vol. 18, no. 1, pp. 61-67, 2018.
DOI: 10.24215/16666038.18.e07
Received: March 01, 2017 **Revised:** June 01, 2017 **Accepted:** August 30, 2017.

Proximity searching can be formalized using the *metric space model* [2]. 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 $\mathbb{U}$.

Two basic primitives in similarity searching, on metrics spaces, are: *range query* and *k-nearest neighbor*. 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)'s [3].

Let be $S \subseteq \mathbb{U}$ a given database, the Nearest Neighbor Graph (NNG) is a graph with S as the vertex set and with an edge from u to v whenever v was the nearest neighbor of u. It is often called the *All Nearest Neighbor Problem*. It could be generalized to retrieve 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.

It is customary to use the number of distance evaluations as the complexity measure, because he distance is considered to have the leading cost in the problem. For general metric spaces there are several methods to preprocess the database, in order to reduce the number of distance evaluations [2], and then avoiding the exhaustive search. However, when the database is very large or the distance is too costly, building an index and then performing an exact k-NN query for each database element could be very expensive too. In these cases, an alternative is to settle for the response to approximate similarity queries, which will save runtime at the price of losing accuracy in the response. But, it still could be very expensive, even more if we consider that in this way many calculated distances during the index construction are wasted, because queries do not take complete advantage of these calculations. Thus, it can be considered that an even cheaper way to calculate the approximate nearest neighbors could use directly the distances calculated during the index building, in order to approximate the response, especially if there is a reasonable chance that during the construction each element would be compared with very close elements. Such is the case of the Distal Spatial Approximation Tree (*DiSAT*) [4].

In this paper we present a novel approach to computing the nNG. The central idea is to use the index itself as an approximation of the NNG. We selected a novel index for that task, the Distal Spatial Approximation Tree (*DiSAT*). After a few rebuilds, we will have a fairly good nNG without performing individual searches, and at a fraction of the cost building the index and querying every object. We present experimental results supporting our claims. A preliminary version of this paper appears in [5].

It is worth noticing that research in metric indexing have a big gap between theory and practice. Complexity models parametrized by the intrinsic dimension of the metric space does not exist. First of all, there is not a usable dimension definition capturing the difficulty of practical implementations. The most recent attempt to evaluate intrinsic dimension estimators is [6] and it does cover all the cases.

This paper is organized as follows: Section 2 presents a brief description of some useful concepts. In Section 3 we give a description of the *DiSAT*. Section 4 presents our proposal, and Section 5 contains the empirical evaluation of our proposed solution. Finally, in Section 6 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 by 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, 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 from $\mathbb{U}$ (a query q), we must retrieve all elements found in S close to q, using as few distance computations as possible. Similarity queries, in metrics spaces, are usually of two types, for a given database S with size $|S| = n$, $q \in \mathbb{U}$ and $r \in \mathbb{R}^+$: $(q,r) = \{x \in S \mid d(q,x) \leq r\}$ denote 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 primitive is a fundamental tool in cluster and outlier detection [7, 8], image segmentation [9], query or document recommendation systems [10], VLSI design, spin glass and other physical process simulations [11], pattern recognition [3], and so on.

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

A version of the k-NN problem, perhaps less studied, is the *All-k*-NN problem. That is, if $|S| = n$, get the *All-k*-NN is retrieve, efficiently, the k-NN(u_i) for each u_i in S, 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. In a centralized database environment, we can use the existing *All-k*-NN algorithms.

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-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 [12, 13].

The naïve algorithm for kNNG calculates the distance function d between each $u_i \in S$ and every element of S, so it has quadratic complexity. Even, when we model similarity as a metric space, we are already approximating the real retrieval need of users. In fact, given a dataset, we can use several distance functions, each of them considering some aspects of objects and neglecting others. Likewise, when we design a model to represent real-life objects, we usually lose some information. Moreover, even if we find the proper metric and a lossless object representation, there are high-dimensional metric spaces where solving similarity queries requires reviewing almost all the dataset no matter what strategy is used. In addition, in many applications, the query speed is much more important than its precision. That is, users want a fast response to their queries and will even accept approximate results (as far as the number of false hits is *moderate*). This has given rise to a new approach to the similarity search problem: we try to find the objects relevant to a given query with high probability.

The goal of the approximate search is to *significantly* reduce search times by allowing some "errors" in the query outcome. This alternative to the "exact" similarity searching is called *approximate similarity searching* [14], and it includes approximate and prob-

abilistic algorithms. The general idea of approximate algorithms is to allow a relaxation on the precision of the query in order to obtain a speed-up the query time complexity. In addition to the query, a precision parameter ε is specified to control how far away we want the outcome of the query from the correct result. A reasonable behavior for this kind of algorithm is oncoming asymptotically to the correct answer as ε get closer to zero, and complementarily, speed up the algorithm, losing precision, as ε moves in the opposite direction. Hence, a successful approximation technique must have a good balance quality/time [15].

To evaluate the performance of an approximate similarity search it must be considered: improvement in efficiency and accuracy of approximate results. The good approximation algorithms should offer large improvements in efficiency and high accuracy of approximate results. But, there must be a trade-off between both. The *improvement in efficiency* can be stated as:

$$\frac{Cost(Q)}{Cost^A(Q)}$$

where $Cost(Q)$ and $Cost^A(Q)$ are the number of distance evaluations needed to perform an exact query and an approximate query Q, respectively. Q can be a range or a k-NN query.

When performing approximate searches we must evaluate the retrieval effectiveness of the method. In an information-retrieval scenario, two measures are used as performance measures: *Recall* and *Precision*. Recall is defined as the number of *relevant objects* retrieved by a search divided by the total number of existing relevant objects. While precision is defined as the number of relevant objects retrieved by a search divided by the total number of objects retrieved by that search. If the R represents the result-set of an exact similarity search query and R^A the result-set returned by the approximation query, these measures can be formally stablished as:

$$Precision = \frac{|R \cap R^A|}{|R^A|} \quad \text{and} \quad Recall = \frac{|R \cap R^A|}{|R|}.$$

As we are focused on k-nearest neighbor searches, we can observe that given k the precise and approximate response sets both have a fixed cardinalities: k. Thus, the recall and precision measures always return identical values. Therefore, as follows we only use the precision measure.

Another measure to evaluate is the *relative error on distances* [16]. Relative error on distances compares the distances from a query object to the object in the exact and approximate results:

$$\frac{d(o_A,q) - d(o_R,q)}{d(o_R,q)} = \frac{d(o_A,q)}{d(o_R,q)} - 1$$

where o_A is the approximate nearest neighbor and o_R is the real nearest neighbor.

By this way, we computed the ratio between the distance to the object reported by the approximate algorithm and the real nearest neighbor minus 1. In our case, because we want to compute the *All*-1-NN, the average of the resulting quantities over all the elements is called the *average relative error on distances*.

3 Distal Spatial Approximation Tree

The Spatial Approximation Tree (*SAT*) is a proposed data structure [17] based on a concept: approach the query spatially. It has been shown that the *SAT* gives better space-time tradeoffs than the other existing structures on metric spaces of high dimension or queries with low selectivity [17], which is the case in many applications. The Dynamic Spatial Approximation Tree (*DSAT*) [18] is an online version of the *SAT*. It is designed to allow dynamic insertions and deletions without increasing the construction cost with respect to the *SAT*. It is very surprising that *DSAT* is more efficient for searching than the *SAT*. For the *DSAT* the database is unknown beforehand and the objects arrive to the index at random as well as the queries. Then, it arises the *Distal Spatial Approximation Trees* (*DiSAT*) [4] that improves regarding search performance over *SAT* and *DSAT*. *DiSAT* obtains better behavior on searches just by considering a different construction heuristic from *SAT*, but it maintains the same principles of searching and building process.

The *SAT* is built as follows. An element a is selected as the root, and it is connected to a set of neighbors $\mathrm{N}(a)$, defined as a subset of elements $x \in S$ such that x is closer to a than to any other element in $\mathrm{N}(a)$. The other elements (not in $\mathrm{N}(a) \cup \{a\}$) are assigned to their closest element in $\mathrm{N}(a)$. Each element in $\mathrm{N}(a)$ is recursively the root of a new subtree containing the elements assigned to it. From the previous definition of the *SAT*, the starting set for neighbors of the root a, $\mathrm{N}(a)$ is empty. Particularly, *SAT* selects the first neighbor between all the elements in $S - \{a\}$, as its closest element and then considers if any other element can become a neighbor by analyzing them in an ordering from nearest to farthest. However, it could be possible to select any database element as the first neighbor. Inversely, *DiSAT* selects the first neighbor as its farthest elements in $S - \{a\}$ and uses the reverse ordering of the other elements to analyze if any of them can become a neighbor. Nevertheless, the same searching algorithm can be used on both trees because both uses the same condition to be a neighbor [17, 4]. This heuristic change of *DiSAT* increases the discarding power of the *SAT* by selecting distal nodes instead of the proximal nodes proposed in the original paper. Please note that this heuristic is the exact opposite of the original ordering in the construction of the *SAT*. Besides, *DiSAT* and *SAT* have the advantage of not having to tune any parameter.

Algorithm 1 gives a formal description of the construction of our data structure. As it can be seen in

Algorithm 1 Algorithm to build a *DiSAT* for $S \cup \{a\}$ with root a.

```
BuildTree(Node a, Set of nodes S)
1. N(a) ← ∅    /* neighbors of a */
2. R(a) ← 0    /* covering radius */
3. For v ∈ S in decreasing distance to a Do
4.    R(a) ← max(R(a), d(v,a))
5.    If ∀b ∈ N(a), d(v,a) < d(v,b) Then
6.       N(a) ← N(a) ∪ {v}
7. For b ∈ N(a) Do S(b) ← ∅
8. For v ∈ S − N(a) Do
9.    c ← argmin_{b∈N(a)} d(v,b),   S(c) ← S(c) ∪ {v}
10. For b ∈ N(a) Do BuildTree(b, S(b))
```

line 3, *DiSAT* uses farthest-to-nearest order from the root. Searching is done with the standard procedure. When working with hyperplanes to perform data separation it is advisable to use object pairs far from each other as documented in [2] for the *GNAT* and *GHT* data structures. Using the above observations, it is possible to ensure a good separation of the implicit hyperplanes by selecting the first neighbor as the farthest element to the root, and as a secondary effect the covering radii of neighbors are smaller than in *SAT*. Thereby, the partition induced by the *DiSAT* construction on the space has the nice property of obtaining a good data separation, that is useful for our approach to *All*-1-NN.

4 Our proposal

We decided to attack the problem of the 1-nNG, i. e. to retrieve a near neighbor of *each* item in the database without comparing it against all the others. The idea of this proposal is maintaining for each item, during *construction* of the index, the closest element seen. This makes sense because the root *knows* the distances to every object in the database and as we descend the tree less information will be stored. No searches will be performed.

We used the *DiSAT* as the choice index, because as we previously mentioned it do not require any parameter and produces a very good partition on the database. During tree construction we maintain for each object its closest element seen among all which it was compared with. When the construction finalizes we can retrieve for each $o_i \in S$ a near neighbor x and its distance $d(o_i,x)$, where 1-nN$(o_i) = \{x\}$.

After the *DiSAT* construction not all the nodes will have its nearest neighbors. Our hypothesis is that by rebuilding the index a few times we can improve the quality of the approximation.

5 Experimental Results

To evaluate our proposal, the experiments consisted in obtaining the approximate *All*-1-NN$_A$ of each element, only by building the *DiSAT*. We computed the true *All*-1-NN using the *DiSAT* as an auxiliary index, and searched for the nearest neighbor of each item in the database. It is cheaper than a brute force approach.

The total cost of the 1-NNG considers all distance evaluations performed for construction and searching. All our results are averaged over 10 executions on different permutations of the datasets. As we provide an approximate answer, we need to analyze its quality by calculating precision, recall, relative error of differences and complexity of each option.

For the experiments, we consider a set of real-life metric spaces with widely different histograms of distances available from `www.sisap.org` [19]:

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.

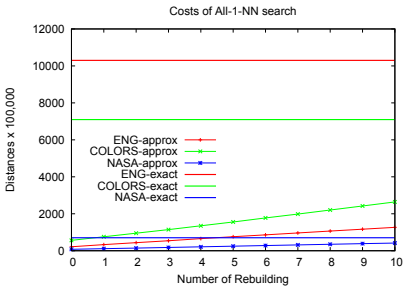
NASA images: a set of 40,700 20-dimensional feature vectors, generated from images downloaded from NASA.The Euclidean distance is used.

Color histograms: a set of 112,682 8-D color histograms (112-dimensional vectors) from an image database. Any quadratic form can be used as a distance, so we chose Euclidean distance.

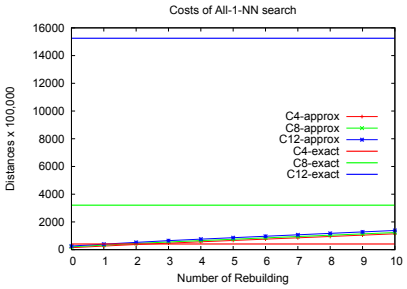
It is interesting to analyze how the intrinsic dimensionality affects the behavior of our approach. To this end we experimentally evaluated the different solutions over synthetic metric spaces where we can control the intrinsic dimensionality. We use collections of 100,000 vectors of dimensions 4, 8, and 12, uniformly distributed in the unit hypercube. We do not use explicitly the information of the coordinates of each vector. In these spaces we also use Euclidian distance.

Figure 1 illustrates the costs of the exact solution against our approximate proposal. We show the cost measured in distance evaluations for each rebuilding of the index. Thus, the first construction is indicated by 0 rebuilding, 1 rebuilding means that we have construct firstly a *DiSAT* and secondly a *DiSAT* from the balls obtained with the first construction, and so on. As can be noticed, the cost of the exact solutions is shown as constant, because it do not depend of any rebuilding. Figure 1(a) shows the costs for the three real metric spaces. For example, in the plots we name STRINGS-exact the cost of 1-NNG and STRINGS-approx the cost of our 1-nNG solution, for the space of Strings. Besides, as it can be seen, we use the same color for both costs on the same metric space. Alike, Figure 1(b) depicts the same experiments on the three synthetic spaces, designiting the spaces of coordinate vectors in dimensions 4, 8, and 12 as C4, C8, and C12, respectively.

As it can be noticed, our proposal is significantly less expensive to perform in almost all the metric spaces used than the exact solution. Only, in the space of vectors in dimension 4, the exact alternative surpasses, although not significantly, our solution from the second rebuilding onwards. In order to show

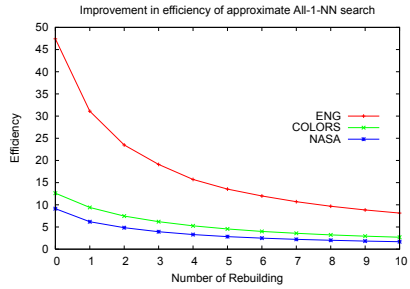


(a) Real spaces.

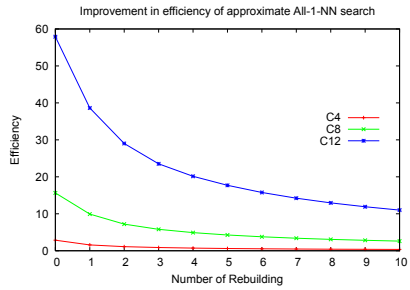


(b) Synthetic spaces.

Figure 1: Comparison of costs of obtaining 1-NNG and 1-nNG, for all metric spaces considered.



(a) Real spaces.



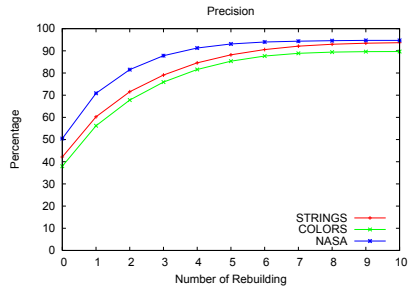
(b) Synthetic spaces.

Figure 2: Improvement in efficiency of 1-nNG, for all metric spaces considered.

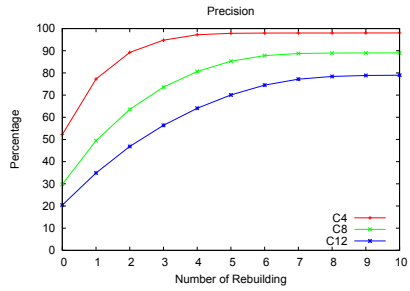
more clearly the improvement of costs, Figure 2 illustrates the improvements in efficiency obtained with our approximate solutions as we made more rebuildings of the index. Figure 2(a) shows that in all real the approximate method obtains a very significant efficiency. On the other hand, Figure 2(b) depicts the efficiency achieved over the three synthetic metric spaces. In this case we can observe that the improvement in efficiency is higher as dimension grows, and on dimensions 8 and 12 is always important, but on dimension 4 is negligible.

In Figure 3 we show the precision obtained with each reconstruction. After the fourth reconstruction, we can observe that the answer exceeds 80% hits in the three real metric spaces (Figure 3(a)). However, in the synthetic spaces it needs more reconstructions to achieve a reasonable precision. Figure 1 shows that there is still some slack to improve, and that a good solution can be obtained using only a fraction of the needed distance computations using the brute force approach.

We estimate the quality of the approximate solution by measuring the average error on distances; that is the average of differences between the distance to the approximate nearest neighbor obtained with our method and to the actual nearest neighbor of each element. Figure 4 exhibits the average error on distances obtained versus the number of rebuilding, for the two kinds of metric spaces used. In the real world metric spaces, Figure 4(a) the distance error decreases fast. In most of spaces the distance error from the first construction is almost zero, one exception is the Dictionary possibly because it uses a discrete distance.

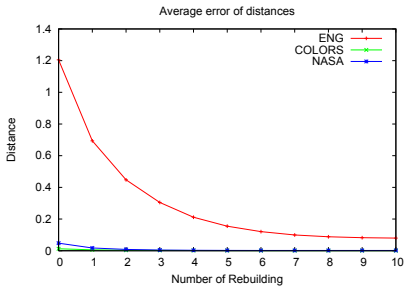


(a) Real spaces.

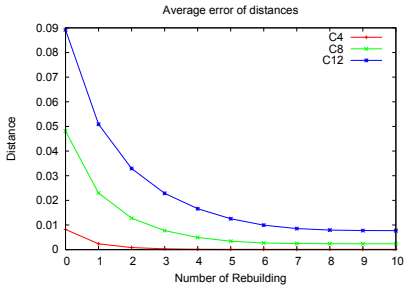


(b) Synthetic spaces.

Figure 3: Precision of the answer of 1-nNG, for all metric spaces considered.

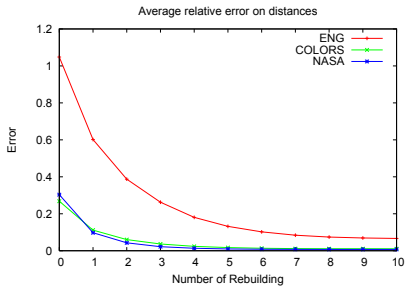


(a) Real spaces.

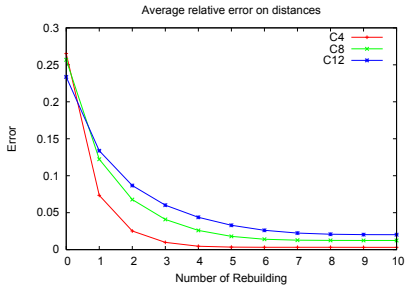


(b) Synthetic spaces.

Figure 4: Average error on distances of the answer of 1-nNG, for all metric spaces considered.



(a) Real spaces.



(b) Synthetic spaces.

Figure 5: Average relative error on distances of the answer of 1-nNG, for all metric spaces considered.

On the other hand, Figure 4(b) shows the error for the three synthetic spaces used. As it can be seen, as dimension grows the average error on distance is greater, but all of them decreases significantly with the reconstructions. For instance, in the space of vectors in dimension 4 (C4) the distance error begins lower than 0.01 and gets close to zero from the second reconstruction.

We also evaluate the quality of the approximate solution by measuring the average relative error on distances. Figure 5 shows the average relative error obtained, versus the number of rebuilding. With the real metric spaces the error decreases fast Figure 5(a), and from the third reconstruction the error is almost zero. Again the Dictionary is one exception, we believe this is due to the discrete distance.

Figure 5(b) shows the error behavior in three synthetic spaces. As it can be seen, as dimension grows the average error decreases more slowly with the reconstructions. For instance, in the space of vectors in dimension 4 (C4) the percentage of error begins lower than 0.3 and achieves close to zero values with the fourth reconstruction.

5.1 Another Improvements

We can notice that even if we want to obtain the 1-NNG, we could take advantage of the information obtained with our approach in order to speed up its solution. For this moment, a good approach to solve efficiently 1-NNG consists of building an index and then performing a 1-NN search of each element in S.

However, as it is known that 1-NN search can be performed more quickly if we know one initial distance to prune the searches, we can to take advantage of the information obtained during the construction of the *DiSAT* in order to accelerate every 1-NN search on the *DiSAT*. That is, we can use for each $o_i \in S$ its 1-nN$(o_i) = x$ and its distance $d(o_i, x)$ obtained, to prune the search of the exact 1-NN(o_i) on the *DiSAT*.

It is still possible to improve our approach to obtain the 1-nNG in terms of distance costs if we consider that with our proposal the better quality of the near neighbor is obtained for elements that are located on the top of the *DiSAT*. The reason is that these objects are compared with more elements of S, so the chance of obtaining its actual nearest neighbor is greater. Therefore, we can save some distances if we stop the rebuilding of the *DiSAT* at some height. In this case we can improve the first near neighbor obtained during the first construction of *DiSAT* for certain subset of S. Then, we can repeat this abbreviated process several times depending of the number of distance evaluations that we are willing to spend.

6 Conclusions

In this paper we tested an alternate approach to computing an approximation to the 1-NNG, that is we obtain a 1-nNG, by using a simple heuristic. We have designed an algorithm able to approximately solve 1-nNG with a low cost, a very good accuracy, and low error. Our algorithm is based on the construction of the *DiSAT*, an index that was originally proposed only for the common similarity queries. Therefore, in addition to obtaining a good method for solving an approximation of 1-NNG, we have expanded the range

of applications of the *DiSAT*.

The novelty of our proposal is that no searches are performed, but only the distances computed during the construction of the *DiSAT* are used. Our results are preliminary and encouraging. We obtained good performance with low and medium dimensionality databases, we are aiming at improving the results to tackle higher dimensions.

Competing interests

The authors have declared that no competing interests exist.

References

[1] C.-H. Liu, E. Papadopoulou, and D.-T. Lee, "The k-nearest-neighbor voronoi diagram revisited," *Algorithmica*, vol. 71, pp. 429–449, Feb. 2015.

[2] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín, "Searching in metric spaces," *ACM Computing Surveys*, vol. 33, pp. 273–321, Sept. 2001.

[3] R. Duda and P. Hart, "Pattern classification and scene analysis," *John Wiley & Sons*, 1973.

[4] E. Chávez, V. Ludeña, N. Reyes, and P. Roggero, "Faster proximity searching with the distal sat," *Information Systems*, vol. 59, pp. 15–47, 2016.

[5] E. Chávez, V. Ludeña, N. Reyes, and F. Kasián, "Approximate nearest neighbor graph via index construction," pp. 824–833, 2016.

[6] G. Navarro, R. Paredes, N. Reyes, and C. Bustos, "An empirical evaluation of intrinsic dimension estimators," *Inf. Syst.*, vol. 64, pp. 206–218, Mar. 2017.

[7] 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*, vol. 35, no. 4, pp. 33–42, 1996.

[8] D. Eppstein and J. Erickson, "Iterated nearest neighbors and finding minimal poly-topes," vol. 11, pp. 321–350, 1994.

[9] N. Archip, R. Rohling, P. Cooperberg, H. Tahmasebpour, and S. K. Warfield, "Spectral clustering algorithms for ultrasound image segmentation," vol. 3750, pp. 862–869, 2005.

[10] R. Baeza-Yates, C. Hurtado, and M. Mendoza, "Query clustering for boosting web page ranking," pp. 164–175, 2004.

[11] P. Callahan and R. Kosaraju, "A decomposition of multidimensional point sets with applications to k nearest neighbors and n body potential fields," *JACM*, vol. 42(1), pp. 67–90, 1995.

[12] R. Paredes, *Graphs for Metric Space Searching*. PhD thesis, University of Chile, Chile, July 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, pp. 85–97, 2006.

[14] P. Ciaccia and M. Patella, "Approximate and probabilistic methods," *SIGSPATIAL Special*, vol. 2, no. 2, pp. 16–19, 2010.

[15] M. Patella and P. Ciaccia, "Approximate similarity search: A multifaceted problem," *J. Discrete Algorithms*, vol. 7, no. 1, pp. 36–48, 2009.

[16] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *J. ACM*, vol. 45, pp. 891–923, Nov. 1998.

[17] G. Navarro, "Searching in metric spaces by spatial approximation," *The Very Large Databases Journal (VLDBJ)*, vol. 11, no. 1, pp. 28–46, 2002.

[18] G. Navarro and N. Reyes, "Dynamic spatial approximation trees," *Journal of Experimental Algorithmics*, vol. 12, pp. 1–68, 2008.

[19] K. Figueroa, G. Navarro, and E. Chávez, "Metric spaces library," 2007. Available at `http://www.sisap.org/Metric_Space_Library.html`.