Approximate Nearest Neighbor Graph via Index Construction

Edgar Chávez, Verónica Ludueña, Nora Reyes, and Fernando Kasián

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

Abstract. Given a collection of objects in a metric space, the Nearest Neighbor Graph (NNG) associate each node with its closest neighbor under the given metric. It can be obtained trivially by computing the nearest neighbor of every object. To avoid computing every distance pair an index could be constructed. Unfortunately, due to the *curse of dimensionality* the indexed and the brute force methods are almost equally inefficient. This bring the attention to algorithms computing approximate versions of NNG.
The DiSAT is a proximity searching tree. It is hierarchical. The root computes the distances to all objects, and each child node of the root computes the distance to all its subtree recursively. Top levels will have accurate computation of the nearest neighbor, and as we descend the tree this information would be less accurate. If we perform a few rebuilds of the index, taking deep nodes in each iteration, keeping score of the closest known neighbor, it is possible to compute an Approximate NNG (ANNG). Accordingly, in this work we propose to obtain de ANNG by this approach, without performing any search, and we tested this proposal in both synthetic and real world databases with good results both in costs and response quality.

Keywords: similarity search, databases, metric spaces, approximate search

1 Introduction

Proximity searching consists in finding objects from a collection near a given query. The literature is vast and there are many specializations of the problem. Similarity search has become a very important operation in applications that deal with unstructured data sources. This has applications in a large number of fields. Some examples are non–traditional databases, text searching, information retrieval, machine learning and classification, image quantization and compression, computational biology, and function prediction. 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 $\mathbb{U}$.

Similarity queries, in metrics spaces, are usually of two types, for a given database $S \subseteq \mathbb{U}$, a query $q \in \mathbb{U}$, and $r \in \mathbb{R}^+$: *range query*: retrieves all elements within distance r to q in S; and k*-nearest neighbor*: retrieves the k closest elements to q in S-$\{q\}$. 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 [9].

The Nearest Neighbor Graph (NNG) is a graph with S the vertex set, with an edge from u to v whenever v is 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, this will be our focus.

As the distance is considered expensive to compute, it is customary to use the number of distance evaluations as the complexity measure. For general metric spaces, there exist a number of 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 too costly, building an index and then performing an exact k-NN query for each database element could be very expensive. 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 (*DiSAT*).

Therefore, in this work we present a new method to solve the version approximate of the *All*-1-NN problem. This is the particular case of approximate *All-*k-NN problem, when $k = 1$ (*All*-1-NN$_A$), that uses the construction of a *DiSAT*, without performing any search. Besides, when the precision obtained with the response is not good enough, we propose an inexpensive way to continue improving it, even without to carry out any similarity search.

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 [4, 10], image segmentation [1], query or document recommendation systems [3], VLSI design, spin glass and other physical process simulations [5], pattern recognition [9], 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. Differents 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 classical 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.

Some solutions to this problem have been proposed and developed for general metric spaces [14, 15], based on the construction of the *k-nearest neighbors graph* (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-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.

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. 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 we use. In addition, in many applications, the efficiency of the query execution is much more important than effectiveness. That is, users want a fast response to their queries and will even accept approximate results (as far as the number of drops and 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. An intuitive notion of what this approach aims is that it attempts not to miss many relevant objects at query time.

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* [8], and it includes approximate and probabilistic 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. Therefore, the success of an approximation technique is based on the compromise quality / time [16].

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 expressed 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 must evaluate the retrieval effectiveness of a 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:

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

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* [2]. 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 resulting quantity of the average over all the database elements is called the *average relative error on distances*.

3 The Distal Spatial Approximation Tree

The Spatial Approximation Tree (*SAT*) is a proposed data structure [12] 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 [12], which is the case in many applications. The Dynamic Spatial Approximation Tree (*DSAT*) [13] 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*) 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 construction 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 [12, 7]. 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 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 [6] 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.

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))
```

4 Our proposal

As it is aforementioned, we decided to attack the problem of the approximate *All*-1-NN, i. e. to retrieve in an approximate way the 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 until this moment, obtaining its aproximate near neighbor without any search in the index.

We use, as auxiliar structure, a *DiSAT* which 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 with which it was compared. When the construction finalizes we can retrieve for each $o_i \in S$ the approximate nearest neighbor x and its distance $d(o_i, x)$, where 1-NN$_A(o_i)) = \{x\}$.

Then, we consider the outcomes as balls centered in o_i with radii $r = d(o_i, x)$. Elements o_i whose balls have larger radii, have their 1-NN$_A(o_i)$ farther away than the rest. This may be its real closest neighbor or not. If it is not its real closest neighbor the reason could be that it has not be compared against enough database objects, and so it is the best it got. We know that tree root compares with each database element, and its neighbors with most of them, but as we go down into the tree, the elements are compared with less objects each time. Then, by trying to improve mainly the neighbors of whose have large balls, we decide ordering all balls, in a decreasing order of its radii, and to rebuilt the *DiSAT* taking the elements in that order [1], the object in the first ball will be the tree root.

In this manner, we assure that by comparing the new root with all the database elements it will achieve its real neighbor. In this way the elements that follow in the order can become the neighbors of the root and thus it could obtain better nearest neighbors. Each time we compare an element o_i with another y that could be a better neighbor, we check if $d(o_i, y) < d(o_i, x)$, being x its current near neighbor, in wich case we update its approximate neighbor as y. If we are willing to pay a more distance calculations, we can repeat this process several times, trying to obtain as good approximate neighbors

[1] In fact, we can avoid the ordering of the elements with respect to the tree root for decreasing the CPU time.

as we can. Finally, we report the better approximate nearest neighbor achieved for each database element. As it can be noticed, we never perform any search on the index.

5 Experimental Results

The experiments consisted in obtaining the approximate *All*-1-NN of each element, only building the *DiSAT*. On the other hand, the exact 1-NN were calculated, building the *DiSAT* and performing the 1-NN search of each item in the database. We use *DiSAT* because in [7] it shown that is one of the most competive index at searches. It is important to notice that there is the cheaper way to obtain the exact 1-NN of all the elements, avoiding the brute force approach.

As we mention previously, if we are willing to spend some distance calculations more, we can iterate the process in order to improve the answer. 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. The total cost to obtain the *All*-1-NN$_A$ of our proposal is the number of distance evaluations done during all index constructions. The total cost of the *All*-1-NN considers all distance evaluations performed for construction and searching. All our results are averaged over 10 executions of the processes performed on different permutations of the datasets.

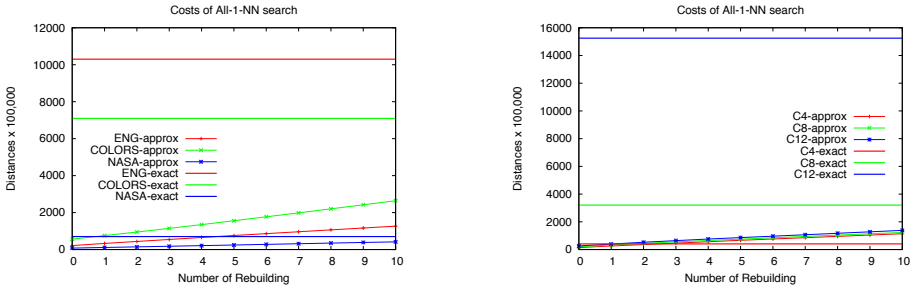
For the experiments, we consider a set of real-life metric spaces with widely different histograms of distances available from `www.sisap.org` [11]:
Strings: a dictionary of 69,069 English words. The distance is the *edit distance*, that is, the minimum number of character insertions, deletions and substitutions needed to make two strings equal.
NASA images: a set of 40,700 20-dimensional feature vectors, generated from images downloaded from NASA [2]. The Euclidean distance is used.
Color histograms: a set of 112,682 8-D color histograms (112-dimensional vectors) from an image database [3]. Any quadratic form can be used as a distance, so we chose Euclidean distance.

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

The 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 *All*-1-NN and STRINGS-approx the cost of our *All*-1-NN$_A$ solution,

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

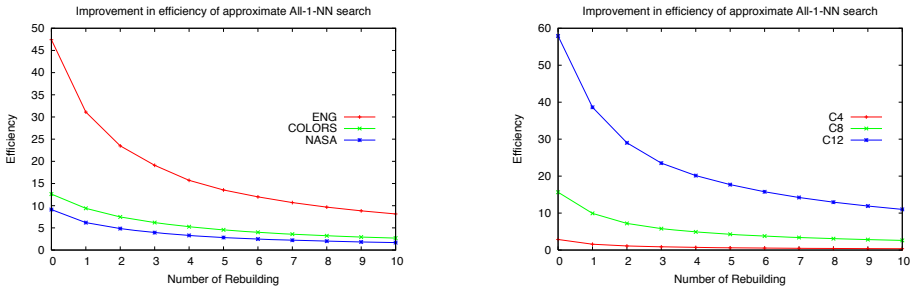
for the space of Strings. Besides, as it can be seen, we use the same color for both costs on the same metric space. Alike, Figure 1(b) depicts the same experiments on the three synthetic spaces, designiting the spaces of coordinate vectors in dimensions 4, 8, and 12 as C4, C8, and C12, respectively.



(a) Real spaces. (b) Synthetic spaces.

Fig. 1. Comparison of costs of exact and approximate *All*-1-NN for all metric spaces considered.

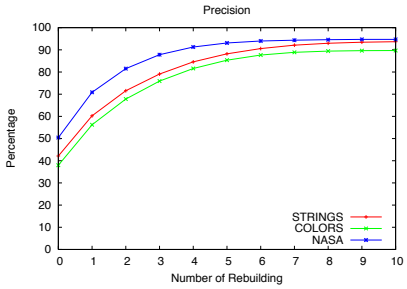
As it can be noticed, our proposal is significantly less expensive to perform in almost all the metric spaces used than the exact solution. Only, in the space of vectors in dimension 4, the exact alternative surpasses, although not significantly, our solution from the second rebuilding onwards. In order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



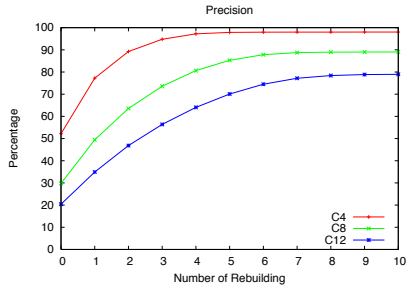
(a) Real spaces. (b) Synthetic spaces.

Fig. 2. Improvement in efficiency of *All*-1-NN$_A$, for all metric spaces considered.

In Figure 3 we show the precision of the response obtained with each reconstruction. After fourth reconstruction, it 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 answer precision, but as it can be seen at Figure 1 we could spend many more reconstructions and even so to obtain much lower costs to response adequatly to *All*-1-NN$_A$; that is to achieve a high quality solution at a low cost in distance evaluations.



(a) Real spaces.



(b) Synthetic spaces.

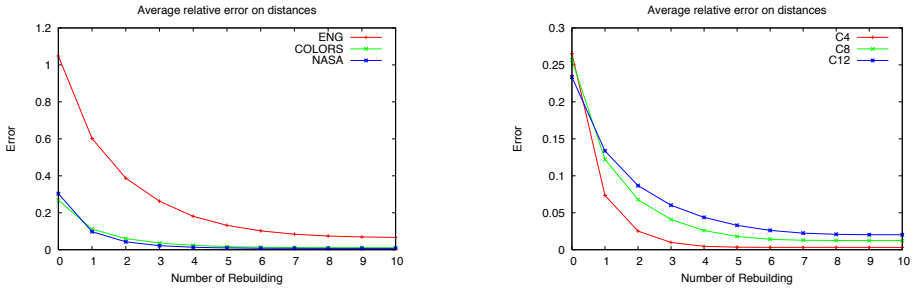
Fig. 3. Precision of the answer of *All*-1-NN$_A$, for all metric spaces considered.

We also evaluate the quality of the approximate solution by measuring the average relative error on distances. Figure 4 exhibits the error obtained versus the number of rebuilding, for the two kinds of metric spaces used. Over the real metric spaces, Figure 4(a) depicts that the error decreases fast, and as soon we rebuild the tree four times the error is almost zero, but for the Dictionary possibly because it uses a discrete distance. On the other hand, Figure 4(b) exposes the error for the three synthetic spaces used. 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.

6 Conclusions

In this paper we tested an alternate approach to computing an approximation to the *All*-1-NN using a simple heuristic. We have designed an algorithm able to retrieves the *All*-1-NN$_A$ 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 solve the *All*-1-NN$_A$, 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 calculated 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 dimensionalities.



(a) Real spaces. (b) Synthetic spaces.

Fig. 4. Average relative error on distances of the answer of *All*-1-NN$_A$, for all metric spaces considered.

References

1. N. Archip, R. Rohling, P. Cooperberg, H. Tahmasebpour, and S. K. Warfield. Spectral clustering algorithms for ultrasound image segmentation. volume 3750, pages 862–869, 2005.
2. Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998.
3. R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query clustering for boosting web page ranking. pages 164–175, 2004.
4. 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.
5. 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.
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. P. Ciaccia and M. Patella. Approximate and probabilistic methods. *SIGSPATIAL Special*, 2(2):16–19, 2010.
9. R. Duda and P. Hart. Pattern classification and scene analysis. *John Wiley & Sons*, 1973.
10. D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal poly-topes. volume 11, pages 321–350, 1994.
11. Karina Figueroa, Gonzalo Navarro, and Edgar Chávez. Metric spaces library, 2007. Available at `http://www.sisap.org/Metric_Space_Library.html`.
12. G. Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 11(1):28–46, 2002.
13. G. Navarro and N. Reyes. Dynamic spatial approximation trees. *Journal of Experimental Algorithmics*, 12:1–68, 2008.
14. R. Paredes. *Graphs for Metric Space Searching*. PhD thesis, University of Chile, Chile, July 2008.
15. 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.
16. M. Patella and P. Ciaccia. Approximate similarity search: A multifaceted problem. *J. Discrete Algorithms*, 7(1):36–48, 2009.