Information Systems 45 (2014) 61–68

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Approximate nearest neighbor algorithm based on navigable small world graphs

Yury Malkov [a,b,*], Alexander Ponomarenko [b,c], Andrey Logvinov [b], Vladimir Krylov [b,d]

[a] The Institute of Applied Physics of the Russian Academy of Sciences, 46 Ul'yanov Street, 603950 Nizhny Novgorod, Russia
[b] MERA Labs LLC, 13, Delovaya St., Nizhny Novgorod 603163, Russia
[c] National Research University Higher School of Economics, Laboratory of Algorithms and Technologies for Network Analysis, 136 Rodionova, Nizhny Novgorod 603093, Russia
[d] Nizhny Novgorod State Technical University, Nizhny Novgorod, Russia

ARTICLE INFO

Available online 4 November 2013

Keywords:
Similarity search
k-Nearest neighbor
Approximate nearest neighbor
Navigable small world
Distributed data structure

ABSTRACT

We propose a novel approach to solving the approximate *k*-nearest neighbor search problem in metric spaces. The search structure is based on a navigable small world graph with vertices corresponding to the stored elements, edges to links between them, and a variation of greedy algorithm for searching. The navigable small world is created simply by keeping old Delaunay graph approximation links produced at the start of construction. The approach is very universal, defined in terms of arbitrary metric spaces and at the same time it is very simple. The algorithm handles insertions in the same way as queries: by finding approximate neighbors for the inserted element and connecting it to them. Both search and insertion can be done in parallel requiring only local information from the structure. The structure can be made distributed. The accuracy of the probabilistic k-nearest neighbor queries can be adjusted without rebuilding the structure.

The performed simulation for data in the Euclidean spaces shows that the structure built using the proposed algorithm has small world navigation properties with $\log^2(n)$ insertion and search complexity at fixed accuracy, and performs well at high dimensionality. Simulation on a CoPHiR dataset revealed its high efficiency in case of large datasets (more than an order of magnitude less metric computations at fixed recall) compared to permutation indexes. Only 0.03% of the 10 million 208-dimensional vector dataset is needed to be evaluated to achieve 0.999 recall (virtually exact search). For recall 0.93 processing speed 2800 queries/s can be achieved on a dual Intel X5675 Xenon server node with Java implementation.

1. Introduction

The scalability of any software system is limited by the scalability of its data structures. Massively distributed systems like BitTorrent or Skype are based on distributed hash tables. While the latter structures have good scalability, their search functionality is limited to the exact matching. This limitation arises because small changes in an element value lead to large and chaotic changes in the hash value, making the hash-based approach inapplicable to the range search and the similarity search problems.

However, there are many applications (such as pattern recognition and classification [1], content-based image retrieval [2], machine learning [3], recommendation systems [4], searching similar DNA sequence [5], semantic

* Corresponding author at: The Institute of Applied Physics of the Russian Academy of Sciences, 46 Ul'yanov Street, 603950 Nizhny Novgorod, Russia.
E-mail address: yurymalkov@mail.ru (Y. Malkov).

0306-4379/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2013.10.006

document retrieval [6]) that require the similarity search rather than just exact matching. The k-nearest neighbor search (k-NNS) problem is a mathematical formalization for similarity search. It is defined as follows: we need to find the set of k closest objects $P \subseteq X$ from a finite set of objects $X \subseteq \mathcal{D}$ to a given query $q \in \mathcal{D}$, where $\mathcal{D}$ is the set of all possible objects (the data domain). Closeness or proximity of two objects $o', o'' \in \mathcal{D}$ is defined as a distance function $\delta(o', o'')$.

A naïve solution for the k-NNS problem is to calculate the distance function δ between q and every element from X. This leads to linear search time complexity, which is much worse than the scalability of structures for exact match search, and makes the naïve version of k-NNS almost impossible to use for large size datasets.

We suggest a solution for the nearest neighbor search problem: a data structure represented by a graph $G(V, E)$, where every object o_i from X is uniquely associated with a vertex v_i from V. Searching for the closest elements to the query q from the data set X takes the form of searching for a vertices in the graph G.

This gives an opportunity for building decentralized similarity search oriented storage systems where physical data location does not depend on the content because every data object can be placed on an arbitrary physical machine and can be connected with others by links like in p2p systems.

One of the basic vertex search algorithms in graphs with metric objects is the greedy search algorithm. It has a simple implementation and can be initiated from any vertex. In order for the algorithm to work correctly (always return precise results), the network must contain the Delaunay graph as its subgraph, which is dual to the Voronoi tessellation [7]. However, there are major drawbacks associated with the Delaunay graph: it requires some knowledge of metric space internal structure [8] and it suffers from the curse of dimensionality [7]. Moreover, for the applications described above, the precise exactness of the search is not required. So the problem of finding the exact nearest neighbors can be substituted by the approximate nearest neighbor search, and thus we do not need to support the whole/exact Delaunay graph.

Graphs with logarithmic scalability of the greedy search algorithm are called navigable small world graphs, they are well known in Euclidean spaces [9]. Note that the small world models (not *navigable* small world) like [10] do not have this feature. Even though there are short paths in the graph, the greedy algorithm do no tend to find them, in the end having a power law search complexity. Solutions for constructing a navigational small world graphs were proposed for general spaces but they are usually more complex, requiring sampling, iterations, rewiring etc. [11–14]. We show that the small world navigation property can be achieved with a much simpler technique even without prior knowledge of internal structure of a metric space (e.g. dimensionality or data density distribution).

In this paper we present a simple algorithm for the data structure construction based on a navigable small world network topology with a graph $G(V, E)$, which uses the greedy search algorithm for the approximate k-nearest neighbor search problem. The graph $G(V, E)$ contains an approximation of the Delaunay graph and has long-range links together with the small-world navigation property. The search algorithm we propose has the ability to choose the accuracy of search without modification of the structure. Presented algorithms do not use the coordinate representation and do not presume the properties of Euclidean spaces, because they are based only on comparing distances between the objects and the query, and therefore in principle are applicable to data from general metric (or even non-metric) spaces. Simulations revealed weak dimensionality dependence for Euclidean data.

2. Related work

Kd-tree [15] and quadra trees [16] were among the first works on the kNN problem. They perform well in 2–3 dimensions (search complexity is close to $O(\log n)$ in practice), but the analysis of the worst case for these structures [17] indicates $O(d^* N^{1-1/d})$ search complexity, where d is the dimensionality.

In Ref. [8] was proposed an exact-proximity search structure that uses the Delaunay graph with the greedy search algorithm. Authors showed the impossibility of finding the exact Delaunay graph in a general metric space, and to keep the search exact they resort to backtracking. Proposed data structure has construction time $O(n \log^2 n / \log\log n)$ and search time $O(n^{1-\Theta(1/\log\log n)})$ in high dimensions and $O(n^{\alpha})$, $(0 < \alpha < 1)$in low dimensions.

In general, currently there are no methods for effective exact NNS in high-dimensionality metric spaces. The reason behind this lies in the "curse" of dimensionality [18]. To avoid the curse of dimensionality while retaining the logarithmic cost on the number of elements, it was proposed to reduce the requirements for the kNN problem solution, making it approximate (Approximate kNN).

There are two commonly used definitions of the approximate neighbor search. One class of methods proposed to search with predefined accuracy ε (ε-NNS). It means that the distance between the query and any element in the result is no more than $1+\varepsilon$ times the distance from query to its true k-th nearest neighbor. Such methods have been described in [19–23]. Another class gives probability guarantee of finding true k closest point to the query [24–31], using "recall" (the fraction of true k nearest elements found).

Some structures [19–23] can be applied only to Euclidean space. Other methods [24–31] are applicable to the general metric space. More can be found in reviews [32,33].

Permutation indexes (PI) [25,34] is an efficient non-distributed algorithm suitable for general metric spaces. The idea behind PI is to represent each database object with the permutation of a set of references, called the permutants, sorted by distance to the object. The distance between objects is hinted by the distance between their respective permutations. PI is known to have high precision and recall even for datasets with high intrinsic dimensionality.

The work by Houle and Sakuma [26] features a probabilistic tree-like structure for the approximate nearest neighbor search in general metric spaces, based on selection of the nearest neighbors. The algorithm was simulated

on real-life data. The work by Chávez and Tellez [27] also uses determination of nearest neighbors in its construction algorithm, with the greedy algorithm used for searching. The main drawback of the algorithm is poor (linear) scalability with the size of the dataset. Both algorithms offer high recall in return for evaluation of only a tiny portion of the dataset.

Kleinberg's work [9] has shown the possibility of using navigable small world networks for finding the nearest neighbor with the greedy search algorithm. The algorithm relied on random long-range links following the power law of link length probability $r^{-\gamma}$, γ for navigation and 2-dimensional lattice for correctness of the results. To have navigable small world properties, the link length distribution has to have a specific value of γ. In Voronet [35], Kleinberg's approach was extended to arbitrary 2-dimensional data by building a two-dimensional Delaunay tessellation instead of a regular lattice. In their next work [13] they have weakened the requirements on the exactness of the search in order to avoid the curse of dimensionality for the d-dimension Euclidean space. The algorithm approximates the Delaunay graph by selecting $3d+1$ neighbors that minimize the volume of the corresponding Voronoi cell. The algorithm relies heavily on the quality of the Delaunay graph approximation, it has to be repeated iteratively to reach acceptable accuracy, and in principle works only in the Euclidean space. The work, together with the others [11–14], also presented some sophisticated algorithms for supporting the Kleinberg's power law link length distribution with a specific exponent value.

3. Core idea

The structure S is constructed as a navigable small world network represented by a graph $G(V,E)$, where objects from the set X are uniquely mapped to vertices from the set V. The set of edges E is determined by the structure construction algorithm. Since each vertex is uniquely mapped to an element from the set X, we will use the terms “vertex”, “element” and “object” interchangeably. We will use the term “friends” for vertices that share an edge. The list of vertices that share a common edge with the vertex v_i is called the friend list of the vertex v_i.

We use a variation of the greedy search algorithm as a base algorithm for the k-NN search. It traverses the graph from an element to another element each time selecting an unvisited friend closest to the query until it reaches a stop condition. See a detailed description of the algorithms in Section 4.2.

It is important to note that links (edges) in the graph serve two distinct purposes:

1) There is a subset of short-range links, which are used as an approximation of the Delaunay graph [7] required by the greedy search algorithm.
2) Another subset is the long-range links, which are used for logarithmic scaling of the greedy search. Long-range links are responsible for the navigation small world properties of the constructed graph [9].

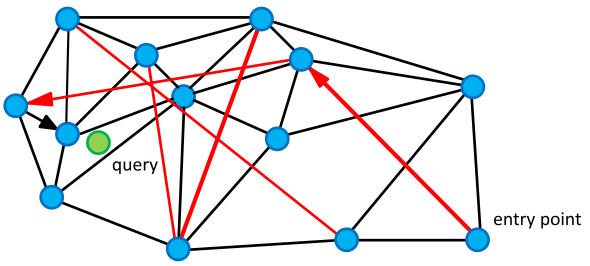

Fig. 1. Graph representation of the structure. Circles (vertices) are the data in metric space, black edges are the approximation of the Delaunay graph, and red edges are long range links for logarithmic scaling. Arrows show a sample path of the greedy algorithm from the entry point to the query (shown green). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The structure performance is illustrated in Fig. 1.

The construction of the structure is based on the consecutive insertion of all elements. For every new incoming element, we find the set of its closest neighbors (Delaunay graph approximation) from the structure. The set is connected to the element and vice versa. As more and more elements are inserted into the structure, links that previously served as short-range links now become long-range links (for details see Section 5) making a navigable small world. All queries in the structure are independent; they can be done in parallel, and if the elements are placed randomly on physical computer nodes, then the processing query load can be shared across physical nodes.

4. Search algorithm

4.1. Basic greedy search algorithm

The basic single nearest neighbor search algorithm traverses the edges of the graph $G(V,E)$ from one vertex to another. The algorithm takes two parameters: query and the vertex $V_{entry\_point} \in V[G]$ which is the starting point of a search (the entry point). Starting from the entry point, the algorithm computes a distance from the query q to each vertex from the friend list of the current vertex, and then selects a vertex with the minimal distance. If the distance between the query and the selected vertex is smaller than the one between the query and the current element, then the algorithm moves to the selected vertex, and it becomes new current vertex. The algorithm stops when it reaches a local minimum: a vertex whose friend list does not contain a vertex that is closer to the query than the vertex itself. The algorithm

```
Greedy_Search(q: object, v_entry_point: object)
1  v_curr←v_entry_point;
2  δ_min←δ(q, v_curr); v_next←NIL;
3  foreach v_friend∈v_curr.getFriends() do
4    δ_fr←d (query, v_friend)
5    if δ_fr<δ_min then
6      δ_min←δ_fr;
7      v_next←v_friend;
8  if v_next=NIL then return v_curr;
9  else return Greedy_Search(q, v_next);
```

The element which is a local minimum with respect to the query $q \in \mathcal{D}$ can be either the true closest element to the

query q from all elements in the set X, or a false closest (an error).

If every element in the structure had in their friend list all of its Voronoi neighbors, then this would preclude the existence of false global minima. Maintaining this condition is equivalent to constructing the Delaunay graph, which is dual to the Voronoi diagram.

It turns out that it is impossible to determine exact Delaunay graph for an unknown metric space [8] (excluding the variant of the complete graph), so we cannot avoid the existence of false global minima. For the problem of approximate search as defined above it is not an obstacle, since approximate search does not require the entire Delaunay graph [13].

Note that there is a distinction from the ANN problem defined in the works [19–23] where it is expressed in terms of ε-neighborhood. Like in [24–31] in our structure there are no constrains on the absolute value of the distance between the algorithm NN results and true NN results. The result guaranties are probabilistic, meaning that only the probability of finding the true nearest neighbor is guaranteed. It may be more convenient to use such definition of the search effectiveness when the data distribution is highly skewed and it is hard to define one value ε for all regions at the same time.

4.2. k-NN search modification

In our previous work [36] we have used a simple algorithm for k-NN search based on a series of `m` searches and returns the best results of these. With each subsequent search, the probability of not finding the nearest neighbors decreases exponentially, allowing boosting the accuracy of the structure without the need for reconstruction.

In this work we present a more sophisticated version of the k-NN algorithm with two key modifications:

1) We use different stop condition. The algorithm iterates on not previously visited elements (i.e. those for which the link list has not been read) closest to the queries. It stops when at the next iteration, `k` closest results to the query do not change. Simply put, the algorithm keeps exploring the neighborhood of the closest elements in a greedy manner as long as it can improve the known `k` closest elements on each step.
2) The list of previously visited elements `visitedSet` is shared across the series of searches preventing useless repeated extractions.

```
K-NNSearch(object q, integer: m, k)
1  TreeSet [object] tempRes, candidates, visitedSet,
   result
2  for (i←0; i<m; i++) do:
3    put random entry point in candidates
4    tempRes←null
5    repeat:
6      get element c closest from candidates
       to q
7      remove c from candidates
8      //check stop condition:
9      if c is further than k-th element from result
10       than break repeat
11     //update list of candidates:
12   for every element e from friends of c do:
13     if e is not in visitedSet than
14       add e to visitedSet, candidates, tempRes
15
16     end repeat
17     //aggregate the results:
18     add objects from tempRes to result
19 end for
20 return best k elements from result
```

The use of `TreeSet` ordered lists allows storing evaluated elements in the order of proximity to the query, thus easily extracting closest elements from the set, which is required on steps 6, 9 and 20.

If `m` is big enough, the algorithm becomes an exhaustive search, assuming that entry points are never reused. If the graph of the network has the small-world property, then it is possible to choose a random vertex without any metric calculations in a number of random steps proportional to the logarithm of the dataset size, which does not yield the overall logarithmic search complexity.

5. Data insertion algorithm

Since we build an approximation of the Delaunay graph, there is a great freedom in the details of the construction algorithm. The main goal is to minimize the probability of false global minima while keeping the number of links as small possible. Some approaches are based on knowledge of topology of the metric space being used. For example, in [13] it is proposed to build an approximate Delaunay graph which would minimize the volume of a Voronoi region (computed by the Monte-Carlo method) for a fixed number of edges for each vertex in the graph (achieved by iterating a selection of neighbors of every node in the graph several times). We propose to assemble the structure by inserting elements one by one and connecting them on each step with the `f` closest objects which are already in the structure. Our approach is based on the idea that intersection of the set of elements which are Voronoi neighbors and the `f` closest elements should be large.

The graph can be constructed by sequential insertion of all elements. For every new coming element, we find the set of its closest neighbors (Delaunay graph approximation) from the structure. The set is connected to the element and vice versa. One of the advantages of this approach (already shown empirically for one-dimensional data [37]) is that a graph created by such algorithm with general metric data arriving in random order has small world navigation properties without any additional arrangements.

To determine the set of `f` closest elements, we use approximate kNN search algorithm (see Section 4.2). The algorithm takes three parameters: the object to be inserted in the structure, and two positive integer numbers: `f` (number of nearest neighbors to connect) and `w` (number of multi-searches).

First, the algorithm determines a set `neighbors` containing `f` local closest elements using the procedure

k-NNSearch (see Section 4.2). After that new_object is connected to every object in a set and vice versa.

```
Nearest_Neighbor_Insert(object: new_object, integer:
  f, integer: w)
1  SET[object]: neighbors←k-NNSearch (new_object, w,
   f);
2  for (I←0; i < f; i++) do
3  neighbors [i].connect(new_object);
4  new_object.connect(neighbors [i]);
```

5.1. Choice of parameters

The parameter w affects how accurate is determination (recall) of nearest neighbors in the construction algorithm [36]. Like in Section 4.2, setting w to a big number is equivalent to exhaustive search of the closest elements in the structure resulting in a perfect recall. The idea is to set w big enough to have the recall close to unity (e.g. 0.95–0.99). Smaller recall will create a fraction of wrong links which solely increase complexity of the algorithm, while our experiments indicate that increasing recall at insertion higher than 0.99 have no measurable effect on the search quality. Test have also shown that w for optimal recall changes slowly (logarithmically) with the dataset size, so if we already know the approximate w0 for a good recall, we can run random query tests, firstly with much larger m (e.g. m=2*w0+10), assuming that m is large enough for the results of the search to be true k nearest neighbors, and then increase w, repeating the tests until we have a high recall (e.g. 0.95–0.99). The complexity of the operation is logarithmic to the size of the dataset so it does not affect the overall construction complexity.

The tests indicate [36] that at least for Euclid data with $d=1...20$, the optimal value for number of neighbors to connect (f) is about $3d$, making memory consumption linear with the dimensionality. Lesser values of f can be used to reduce the complexity of a single search, sacrificing its recall quality.

6. Test results and discussion

6.1. Test data

We have implemented the algorithms presented above in order to validate our assumptions about the scalability of the structure, and to evaluate its performance. In our tests we have used a workstation based on two Intel Xeon X5675 six core processors with 192 Gb of RAM. The algorithm was written in Java using the Oracle Java Platform.

We have used the following test datasets:

- Uniformly distributed random points with L_2 (Euclidean distance) distance function (up to 5×10^7 elements, up to 50 dimensions).
- A subset of the CoPHiR [38] dataset for comparison with other works. 208-dimensional feature vectors were extracted from the database. L_1-metric was used as a distance function. 30 approximate nearest neighbors are found during a search.

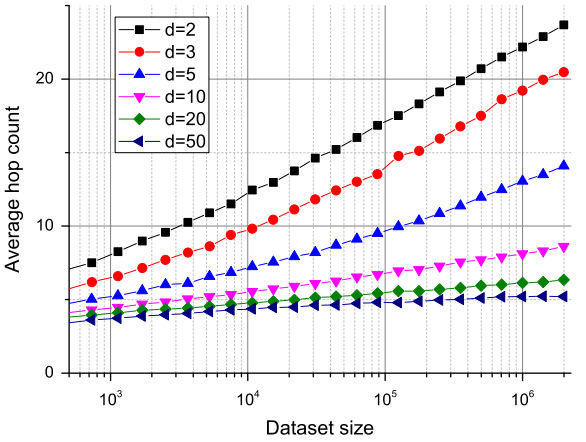

Fig. 2. The average hop count induced by a greedy search algorithm for different dimensionality Euclidean data (k=10, w=20). The navigable small world properties are evident from the logarithmic scaling.

6.2. Small world navigation properties

To verify the small world navigation properties of the proposed structure, we have measured the average path length induced by the greedy search algorithm (see Section 4.1) for the points in different dimensionality Euclidean spaces (see Fig. 2). The values of f were set to $3d$. The plot clearly shows logarithmic dependence of the greedy search path length on the dataset size, proving that the proposed structure is a navigable small world. Note that for bigger dimensionalities dependence is weaker. This is not due to different values of f (they do not affect the length of the path significantly), but possibly due shortening of the set's "topological diameter". When greedy algorithm encounters long range links, it selects the elements in the direction close to the query and disregards other directions, making the search quasi one-dimensional.

The logarithmic complexity corrupts if we add elements that gradually expand the volume of the set (i.e. non-random insertion) or update the links of elements deleting links that do not connect to a one of the closest f elements. These facts allow concluding that the navigable small world feature of the graph is based on keeping long-range links of Delaunay approximation links, created in the beginning of construction.

6.3. Parallel and distributed operation

One of the key features of the proposed approach is that the structure is expressed in terms of independent objects connected only by links. The elements can be located on different computers sharing the load. A distributed prototype based on Apache Tomcat was developed. The communication between the servers was based on a client-server model. A client (which may be any server) performs the algorithm described in Section 4.2 with an exclusion of step 12 which requires getting link list from other servers. To improve the performance (but in expense of memory requirements) of the prototype the

link list of each element also contains a copy of the object for every link in the list. This allows saving about 6d data requests between the servers at each step of the algorithm

Our earlier tests for $d=1$ with on a 4-node computer cluster show that this approach leads to almost linear scalability of total throughput with the respect to the number of processing cores in the system. Future test are required to demonstrate the work of the distributed prototype for different data.

We have also performed experiments with parallel insertion of the elements. For $d=10$ the first 1000 elements in the database where inserted serially. After that the insertion was done by 16 parallel threads. In spite of the very small database size and potentially many assembling errors, we found no measurable decrease of search accuracy in the tests, allowing massive parallel construction without any additional synchronization means.

6.4. k-Nearest neighbor search complexity scaling

Our primary quality measure is the **recall**: the ratio between relevant results and the objects obtained by the approximate search. We calculate recall by dividing the average number of true results within a search by `k`, the number of neighbors to find (so it ranges from zero to one). We also measure the fraction of visited elements to evaluate the complexity of search. This fraction is calculated by dividing the overall metric calculations during a single search by the total number of elements.

We have run tests on random Euclidean data with a fixed recall 0.999 for different dimensionalities (from 3 to 50). The fixed value of a search recall was controlled by adjusting the parameter `m`. Construction parameter `f` was set to 3*d* for all trials (optimum value for high recall) and `w` was updated to get high recall (0.99) of test data in the construction algorithm. We have used 20,000 random elements with different seed number as queries, and found `k`=10 closest neighbors during the search. The plot on Fig. 3 presents the result in a log–log scale for different dimensionalities. It shows that with the increase of the number of elements in the structure, the percentage of visited elements decreases, and the curves become close to straight lines (corresponding to power law of decay). This means that that for a fixed accuracy search complexity does not change significantly with the size of the dataset. The overall complexity of a fixed recall search together with the value `m` to get the desired recall is depicted in Fig. 4. The complexity scales as $C\log^2(n)$, just as it might be expected. One "log" comes from the average path length of the navigable small world (see Fig. 2), and the other comes from the number of multi-searches (Fig. 4).

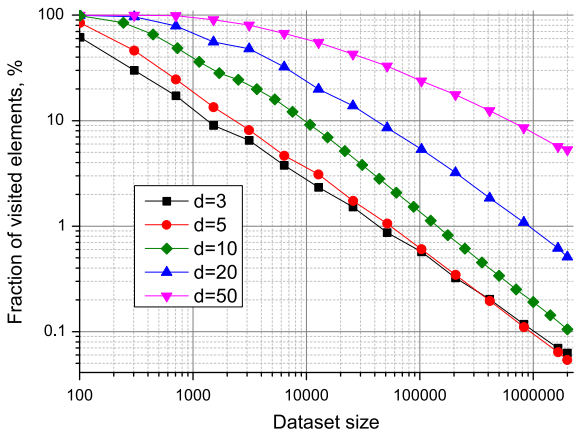

Fig. 3. Average fraction of visited elements within a single 10-NN-search with 0.999 recall versus the size of the dataset for different dimensionality.

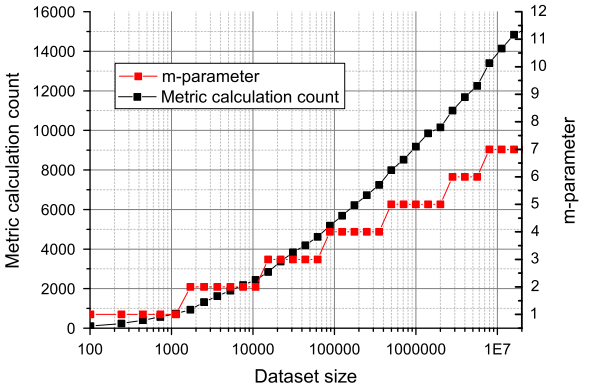

Fig. 4. Distance calculations and the value of `m` to get a 0.999 recall versus the size of the dataset for $d=20$. The metric calculation count has $C\log^2(n)$ complexity scaling.

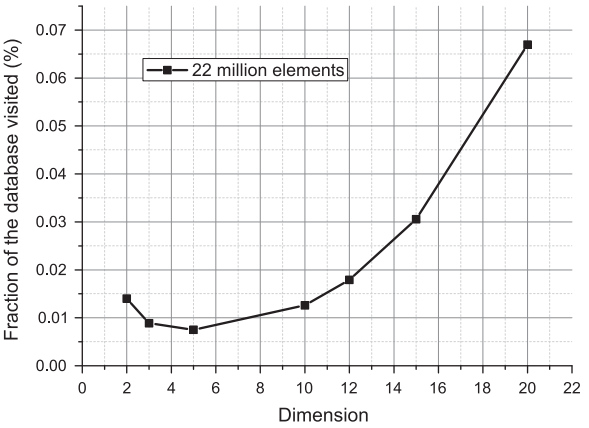

Fig. 5. Average fraction of visited elements within a single 10-NN-search with 0.999 recall for about 22 million elements dataset.

6.5. Dimensionality scaling

To test the dimensionality scaling we have plotted in **Fig. 5** the average fraction of visited elements within a single 10-NN-search with 0.999 recall for about 22 million elements dataset versus dimensionality. A plateau with an "optimal" value of the dimensionality is clearly seen from the plot. The position of the "optimum" value of dimensionality slowly shifts with increasing dataset size, which may be attributed to shorter greedy paths at higher dimensionality (see Section 6.2).

6.6. CoPHiR, comparison with other works

To get an idea about how the algorithm performs compared to previous k-NN algorithms, we have run a test from [25], a 10 million entries subset of CoPHiR collection [38]. We have used the same L_1 distance on 208 dimensional features extracted from xml documents. $k=30$ nearest neighbors were found during a search. 100 thousands different (other) elements from the dataset were used as queries. Construction of the structure was done in parallel by 16 threads and took about 2 h.

Since for optimal $f\sim 30$–40 (effective dimensionality 10–13) the algorithm achieves high recall even at a single search, we have compared the recall error (one minus recall) instead of recall (see in Fig. 6 the recall error versus fraction of the visited elements in a logarithmic scale). The achieved results are even slightly better than the expected exponential decrease. Only 0.031% of the database needed to be evaluated to get 0.999 recall, which makes the search virtually exact. In terms of throughput, at $m=1$ with recall $\sim$0.92 about 2800 searches per second can be done in parallel on our 12-core test system. The inset of the Fig. 6 shows logarithmic rise of number of evaluated elements for a single 0.999 recall search with the growth of the dataset size.

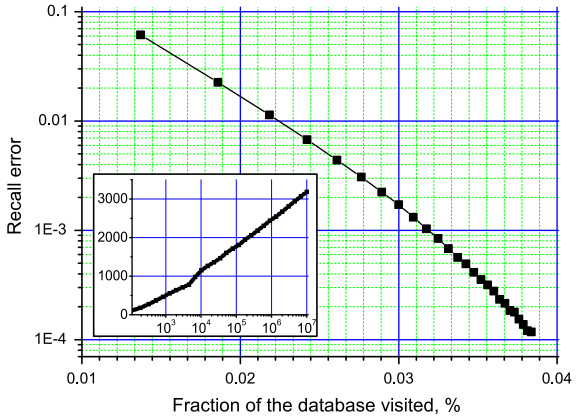

Fig. 6. Average fraction of visited elements within a single k-NN-search vs recall error for 10 M 208 dimensional vectors from CoPHiR database. The inset shows logarithmic rise of distance calculations to get 0.999 recall (vertical) with the dataset size (horizontal).

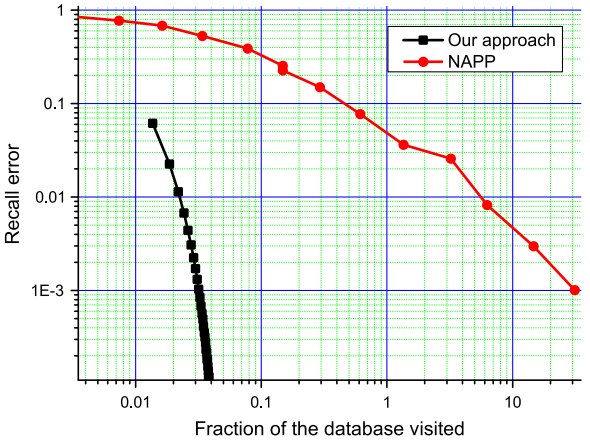

Fig. 7. Average fraction of visited elements within a single k-NN-search vs recall error for 10 million CoPHiR objects.

See the Fig. 7 for the comparison with the data for NAPP, with $K=7$ the parameter [25]. Values of σ in NAPP were selected to get best recall at fixed fraction of visited elements. Our algorithm is very effective at big dataset size, especially in case of high recall, requiring more that hundred time less metric computation at a recall of 0.999.

The comparison to Ordering Permutation index [34] at low(10^4) number of points but high dimensionality ($d=1024$), which means that the small world navigation properties do not play a critical role, showed that our algorithm yields in performance(about 65% database elements visited for our algorithm get 0.9 recall versus 42% for the OP).

7. Conclusions and open problems

We have proposed a method of organizing data into a navigable small world graph structure suited for the distributed approximate k-nearest neighbor search in metric spaces. The algorithm uses no information about inner topology of the data and space (i.e. relying only on relative distances between the objects from a set and query), and thus in principle is applicable to arbitrary metric data. The search is approximate from the probabilistic point of view.

The algorithm is very simple and easy to understand. The navigable small world feature of the graph is due to keeping long-range links of Delaunay approximation links, created in the beginning of construction, it does not use the metric space structure. All elements in the structure are of the same type, there is no central or root element. The algorithm handles insertions the same way as queries, by finding approximate neighbors for the inserted element and connecting it to them. The algorithm uses only local information on each step and can be initiated from any vertex.

Accuracy of the approximate search can be raised by using multiple searches with a random initial vertex, and the recall error decreases exponentially with the number of visited elements. Very high recall (better than 0.999) can be obtained at low complexity, making our approach a strong rival for exact search structures. Both logarithmic search and construction complexity at fixed accuracy can be achieved, and they both can be done in parallel without special care. It is shown experimentally that the dimensionality dependence is weak for Euclidean data.

Comparison with other algorithms based on permutation indexes has shown that on a big dataset there are scenarios where the proposed approach can offer much higher efficiency (up to more than an order of magnitude at fixed recall) in the number of metric computations. Only 0.03% percent of the 10 million 208-dimensional CoPHiR dataset is needed to be evaluated to achieve 0.999 recall (virtually exact search). For recall 0.93 processing speed 2800 queries/s can achieved on a single server node.

The proposed modified k-NN search algorithm provides very high efficiency and good scalability at large datasets. However, still there are several ways to optimize the structure in order to get lower complexity and/or better accuracy constants, such as

- More sophisticated algorithms for node friends selection (see Section 5). It is quite evident that selecting nearest neighbors as friends is not the best way to approximate Delaunay graph, since this approach takes into account only distances between the new element and candidates, and disregards distances between the candidates. Knowledge of internal structure of the metric space can boost search performance. In [13] is was shown that for Euclidean space the accuracy of a single search can be significantly increased while keeping the number of friends per node fixed.
- More sophisticated algorithms for navigable small world creation.
- More efficient management of multiple searches.

Even though the algorithm in principle can be applied for arbitrary metric spaces and its performance is shown experimentally in vector spaces and CoPHiR it is not clear what is the actual the range of applicability of the algorithm. Further investigations are required to clarify this point.

To sum up, simplicity, effectiveness, high scalability both in size and data dimensionality, and the distributed nature of the algorithm are a good base for building many real-world similarity search applications.

Acknowledgments

We would like to thank G. Navarro for helpful advices and I. Malkova for help in preparation of the manuscript.

References

[1] T.M. Cover, P.E. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theor. 13 (1) (1967) 21–27.
[2] M. Flickner, et al., Query by Image and Video Content: the Qbic System Computer, vol. 28, 1995, 23–32.
[3] S. Cost, S. Salzberg, A weighted nearest neighbor algorithm for learning with symbolic features, Mach. Learn. 10 (1) (1993) 57–78.
[4] B., Sarwar, G. Karypis, J. Konstan, J. Riedl, Item-Based Collaborative Filtering Recommendation Algorithms, New York, USA, 2001.
[5] R. Rhoads, W. Rychlik, A computer program for choosing optimal oligonudeotides for filter hybridization, sequencing and in vitro amplification of DNA, Nucletic Acids Res. 17 (21) (1989) 8543–8551.
[6] S. Deerwester, S.T. Dumais, T.K. Landauer, G.W. Furnas, R.A. Harshman, Indexing by latent semantic analysis, J. Am. Soc. Inf. Sci. 41 (1990) 391–407.
[7] F. Aurenhammer, Voronoi diagrams — a survey of a fundamental geometric data structure, ACM Comput. Surv. (CSUR) 23 (3) (1991) 345–405.
[8] G. Navarro, Searching in metric spaces by spatial approximation, VLDB J. 11 (1) (1999) 28–46.
[9] J., Kleinberg, The small-world phenomenon: an algorithmic perspective, in: Proceedings of the Annual ACM Symposium on Theory of Computing, vol. 32, 2000, pp. 163–170.
[10] D.J. Watts, S.H. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (6684) (1998) 440–442.
[11] O. Sandberg, Neighbor selection and hitting probability in small-world graphs, Ann. Appl. Probab. 18 (5) (2008) 1771–1793.
[12] O. Sandberg, The Structure and Dynamics of Navigable Networks, Division of Mathematical Statistics, Department of Mathematical Sciences, Chalmers University of Technology and Göteborg University, 2007.
[13] O. Beaumont, A.-M. Kermarrec, É. Rivière, Peer to Peer Multidimensional Overlays: Approximating Complex Structures, Heidelberg, Berlin, 2007.
[14] O. Sandberg, I. Clarke, The evolution of navigable small-world networks. ⟨arXiv:preprint cs/0607025⟩, 2006.
[15] J.L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (9) (1975) 509–517.
[16] R.A. Finkel, J.L. Bentley, Quad trees: a data structure for retrieval on composite keys, Acta Inf. 4 (1) (1974) 1–9.
[17] D.T. Lee, C.K. Wong, Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees, Acta Inf. 9 (1) (1977) 23–29.
[18] E. Chávez, G. Navarro, R. Baeza-Yates, J.L. Marroquín, Searching in metric space, J. ACM Comput. Surv. (CSUR) 33 (3) (2001) 273–321.
[19] S. Arya, D. Mount, Approximate nearest neighbor queries in fixed dimensions, in: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms, SODA'93, Philadelphia, PA, USA, 1993.
[20] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proceedings of STOC'98, New York, USA, 1998.
[21] E., Kushilevitz, R. Ostrovsky, Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces, in: Proceedings of STOC'98, New York, USA 1998.
[22] P., Haghani, S. Michel, K. Aberer, Distributed similarity search in high dimensions using locality sensitive hashing, in: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, New York, USA, 2009, pp. 744–755.
[23] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A. Wu. An optimal algorithm for approximate nearest neighbor searching. in: Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 1994.
[24] E. Chávez, K. Figueroa, G. Navarro, Effective proximity retrieval by ordering permutations, IEEE Trans. Pattern Anal. Mach. Intell. 30 (9) (2008) 1647–1658.
[25] E.S. Tellez, E. Chávez, G. Navarro, Succinct nearest neighbor search, in: Proceedings of SISAP, 2011.
[26] M.E., Houle, J. Sakuma. Fast approximate similarity search in extremely high-dimensional data sets, in: Proceedings of ICDE 2005.
[27] E. Chávez, E. Sadit Tellez, Navigating k-nearest neighbor graphs to solve nearest neighbor searches, Adv. Pattern Recog. (2010) 270–280.
[28] K.L. Clarkson, Nearest neighbor queries in metric spaces, Discrete Comput. Geometry 22 (1) (1999) 63–93.
[29] B. Bustos, G. Navarro, Probabilistic proximity searching algorithms based on compact partitions, J. Discrete Algorithms 2 (1) (2004) 115–134.
[30] E. Chávez, G. Navarro, A probabilistic spell for the curse of dimensionality, in: Proceedings of the Algorithm Engineering and Experimentation, Springer, 2001 pp. 147–160.
[31] E. Chávez, G. Navarro, Probabilistic proximity search: fighting the curse of dimensionality in metric spaces, Inf. Process. Lett. 85 (1) (2003) 39–46.
[32] M. Patella, P. Ciaccia, Approximate similarity search: a multi-faceted problem, J. Discrete Algorithms 7 (1) (2009) 36–48.
[33] T. Skopal, B. Bustos, On nonmetric similarity search problems in complex domains, ACM Comput. Surveys (CSUR) 43 (4) (2011) 34.
[34] E.C. Gonzalez, K. Figueroa, G. Navarro, Effective Proximity Retrieval by Ordering Permutations, IEEE Trans. Pattern Anal. Mach. Intell. 30 (9) (2008) 1647–1658.
[35] O. Beaumont, A.-M. Kermarrec, L. Marchal, E. Riviere., VoroNet: A scalable Object Network Based on Voronoi Tessellations, Long Beach, US, 2007.
[36] Y. Malkov, A. Ponomarenko, A. Logvinov, V. Krylov, Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces, in: Proceedings of the Similarity Search and Applications, Springer Berlin Heidelberg, 2012, pp. 132–147.
[37] V. Krylov, A. Ponomarenko, A. Logvinov, D. Ponomarev, Single-attribute distributed metrized small world data structure, in: Proceedings of the IEEE International Conference on Intelligent Computing and Intelligent Systems (CAS), 2009.
[38] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli, F. Rabitti, CoPhIR: A Test Collection for Content-Based Image Retrieval, 2009. Available from: ⟨http://arxiv.org/abs/0905.4627v2⟩.