ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/225726826

Navigating K-Nearest Neighbor Graphs to Solve Nearest Neighbor Searches

Chapter · December 2010

DOI: 10.1007/978-3-642-15992-3_29

CITATION
1

READS
68

2 authors:

Edgar Chávez
Ensenada Center for Scientific Research an…
110 PUBLICATIONS **2,704** CITATIONS
SEE PROFILE

Eric S. Tellez
Consejo Nacional de Ciencia y Tecnología
27 PUBLICATIONS **65** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

RedICA: Red temática CONACYT en Inteligencia Computacional Aplicada View project

All in-text references underlined in blue are linked to publications on ResearchGate, letting you access and read them immediately.

Available from: Eric S. Tellez
Retrieved on: 18 October 2016

Navigating *K*-Nearest Neighbor Graphs to Solve Nearest Neighbor Searches

Edgar Chávez[1,2] and Eric Sadit Tellez[1]

[1] Universidad Michoacana. México.
[2] CICESE. México.

Abstract. Nearest neighbor queries can be satisfied, in principle, with a greedy algorithm under a proximity graph. Each object in the database is represented by a node, and proximal nodes in this graph will share an edge. To find the nearest neighbor the idea is quite simple, we start in a random node and get iteratively closer to the nearest neighbor following only adjacent edges in the proximity graph. Every reachable node from current vertex is reviewed, and only the closer-to-the-query node is expanded in the next round. The algorithm stops when none of the neighbors of the current node is closer to the query. The number of revised objects will be proportional to the diameter of the graph times the average degree of the nodes. Unfortunately the degree of a proximity graph is unbounded for a general metric space [1], and hence the number of inspected objects can be linear on the size of the database, which is the same as no indexing at all.
In this paper we introduce a *quasi*-proximity graph induced by the all-k-nearest neighbor graph. The degree of the above graph is bounded but we will face local minima when running the above greedy algorithm, which boils down to have false positives in the queries.
We show experimental results for high dimensional spaces. We report a recall greater than 90% for most configurations, which is very good for many proximity searching applications, reviewing just a tiny portion of the database.
The space requirement for the index is linear on the database size, and the construction time is quadratic in worst case. Relaxations of our method are sketched to obtain practical subquadratic implementations.

1 Introduction and Related Work

Nearest neighbor search is a fundamental problem with a very large number of applications, ranging from pattern recognition, knowledge discovery and probability density estimation to multimedia information retrieval.

A nearest neighbor search (NNS) consist in finding the closest object to a given query among a dataset equipped with a distance function. The problem can be easily solved by a sequential scan of the dataset, but in a number of situations this solution is not acceptable; for example when comparing objects is expensive and/or when the dataset is very large. To avoid the sequential scan the dataset must be preprocessed. A data structure obtained from this preprocessing is usually called an index. The simpler

(and older) applications are related to spatial queries, in this particular case it is possible to obtain logarithmic complexity guarantees. Multidimensional spatial queries still have logarithmic guarantees, but the complexity is exponential on the dimension of the space, hence above certain dimension a sequential scan is faster than using an index. This is called the *curse of dimensionality* and one of the motivations of stating the problem in a coordinate-free model is to avoid such exponential dependence on the dimension. Unfortunately, this problem also appear in the coordinate-free model, being the only resource to maintain scalable indexes the use of approximate and probabilistic methods.

The most basic model of complexity for this problem is just to count the number of distance computations performed to answer a query. This model assume all other operations have a time complexity that could be neglected. To have this in perspective think in the cost of computing the distance between two fingerprints, two faces or two complex objects in general. The time to compute the distance in this examples is in the order of seconds, while the time to traverse a data structure could be orders of magnitude smaller. We will stick to this model for its simplicity and because we are aiming at answering a basic, theoretical question instead of an application.

1.1 The Metric Proximity Searching Model

Formally the database of objects is a finite sample $\mathbb{S}$ of a (possibly) infinite set $\mathbb{X}$ and the objects are comparable only with a distance $d(\cdot,\cdot) : \mathbb{X} \times \mathbb{X} \to \mathbb{R}^+$. The pair $(\mathbb{X}, d)$ is called a *metric space*. The distance is usually assumed to obey the triangle inequality, be symmetric, and reflexive. A nearest neighbor query is defined as $NN(q) = \arg\min_{x \in \mathbb{S}} d(x, q)$, and can be solved by successive applications of *range query* $(q, r)_d = \{x \in \mathbb{S} | d(q, x) \leq r\}$ in a hierarchical index with the algorithm described in [2].

Searching for the nearest neighbor in either a multidimensional space (a.k.a. *vector space*) or a metric space with any indexing method may turn into a sequential scan of the database. This happens when the *intrinsic dimensionality* of the space is very high (about 20 dimensions for the unitary cube with uniformly distributed points) and for the coordinate-free or metric space model, when the histogram of distances is very sharp and is concentrated far from the origin. The NNS problem, the curse of dimensionality, a large number of indexes, and the problems found for *exact* nearest neighbor searching are reported in several places and surveyed in [3–5].

The practical impossibility of building a sublinear algorithm for NNS, and in general for exact proximity searching, has led to develop *inexact proximity searching algorithms* as surveyed in [6]. Here the algorithm returns for example ε-nearest neighbors with some closeness guarantees or just return a giving percentage of the correct nearest neighbors. Inexact indexes are faster at the expense of loosing some answers. In one form or another exact proximity searching is limited to low dimensional metric spaces. Being the focus of this paper we are interested in approximate and probabilistic algorithms.

1.2 Proximity Graphs

A *proximity graph* is a graph with the *local-imply-global* property for proximity searching. Formally $G = (V, E)$ with $V = \mathbb{S}$ the set of vertices and E the edges of the graph, G_κ will be a proximity graph if for any $x \in \mathbb{X}$ and any $v \in V$ it holds $v = NN(x)$ or

there is an edge $(v, w) \in E$ such that $d(x, w) < d(x, v)$.[3] Informally, the property is: at any vertex of the graph, and for any query point $x \in \mathbb{X}$ the current node v is the nearest neighbor of x or one of the edges adjacent to v is closer to x than v.

A proximity graph defines a greedy algorithm for NNS. To find the nearest neighbor the idea is quite simple, we start in a random node and get iteratively closer to the nearest neighbor following only adjacent edges in the proximity graph, all nodes reachable from the current object are inspected and only the closer-to-the-query node is expanded in the next round. The algorithm stops when none of the neighbors of the current node is closer to the query. The number of revised objects will be proportional to the diameter of the graph times the average degree of the nodes. Please notice that extra edges in the proximity graph affect only the number of revised objects, but deliver a correct algorithm.

An example of such a proximity graph in multidimensional (vector) spaces is the Delaunay graph, the dual of the Voronoi graph surveyed in [7]. The quest for a proximity graph in a general metric space was deterred when it was proved in [1] that the degree of the graph is unbounded if only the distance information is used. The proof is essentially the following observation, for any given pair of nodes $u, v \in \mathbb{S}$ one can build a query $q \in \mathbb{X}$, not violating the triangle inequality, which needs the edge $(u, v) \notin E$ in the greedy search for the nearest neighbor; hence the graph degree is unbounded. If the degree of the nodes is unbounded then each step in the greedy algorithm will take $O(n)$ distance computations, rendering the technique useless: the curse of dimensionality again in another facet.

1.3 Approximate Greedy Searching

The inability to build a proximity graph with bounded degree in a general metric space is due to the lack of knowledge about the probable position of the query with respect to the database elements. This restrictions can be fixed for vector spaces, the Delaunay graph is a constructive proof of existence; but we are not aware of the existence of algorithmic extensions for general metric spaces.

We need a definition of the *All Nearest Neighbor Graph*. First, we need to define the KNN set.

Definition 1. *The K Nearest Neighbors search of q, $KNN(q)$ for short, comes as a natural extension to $NN(q)$ search: Sort the entire database by increasing distance to q, the first K objects in the rank are the $KNN(q)$ set.*

Definition 2. *The AKNN graph is defined as $AKNN(V, E)$ where $V = \mathbb{S}$ and $E = \{(u, v) \mid v \in KNN(u)\}$, there will be an edge from every object towards each one of its k-nearest neighbors.*

This graph has been used for approximate proximity searching in at least two contexts described next. The AKNN has been used for proximity searching in [8], where authors propose the use of the graph as a distance bounding device. Each edge in the AKNN is weighted by the distance between the objects. The graph distance between two objects u and v in $\mathbb{S}$ is defined as the sum of the weights of the path joining u and v, if there is no such path the graph distance is infinite. In [8] the graph distance is

[3] The case of ties is special, if two objects are at the same minimal distance to the query, either one of them is a nearest neighbor

used to estimate both an upper and a lower bound for the actual distance to the query using the triangle inequality. The AKNN graph is connected with high probability if k is *large enough* and the objects are distributed with a bounded away from zero distribution as discussed in [9]. The algorithm is for exact proximity searching for both nearest neighbor and range queries. The performance of the algorithm is comparable to the baseline, pivot based algorithm AESA [10], using only a fraction of the memory requirements. Nevertheless the index is useful only for low dimensional data.

Other approach, closer to ours, was presented in [11] where the authors propose an interesting alternative which consists essentially in applying the greedy algorithm in a *non* proximity graph. They propose the use of an extended version of the AKNN graph (or just AKNN were the context allow us) as an approximation to the proximity graph. The extension consists in dynamically adding edges to the AKNN for nodes τ-closer to the query q, where a node p_i is τ-closer to q with respect to p if $d(q,p) \leq \tau d(q,p_i)$ and p_i is in the connected component of p. The degree of node p depends on the value of τ, then for large values produces larger degrees. They report perfect recall for shape queries examining around 25% of the database. Even if this percentage is better than the baseline AESA, it is still not scalable. Perfect recall cannot always be obtained, for example when the nearest neighbor is in a different connected component of the graph; and they use different starting points, named *seeds* in the paper, to increase the probability of not getting stuck in a local minimum.

2 Our Contribution

From the previous work described above in [8, 11] we learned that AKNN graph is useful for proximity searching, both as a distance bounding device and as an approximation to a proximity graph, since it acceps greedy nearest neighbor searching. In this paper we investigate the *proximity power* of a graph induced by the AKNN. Instead of extending a candidate graph adding nodes as in [8, 11] (which increase the degree of the nodes), we propose to find a subgraph of the AKNN with the proximity property, even if the property is valid only in probability.

We begin by observing that the AKNN graph is not necessarily symmetric. If u is the k nearest neighbor of a node v, v is not necessarily the k nearest neighbor of u. There are two ways to make the AKNN graph symmetric. If a node is at the same time a nearest neighbor and a reverse nearest neighbor the two nodes realizing the relation are said to be *mutual* (k) nearest neighbors, $u \in MKNN(v) \leftrightarrow u \in KNN(v)$ and $v \in KNN(u)$, this procedure delete an edge if the symmetric does not exist. The other alternative is to symmetrize adding edges. We can also define the *symmetric* (k) nearest neighbors as $u \in SKNN(v) \leftrightarrow u \in KNN(v)$ or $v \in KNN(u)$.

2.1 Approximate Proximity Hypothesis

In the greedy setup we use adjacent nodes as routing devices for queries. Proximal nodes are certainly useful for routing and should be adjacent in the graph. We additionally postulate that two nodes sharing a certain amount (κ) of their k-nearest neighbors should be also adjacent in the proximity graph.

Definition 3. *Define* $G_\kappa(\mathbb{S}, E)$ *the proximal transitive graph of strength* κ. *With* $(u,v) \in E \leftrightarrow |KNN(u) \cap KNN(v)| \geq \kappa$.

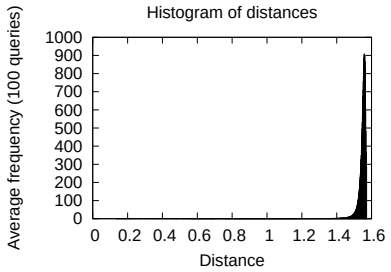
We can also define a similar graph using $SKNN$ as the building block. We call this the *symmetric* version in the experimental results section, below.

Definition 4. *Define* $\hat{G}_\kappa(\mathbb{S}, E)$ *the* symmetric *proximal transitive graph of strength* κ. *With* $(u, v) \in E \leftrightarrow |SKNN(u) \cap SKNN(v)| \geq \kappa$.

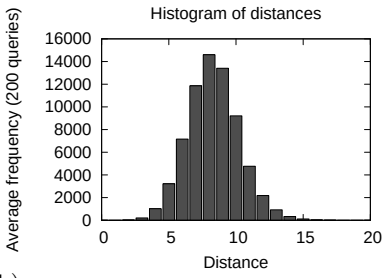
We will work on the basis of the following hypothesis: *The connected components of* G_κ *and* $\hat{G}_\kappa$ *are quasi-proximity graphs.*

There is a potentially large number of connected components in a giving graph. Since we do not know in advance in what component we can find the query we need to search in each connected component. In the next section we report on the time complexity of searching in all the connected components.

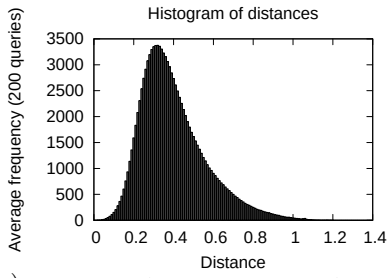
3 Experimental Results



(a) News documents. TFIDF vectors. More than 237000 coordinates 25057 objects.



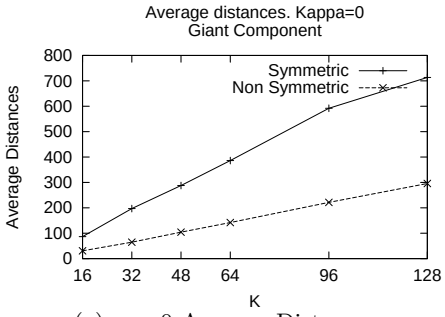
(b) English dictionary. Edit distance. 200 queries. 69069 entries.



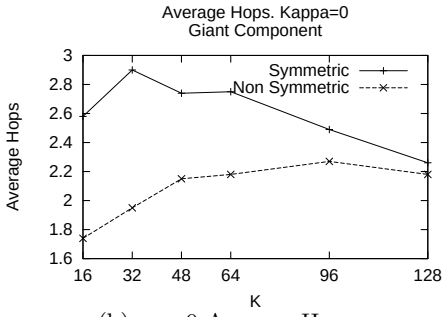
(c) Histogram of colors as vectors of dimension 112. Euclidean distance. 200 queries. 112682 objects.

Fig. 1. Histograms of different databases. The news articles holds clearly the higher intrinsic dimensionality (large mean, small variance).

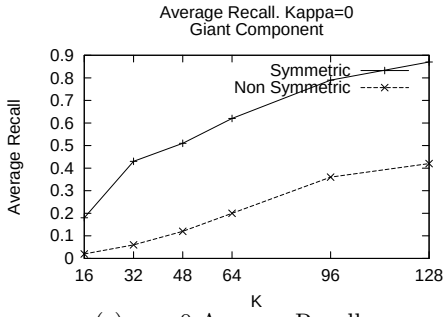
We used a collection of 25057 TFIDF (term frequency $\times$ inverse document frequency) vectors using the angle between vectors as the metric. We used the data sets



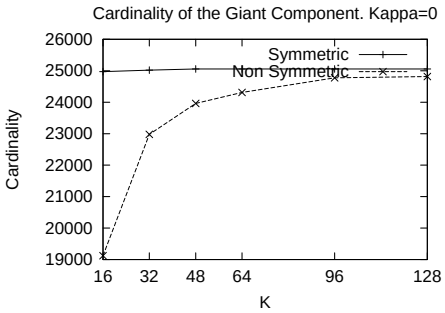
(a) $\kappa = 0$ Average Distances



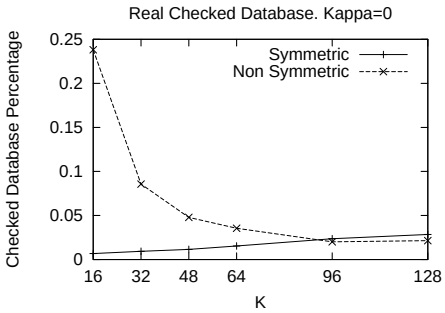
(b) $\kappa = 0$ Average Hops



(c) $\kappa = 0$ Average Recall



(d) $\kappa = 0$ Giant Component Cardinality

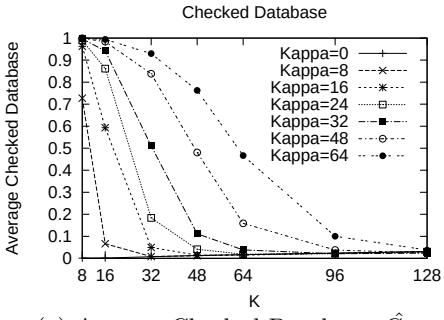


(e) $\kappa = 0$ Average Checked Database

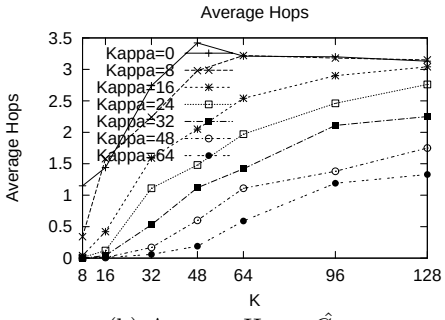
Fig. 2. General behavior for G_0 and $\hat{G}_0$ graphs.

of the SISAP metric space library[4], which is a standard set of databases and objects used to compare proximity searching algorithms. The TFIDF vectors are commonly described using several thousands of coordinates. Every vector is sparse and is represented using only with a few hundred of coordinates, e.g. the average number of effective coordinates in our query set is of 360 while needs more than 237000 explicit dimensions, which is really large for standard proximity searching algorithms. The in-

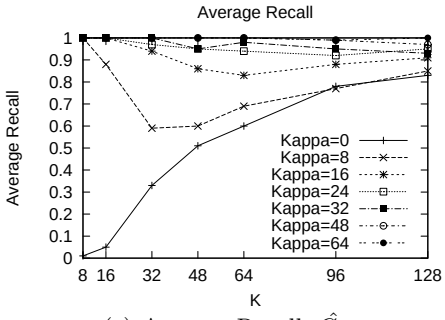
[4] The homepage of the SISAP project is `http://www.sisap.org`



(a) Average Checked Database. $\hat{G}_\kappa$



(b) Average Hops. $\hat{G}_\kappa$



(c) Average Recall. $\hat{G}_\kappa$

Fig. 3. Searching in queries in the connected components of $\hat{G}_\kappa$ with $\kappa \geq 0$.

trinsic dimensionality is also very large, as depicted in Figure 1. We can see in Figure 1(a) that the set of distances is concentrated around a large value with a small variance, according to [3] this characteristic is an indication of a large intrinsic dimension. Similarly Figures 1(c) and 1(b) show the histogram of distances for vector images and an english dictionary, respectively.

Please notice that the news articles have the higher intrinsic dimension, as they have the larger mean and the smaller variance. We will focus our attention in this dataset. We have omitted the experiments for the other datasets for the space constraints of the conference format. We believe solving the problem for the hardest problem is enough evidence of the strong points of the technique. We also omitted investigating the use of $MKNN$ as the building block of the proximal transitive graph. As a quick comment, an exact index such as the Burkhard-Keller Tree (BKT) [12, 3] searching the nearest neighbor needs to check more than 22% for the english dictionary 1(b) for words not in the dictionary and 9.8% for the image database with random queries selected from the same database 1(c), and 98.8% for the short news articles 1(a) (with random queries selected from the database).

A set of six indexes for $K = 16, 32, 48, 64, 96$ and 128 were built for G_κ and $\hat{G}_\kappa$. We repeated the nearest neighbor queries 300 times per index reporting the average values of the results as follows:

Recall. How many times we found the real NN, as a ratio of the 300 searches.

Hops. The path inside a single connected component, measured in distances, to reach the NN.

Checked ratio. The average number of distance computations as a ratio over the total size of the database.

The special case of $\kappa = 0$ was investigated first. For the curves of $\hat{G}_\kappa$ the first observation was that a giant connected component was obtained for $\kappa = 0$, covering most of the graph. This makes sense because this is the least restrictive condition. Figure 2 shows the general behavior of the giant component and the index with $\kappa = 0$. The number of hops is very small, and experimentally is independent of the database size (see Figure 2(b)). The recall is very good for the giant component in the symmetric graph as depicted in 2(c). The cardinality is really close to the entire database and the leftovers can be checked sequentially or by using other indexes, this can be observed in Figure 2(d).

A notable property of the method is that for $K \geq 48$, in G_κ, we check less than 5% of the entire database even if we add the cost of the linear scan outside of the giant component. The $\hat{G}_\kappa$ needs to check an even smaller ratio of the database as shown by Figure 2(e). Due to the above fact, in the rest of the section we restrict the experiments to $\hat{G}_\kappa$ since it is superior to G_κ experimentally.

3.1 Larger κ

Figure 3 shows the result for $\kappa > 0$. Large κ values creates a large number of small CC, these behavior can be seen in 3(a) where the entire database should be checked for small K and κ. In fact, an interesting relation can be found in $\frac{\kappa}{K} \approx 1$ because it defines an inflection point affecting the performance and recall. Since the graph nodes have higher degree, this inflection point has a smaller impact than in G_κ. In general $\frac{\kappa}{K} < 1$ gives better performance, see Figure 3(a).

The recall, Figure 3(c), it is affected inversely proportional to the performance because we are avoiding the sequential scan of the database. In the other hand, the number of hops is still small and can be considered $O(1)$ as in the giant component of the previous experiment. Surprisingly, we have setups needing less than 1 hop in average to find the result. This value puts in evidence the closeness in every connected component, i.e. smaller components with strongest notions of proximity. The cost is delegated to find the correct CC which is a minimum cost for the symmetric graph, see Figure 3(b).

We have found some interesting values for some configurations, for example for $K = 62$ (i.e. 31 if we represent symmetric graphs with undirected edges) we found that for $\kappa = 24$ it fails 10 times (10 in 300 queries). The reported nearest neighbors are at a distance of 1.28. In Figure 1(a) we can see that the mean is located 1.55 and the mass is concentrated around 1.5 and 1.6. This means that 1.28 is a very (relatively) close object, unfortunately remains inexact. Another important value is the degree, which is 31. This means that we can really control the necessary space for the index using K and κ.

4 Lightweighting the Preprocessing Step

As we see the indexing process takes $O(n^2)$ comparisons to compute the $AKNN$ and hence computing either G_κ or $\hat{G}_\kappa$. Using an index to obtain AKNN is not really an

option, since exact indexes will degrade to sequential search in high dimensions. The space needed to store AKNN, G_κ or $\hat{G}_\kappa$ is $O(Kn)$ and it can be handled easily even for large datasets.

Due to the high cost of the construction, the index can be used in medium or large databases, this is specially a concern for practical implementations.

We can speed up the preprocessing stage using a small modification to the algorithm. Instead of using $\hat{G}$ we can compute a graph with similar properties as follows:

- Randomly select $Y \subset S$.
- Let $m = |Y|$, $m \ll n$ in order to consider it $o(1)$ with respect to n, but large enough to accomplish the previous requirement.
- We construct in at most mn comparisons a graph of all K nearest neighbors of $u \in S$ in Y, defining $KNN_Y(u)$, and $SKNN_Y(u)$ to represent the same operations working over the restricted Y.
- In the same way, we define $\hat{G}_Y(\mathbb{S}, \kappa)$, as in definition 4, but using $SKNN_Y(\cdot)$ instead of $SKNN(\cdot)$.
- Create an inverted list I [13], from $Y \to S$, as follows: Let $u \in S$, $v \in Y$ if $v = NN(u, Y)$ then we add u to the v entry in I. This can be created in n steps with no additional distance comparisons, since we already compute the graph.
- Finally, every NN query is solved using $\hat{G}_Y$ and the greedy search algorithm. At the end, we will be placed in the neighborhood of NN at some node $w \in S$. So, we must find in I the place for w and filter the list to get the real result.

Each posting list w inside I can be sorted in increasing distance to w, allowing to prune using the triangle inequality. Another enhancement to this linear algorithm is to increase the number of verification lists on I of the $KNN_Y(w)$ instead of just w, this should increase the possibility of find the right result.

The K parameter should be smaller than the complete algorithm because our graph holds an smaller diameter. Clearly, the above algorithm only works for symmetric graphs.

5 Conclusions and Future Work

In this paper we introduced a new approximation to a proximity graph using the notion of shared (symmetric, mutual)k-nearest neighbors. The defined graph $\hat{G}$ is divided in connected components, and the query is searched greedily in each connected component. Our algorithm solves effectively and efficiently the nearest neighbor problem using an approximate approach with high recall and checking a very small fraction of the database.

Additionally, we sketch a linear preprocessing time algorithm allowing to create a practical implementation of the method. We are currently investigating another possible enhancements using a hierarchical structure defined applying recursively different κ values.

One of the main problems is the avoidance of paths leading to local minimums, or be capable to know when a search process is stalled. In other words, converge to an exact algorithm. Many standard techniques can be used to search outside local minima.

We should notice that searching in many connected components leads to a natural parallelization technique, one thread for each connected component, this an interesting optimization exploiting capabilities of new hardware as powerful GPU's, and special networking schemes like clouds or grids.

An interesting alternative for effectively indexing large databases is to consider a mixture of the distance bounds obtained in [8] with the current work. It is not hard to see a more clever way to navigate the collection of connected components by estimating the distances between them and obtaining distance bounds to prune some candidate fractions of the database. We see many potential applications to the decomposition technique into small clusters. Our current attention is the practical implementation of the method, linear time construction keeping the recall and time characteristics of the approach, presented in this paper.

References

1. Navarro, G.: Searching in metric spaces by spatial approximation. The VLDB Journal **11**(1) (2002) 28–46
2. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers (2006)
3. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. **33**(3) (2001) 273–321
4. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces (survey article). ACM Trans. Database Syst. **28**(4) (2003) 517–580
5. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity search: The metric space approach. Springer-Verlag New York Inc (2006)
6. Patella, M., Ciaccia, P.: Approximate similarity search: A multi-faceted problem. Journal of Discrete Algorithms **7**(1) (2009) 36–48
7. Aurenhammer, F.: Voronoi diagrams—a survey of a fundamental geometric data structure. ACM Computing Surveys (CSUR) **23**(3) (1991) 405
8. Paredes, R., Chávez, E.: Using the k-nearest neighbor graph for proximity searching in metric spaces. Lecture Notes in Computer Science **3772** (2005) 127–138
9. Brito, M., Chavez, E., Quiroz, A., Yukich, J.: Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection. Statistics & Probability Letters **35**(1) (1997) 33–42
10. Vidal, E.: New formulation and improvements of the Nearest-Neighbour approximating and eliminating search algorithm(AESA). Pattern Recognition Letters **15**(1) (1994) 1–7
11. Sebastian, T., Kimia, B.: Metric-based shape retrieval in large databases. In: Pattern Recognition, 2002. Proceedings. 16th International Conference on. Volume 3. (2002)
12. Burkhard, W.A., Keller, R.M.: Some approaches to best-match file searching. Communications of the ACM **16**(4) (1973) 230–237
13. Baeza-Yates, R.A., Ribeiro-Neto, B.A.: Modern Information Retrieval. ACM Press / Addison-Wesley (1999)