Scalable Distributed Algorithm for Approximate Nearest Neighbor Search Problem in High Dimensional General Metric Spaces

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, Vladimir Krylov

MERA Labs LLC, Nizhny Novgorod, Russia

ymalkov@meralabs.com, aponom@meralabs.com,
alogvinov@meralabs.com, vkrylov@meralabs.com

Abstract. We propose a novel approach for solving the approximate nearest neighbor search problem in arbitrary metric spaces. The distinctive feature of our approach is that we can incrementally build a non-hierarchical distributed structure for given metric space data with a logarithmic complexity scaling on the size of the structure and adjustable accuracy probabilistic nearest neighbor queries. The structure is based on a small world graph with vertices corresponding to the stored elements, edges for links between them and the greedy algorithm as base algorithm for searching. Both search and addition algorithms require only local information from the structure. The performed simulation for data in the Euclidian space shows that the structure built using the proposed algorithm has navigable small world properties with logarithmic search complexity at fixed accuracy and has weak (power law) scalability with the dimensionality of the stored data.

Keywords: Similarity Search, Nearest Neighbor, Approximate Nearest Neighbor, Small World, Distributed Data Structure, Metric space

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 the distributed hash tables. While the latter have good scalability, their search functionality is limited to the exact element hash value 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 document retrieval [6])

that require the similarity search rather than just exact matching. The nearest neighbor search (NNS) problem is a mathematical formalization for the similarity search. It is defined as follows: we need to find the closest object $p \in X$ from a finite set of objects $X \subseteq \mathcal{D}$ to a given query $q \in \mathcal{D}$, where $\mathcal{D}$ is a 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 $\sigma(o', o'')$.

A naïve solution for the NNS problem is to calculate the distance function σ between q and every element from X. This leads to linear search time complexity scalability with the number of elements which is much worse than the scalability of structures with the exact value search and makes it almost impossible to use the NNS for extreme size datasets.

We suggest a solution for the nearest neighbor search problem, a data structure with a small world network topology 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 element to the query q from the data set X takes a form of searching for a vertex in the graph G.

We chose this approach based on the following:

- There are many existing well-developed algorithms for building small world networks for some special cases [7].
- Small world networks principally have no root element.
- All operations (addition and search) use only local information and can be initiated from any element that was previously added to the structure.

This gives an opportunity for building decentralized similarity search oriented storage systems where physical data location doesn't 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. Such storage systems can provide a simultaneous access to large numbers of users performing data search and addition, have good fault tolerance and are highly scalable in terms of performance and capacity.

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 a result of the algorithm to be always the exact nearest neighbor to any query, the network must contain the Delaunay graph as its subgraph, which is dual to the Voronoi tessellation [8]. However, there are major drawbacks associated with the Delaunay graph, it requires some knowledge of metric space internal structures [9] and it suffers from the curse of dimensionality [8]. Moreover the requirement of the search for the exact nearest neighbor can be not necessary (optional) for the applications described above. So the problem of finding the exact nearest neighbor can be substituted by the approximate nearest neighbor search, and thus we don't need to support the whole/exact Delaunay graph.

For the greedy search algorithm to be logarithmically scalable, the small world network should have the navigation property [7].

In this paper we present a very simple algorithm for the data structure construction based on a small world network topology with a graph $G(V,E)$ which uses the greedy search algorithm for the approximate 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 has an ability to adjust the accuracy of search without modification of the structure. Presented algorithms do not use the coordinate representation and do not presume the properties of linear spaces, because they are based only on the metric computation between the objects, and therefore are applicable to data from general metric spaces. It is shown experimentally that the dimensionality dependence is polynomial for a vector data.

2 Related Works

All papers that are dedicated to the nearest neighbor search problem can be divided into four categories: centralized nearest neighbor search structures; centralized approximate exact nearest neighbor search structures, distributed exact nearest neighbor search structures and distributed approximate nearest neighbor search structures.

2.1 Centralized Exact Nearest Neighbor Search Structures

Kd-tree[10] and quadra trees[11] were among the first works on the NNS problem. They perform well in 2-3 dimensions (search complexity is close to $O(\log n)$), but the analysis of the worst case for that structures[12] indicates $O(d * N^{1-1/d})$ search complexity, where d is the dimensionality.

Other structures which have a tree topology such as variants of kd-trees, R-trees and structures based on space-filling curves are surveyed in [13]. They also have good performance when searching in a low-dimension ($d < 4$) metric space, but they quickly lose their effectiveness with the increasing number of dimensions [14].

In general, presently there are no methods for effective exact NNS in high-dimensionality metric space. The reason behind this lies in the "curse" of dimensionality [15]. To avoid the curse of dimensionality while retaining the logarithmic scaling on the number of elements, it was proposed to reduce the requirements for the NNS problem solution, making it approximate (ANN).

2.2 Centralized Approximate Nearest Neighbor Search Structures

Thus a large number of papers appeared which proposed to search for the nearest neighbor with predefined accuracy ε (ε-NNS). For example, Arya and Mount proposed methods with search complexity $O(\log^3 n)$, but preprocessing requires $O(n^2)$ and the algorithm was applicable only to data from E^d [16] .

Kleinberg proposed two methods [17] for solving ε-NNS. First method requires $O(\mathrm{n}\log d)^{2d}$ preprocessing time and query time polynomial in d, ε , and $\log n$. The other method with preprocessing time polynomial in d, ε , and n, but with query time $O(n + d\log^3 n)$. Also both methods are applicable only to data from E^d.

The first algorithms with search complexity polynomial in d , $\log n$, ε^{-1} and polynomial preprocessing time for fixed ε were proposed by Indyk and Motwani in [18] and Kushilevitz, Ostrovsky and Rabani in [19]. Indyk and Motwani were the first ones to relax ε-ANN problem to approximate point location in equal balls (ε-PLEB). For the formulation of the problem in ε-PLEB points in metric space expand to the balls with center at this point and radius $(1+ \varepsilon)r$, it is necessary to determine which ball belongs to the query q . Also in [18] proposed a second method, which uses the concept of locality-sensitive hashing in regard to formulation of the problem ε-PLEB, with search time $\mathrm{O}(\mathrm{n}^{1/(1+\varepsilon)})$. This method however requires near quadratic memory (for small ε). In addition, the first method is applicable only for E^{d} , and the second for the Hamming space.

In general, the concept of locality-sensitive hashing has become popular in the last decade to solve the ANN problem. Other works using the concept of locality-sensitive hashing are [20], [21]. But they all have the same major drawback: each algorithm is focused on a narrow class of metrics such as Hamming distance, Jakarta or l_s norms for Euclidean space.

In [22,23] there were proposed non-distributed algorithms for the approximate k-NN problem suitable in general spaces performing well even in case of high dimensionality. The drawback for the ordering permutations index [23] is that it has a part of search algorithm with a CPU time linear dataset size scaling, and [22] is an essentially static index.

2.3 Distributed Exact Nearest Neighbor Search Structures

There are a number of distributed structures that doesn't support nearest neighbor search in general metric spaces but provide search for interval queries in attribute-based (vector) data or simple Euclidian space. MAAN [24], SCRAP[25] , Mercury [26] support multi-dimensional range queries and Voronet [27] is p2p network oriented to search nearest neighbor in E^2 based on Voronoi tessellation [8]. Every peer has coordinates in E^2 and has links to all neighbors of its Voronoi region. For the logarithmic navigation Voronet supports long-range links.

The only metric-based distributed structures are M-Chord [28], GHT [29] and MCAN[25]. MCAN uses a pivot-based technique to map the high dimensional metric data to an N-dimensional vector space, and then uses CAN protocol as its underlying structured P2P system, however they all suffer from the curse of dimensionality.

2.4 Distributed Approximate Nearest Neighbor Search Structures

Authors in [30] explain how to use locality-sensitive hashing scheme for building the structure in a distributed environment. They suggest using a two-level mapping from a d-dimensional space to the peer identifier space. However the lack of versatility inherent to all LSH schemes remains as its main drawback.

Kleinberg's work [7] has shown the possibility of using navigable small world networks for finding the nearest neighbor with the greedy search algorithm. The algorithm relied on long-range links following power-law length distribution for navigation and 2-dimensional lattice for correctness of the results. In Voronet[27] the approach was extended to arbitrary 2-dimensional data by building a two dimensional Delaunay tessellation instead of a regular lattice. In their next work [31] they have weakened the requirements on the exactness of the search in order to avoid the curse of dimensionality for the d-dimension Euclidian space. The algorithm approximates the Delaunay graph by selecting $2d+1$ neighbors that minimize the volume of the corresponding Voronoi cell. The algorithm is rather complicated; it 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 Euclidian space. The work also presented some sophisticated algorithms for managing the long range links.

3 Structure Definition

The structure S is constructed as a 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 variant of the greedy search algorithm as a base algorithm for the NNS. It traverses the graph from an element to an element each time selecting the friend closest to the query until it reaches a local minimum. See a detailed description of the algorithm in the section 4.

Links (edges) in the graph serve two distinct purposes. There is a subset of short-range links, which are used as an approximation of the Delaunay graph[8] required by the Greedy Search algorithm. Another subset is the long-range links, which are used for logarithmic scaling of the Greedy Search, they are responsible for the navigation small world properties of the constructed graph similar to the ones in Kleinberg's [7] work. The structure is illustrated on the **Fig. 1**.

In our work we focus on the approximation of the Delaunay graph and ways to nullify the errors rising from of the approximation. It can be studied independently because there is a very simple and strict way to create long range links for a predefined data set (see the section 5).

All queries in the structure are independent, they can be done in parallel and if the elements are placed randomly on physical computer nodes the processing query load is shared evenly across physical nodes. And the performance of the system (parallel queries per second) is limited only by the number of the nodes.

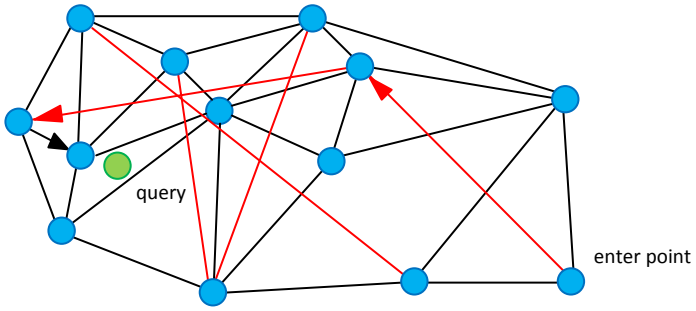

Fig. 1. Graph representation of the structure. Circles (vertices) are the data in metric space, **black** edges are the approximation of the Delaunay graph, and **red** edges are long range links for logarithmic scaling. Arrows show a sample path of the greedy algorithm from an enter point to a query (shown green).

4 Search Algorithm

4.1 Greedy Search

The basic 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 $V_{enter\_point} \in V[G]$ which is the starting point of a search (the entry point). Starting from the entry point at each vertex the algorithm computes a metric value from the query q to each vertex from the friend list of the current vertex and then selects a vertex with the minimal metric value. If the metric value between the query and the selected vertex is smaller than the one between the query and the current element, then the algorithm moves to that (new) vertex. The algorithm stops when it reaches a local minimum, a vertex whose friend list doesn't contain a vertex that is closer to the query than the vertex itself. The algorithm:

```
Greedy_Search(q: object, v_enter_point: object)
1  v_curr ← v_enter_point;
2  σ_min ← σ(q, v_curr); v_next ← NIL;
3  foreach v_friend ∈ v_curr.getFriends() do
4     if σ_fr ← σ(query, v_friend) < σ_min then
5        σ_min ← σ_fr;
6        v_next ← v_friend;
7  if v_next = Nil then return v_curr;
8  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 the entire set of elements of X, or a false closest.

If every element in the structure had in their friend list all of its Voronoi neighbors, then this would preclude the existence of false local 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 [9] (excluding the variant of the complete graph) so we cannot avoid the existence of local minima. For the problem of approximate searching as defined above it is not an obstacle since approximate search does not require the entire Delaunay graph [31].

Note that there is a distinction from the ANN problem defined in the works [16], [17] where it is expressed in terms of ε-neighborhood for which if there are several elements within the ε of the true nearest neighbor the result of the query can be any of these elements with comparable probabilities. There are no constrains on an absolute value of the distance between the false NN result and true NN result in our structure. Inaccuracy of the algorithm is «topological» in our case, meaning that the most likely result (e.g. with probability 0.95) is the true nearest neighbor, if not, the most likely it will be the second closest and so on with sharply decreasing probability. It may be more convenient to use such definition when the data distribution is highly skewed and it is hard to define one ε for all regions at the same time.

4.2 Multi-Search

In order to diminish search errors arising in a network with local minima, we propose a following modification of the search algorithm. We use a series of m searches initiated from random vertices and choose a result element that is closest to the query from the set of found elements. Since the greedy search `Greedy_Search(q, v`$_{enterPoint} \in$ `V)` is unambiguous, for each entry point $v_{enterPoint} \in V$ it either results in a success, finding the true nearest neighbor, or in a failure, finding an element that is not the nearest neighbor of q.

Thus a search of the closest element to a fixed query q may result in finding the true nearest neighbor (a global minimum) or a false nearest neighbor depending on the entry point from which the search algorithm started (see **Fig. 2**).

Since we can choose an entry point at random, there is a probability p of finding the true closest element to a particular element q. Moreover, this probability is always nonzero, because it is always possible to choose the exact nearest neighbor as an entry point, which subsequently will be returned by the greedy search algorithm. As an example the probability of finding query element in **Fig. 2** is about 73% since there are 8 elements for which taken as the entry point the algorithm will succeed and 3 elements for which he will not (3/8 results in 73%).

If for a fixed query element probability of finding the true closest in a single search attempt is p then probability of finding the true closest element in at least one of `m`

attempts is $1-(1-p)^m$, thus failure probability decreases exponentially with the number of search attempts. Thus we can improve the search precision, increasing the parameter `m` - the number of searches from random entry points. For example in the **Fig. 2** for `m` =5 the result probability is 99.985%, which is more than sufficient for the most applications.

The modified greedy search algorithm:

```
Multi_Search(object q, integer: m)
1  results: SET[objects];
2  for (i ← 0; i < m; i++) do
3    entry_point ← getRandomEntryPoint();
4    local_min ← Greedy_Search(query, entry_point)
5    if local_min ∉ results then
6       results.add(result);
7  return results;
```

By selecting the closest element from the `results` we get an answer to the query.

If `m` is comparable to the number of elements in the structure, the algorithm becomes an exhaustive search, assuming that entry points are never reused. If the graph of the network has the small-world properties, then it is possible to choose a random vertex in a number of random steps proportional to $\log n$, which doesn't affect the overall logarithmic search complexity. Therefore the overall complexity of a search will increase in no more than `m` times.

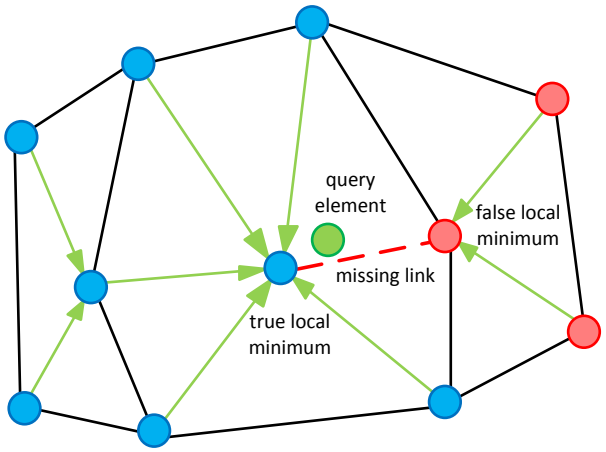

Fig. 2. An illustration of the multisearch approach. **Blue** circles represent metric space elements for which taken as entry points for the greedy algorithm it will succeed finding the true NN for a query (**green** circle). **Red** circles represent elements for which taken as entry points the algorithm will stuck in a local minimum. **Arrows** represent gradients direction of the greedy search algorithm. The probability of finding the query in a single search is about 73%. For the multisearch algorithm with `m` =5 it is 99.985%.

5 Data Addition Algorithm

Since we build an approximation of the Delaunay graph, there is a great freedom of choice of the construction algorithm. The main goal of all the works is to minimize the probability of the false local minima while the keeping number of links small. Some approaches are based on knowledge of topology of a metric space being used. For example in [31] it is proposed to build an approximate Delaunay graph which would minimize a volume of a Voronoi region (computed by the Monte-Carlo method) for a fixed number of edges for each vertex in the graph, this was done by iterating a selection of neighbors of every node in the graph several times. We propose to assemble the structure by adding elements one by one and connecting them on each step with the k closest objects which are already in the structure. It is based on the idea that intersection of the set of elements which are Voronoi neighbors and the k closest elements should be large. Another advantage of this approach was shown empirically in for one-dimensional data[32]. A graph created by such algorithm with data arriving in random order has small world navigation properties without any additional algorithms. That allows us to fully concentrate on the short-range links which approximate the Delaunay graph.

In this work we use a variant of the algorithm which is distinguished by the fact that the search for the k nearest elements uses a series of searches (an analogy to the multi-search, see 4.2).The algorithm takes three parameters: an object to be added to the structure and two positive integer numbers `k` and `w`. First, the algorithm determines a set of local minima using the procedure `Multi_Search` (see 4.2), which produces a series of `w` searches using random enter points. After that the algorithm determines a neighborhood `u` which contains all neighbors of the each found local minima. The set `u` is sorted in ascending order by distance from the object `new_object` to be added. After that `new_object` is connected with first `k` nearest elements from the set `u`.

```
Nearest_Neighbor_Add(object: new_object, integer: k, integer: w)
1  SET[object]: localMins ← Multi_Search (new_object, w);
2  SET[object]: u ←∅; //neighborhood;
3  foreach object: local_min ∈ localMins do
4    u ← u ∪ local_min.getFriends();
6  sort the set u so to satisfy the condition σ (u[i],
new_object) < σ (u[i+1], new_object)
7  for (I ← 0; i < k; i++) do
8    u[i].connect(new_object);
9    new_object.connect(u[i]);
```

The choice of the parameter `k` is not clear, it depends on the space, but it can be evaluated automatically for an unknown space with a distributed algorithm; we are planning to describe it in our next works. Note that as in 4.2 setting `w` to a big number is equivalent to an exhaustive search of the closest elements in the structure. More on the choice of `w` and `k` see in the next section.

6 Test Results and Discussion

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. For a test dataset we have used:

- Uniformly distributed random points with a L_2 (Euclidean distance) proximity function (up to 10^6 elements).
- To test our algorithm in a general metric space we have used a database of chemical compounds [33] with a Tanimoto [34] distance function. We have randomly selected 10^5 elements from the database to test the algorithm.
- A subset of the TREC-3 documents collection containing 24276 documents[23] for comparison with other works.

Small World.

To verify the small world properties of the proposed structure we have measured the average path length induced by the greedy search algorithm for the vectors and chemical compounds (see **Fig. 3**). The plot clearly shows a logarithmic dependence on the dataset size proving it is a navigable small world. Thus the complexity of a single search scales logarithmically. It can be shown that the small world properties retain at any size (we are going to focus on it in one of our next works). Note that for bigger dimensionalities dependence is weaker due to smaller diameter of a set at a fixed number of elements.

Construction Parameters.

We adjusted the number of search attempts m, so that the probability of finding the true closest element to the query was not less than a fixed value (we took 95% as a reference).

To test the scaling of the search algorithm with the number of elements n we have plotted (see **Fig. 4**) the number of multi-searches `m` required to get the 95% true nearest neighbor rate versus the size of the dataset for d=10 and different `w` parameters of the construction algorithm. For `w`=20 the dependence is clearly logarithmic up to 10^6 elements. For low values of `w` the algorithm complexity dependence deviates from the expected. Arrows denote the point where the dependence deviates from the logarithmic for `w`=1..4. One can see that the points are almost equidistant in the logarithmic scale.

So, if we need to get the logarithmic scaling up to n elements we have to have $\mathtt{w} > A \times \log(n); (6.1)$, where A is a constant value. And the overall complexity of both the search and the construction algorithms can be made logarithmic at the same time. Such dependence on the construction parameters can be easily understood. For the low `w` parameters the probability of finding the true nearest neighbor to a new element is low and the algorithm cannot choose the closest neighbors links correctly. The number of searches required to get P close to unity scales logarithmically with the

size of the dataset, leading to equation (6.1). We can set w high enough for any reasonable size of the dataset (like 10^{100}) while keeping acceptable construction complexity or if the size of the dataset is known (or evaluated dynamically) we can always set the parameters optimal and maintain an overall logarithmic scaling.

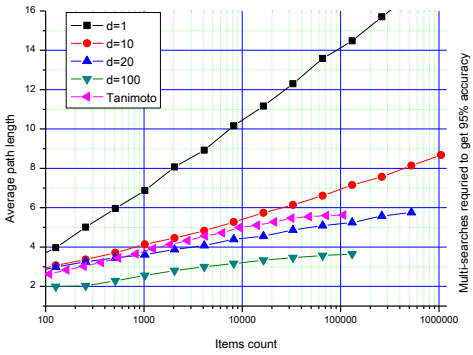

Fig. 3. The average hop count induced by a greedy search algorithm for different dimensionality Euclid data and for a chemical compounds dataset (k=10, w=20). The navigable small world properties are evident from the logarithmic scaling.

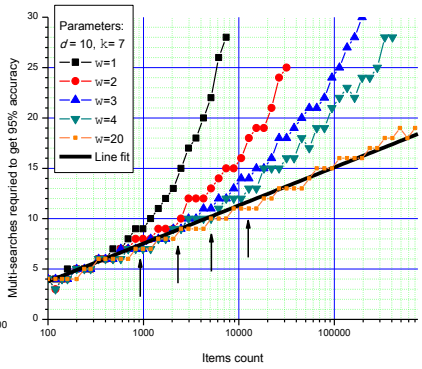

Fig. 4. The number of multi-searches required to get the 95% true nearest neighbor rate versus the size of the dataset for different w parameters of the construction algorithm. Arrows denote points where the dependence deviates from the logarithmic. The points are almost equidistant in the log scale.

Fig. 5 presents the number of multi-searches m for the same parameters as in **Fig. 4** setting $w=\left[A\times\log(n)-c\right]$ for A=1.5, 2, 2.5, 3. For any value of A the scaling stays logarithmic but at expense of worse complexities for the small values of A. Setting A higher than 2.5 does not affect the complexity of the search.

To define the best choice of the parameter k we have plotted the probability of failing finding the true nearest neighbor versus the fraction of visited elements (metric calculations) for d=10 and different parameters k (see **Fig. 6**). For k smaller than $2...3\cdot d$ there is a significant fall of performance, while for bigger values of k there is a very slow decay with the rise of the parameter. For d=2…50 it was verified that the optimal value for k is close to $3\cdot d$. Also one can see that the probability of a wrong NN result falls exponentially with the fraction of visited elements confirming assumptions from section 4.

The bottom line is that the optimal value for k is $3\cdot d$; the value of w has to be dynamically changed $A\times\log(n_{current})$ with a constant A or set fixed to $A\times\log(n)$, where n is the maximum database size.

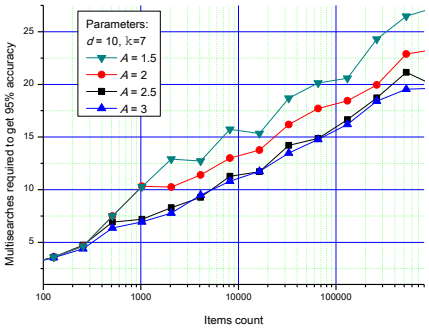

Fig. 5. The number of multi-searches required to get the 95% true nearest neighbor rate versus the size of the dataset for a logarithmic scaling of `w` (*A*=1.5, 2, 2.5, 3).

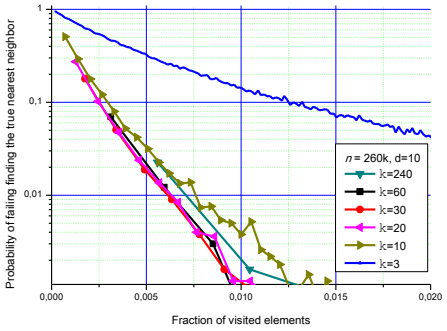

Fig. 6. Probability of failing finding the true nearest neighbor versus fraction of visited elements for d=10, $n = 2.6 \cdot 10^5$.

Absolute Speedup and Scaling.

The graph (**Fig. 7**) shows the percent of visited (extracted) elements (vertical) versus the dataset size (horizontal) in a log-log scale for different dimensionalities. `k` was fixed to $3 \cdot d$ for all trials and `w` was fixed to a big number. The plot 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 with an angle of 45 degrees (corresponding to the $1/n$ law of decay). This means that the single search complexity does not change significantly with the size of the dataset. From the graphs the overall scaling for complexity of the search can be extracted. It turns out that it scales as $\log^2(n)$, just as it might be expected. One "log" coming from the average path length and the other is from the number of multi-searches.

We have also plotted the average fraction of visited elements for n= $2.6 \cdot 10^5$ in a log-log scale to check the dimensionality dependence (see **Fig. 8)**, it can be approximated by a $d^{1.7}$ power law. Judging on the **Fig. 7** it seems that for low d with rise of n at some size the difference in performance between the dimensionalities diminishes. It might be suspected that such behavior will be the same for bigger dimensionalities but it requires further study.

Overall, the measured search complexity scaling for n>10^5 and d = 5..100 is not worse than $d^{1.7} \times \ln^2(n) \times \ln(1/P_{fail})$ and the construction complexity (deduced from the search complexity) is $d^{1.7} \times n \ln^2(n)$, where P_{fail} is an acceptable probability of failing finding the true nearest neighbor.

To get an idea about how the algorithm performs compared to the other k-NN algorithms we have run a test from [23], a subset of collection TREC-3 documents containing 24276 documents. For a k-NN algorithm we have used a part of the construc-

tion algorithm from section 5. To get the averaged 90% recall of 9 documents from the database it required visiting 5% of the database compared to about 2% from the [23]. We believe it is a good result for such a simplified algorithm. We have also made a slight modification of the k-NN search algorithm, changing the stop condition (the algorithm continues to travel the graph while it can improve distance for the k-th element) yielding about 2.5% extraction of the database at the same recall, very close to the state of art.

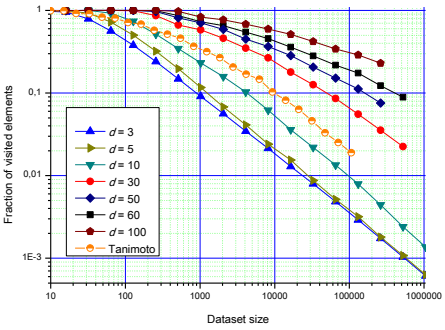

Fig. 7. Average fraction of visited elements within a single NN-search versus the size of the dataset for different dimensionality and data types.

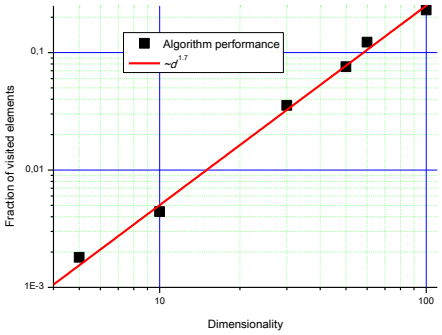

Fig. 8. Average fraction of visited elements within a single Nearest Neighbor search versus the dimensionality of the dataset for n = 262k with a power-law fit.

7 Conclusions And Future Work

We have proposed a method of organizing data into a distributed small world graph structure suited for the distributed approximate nearest neighbor search in a metric space. The algorithm uses no information about inner topology of the data and space, thus it is applicable to arbitrary metric data. The algorithm is very simple and easy to understand. All elements in the structure are of the same type, there is no central or root element. There is no dedicated algorithm for managing the small world properties, they arise automatically. The algorithm uses only local information on each step and can be initiated from any vertex. The search is approximate from the topological point of view. An unsuccessful Nearest Neighbor query typically results in the second nearest element.

Accuracy of the approximate search can be tuned by using multiple searches with a random initial vertex and the probability of finding a false nearest neighbor decreases exponentially with the number of multi-searches.

The performed simulation for data in the Euclidian space shows that the structure built using the proposed algorithm has the navigable small world property. Both loga-

rithmic search and construction complexity at fixed accuracy can be achieved with appropriate algorithm parameters. There are reasons to believe such behavior will be retained for any dataset size. The algorithm also shows a power law scalability of metric calculation count with dimensionality of the stored data. Simulations for chemical compounds and documents have shown the effectiveness of the approach for non-Euclidian spaces comparable to best algorithms.

The proposed structure was intentionally slimmed-down to demonstrate its scalability over the dataset size and dimensionality. There are several ways to optimize the structure in order to get lower complexity or/and better accuracy constants, such as:

- More complicated algorithms for node friends selection (see sec. 5). It is obvious that selecting nearest neighbors as friends is not the best way to approximate Delaunay graph since it takes into account only distances between the new element and candidates and neglects distances between the candidates. Knowledge of internal structure of the metric space can boost up search performance. In [31] is was shown that for Euclidean space the accuracy of a single search can be significantly increase while keeping the number of friends per node fixed.
- More complicated search algorithms can be used. Excluding already visited elements in consequent searches or/and changing the stop parameters in search algorithm can potentially reduce the number of metric computations several times at the same accuracy.
- More complicated algorithms for navigable small world creation suitable for correlated (non-random) data.

As a future work, we are going to enhance the performance of the structure while keeping good scalability and distributed nature and to make a detail comparison with the state of art algorithms from the area.

Summing up, simplicity, high scalability both with size and data dimensionality and the distributed nature of the algorithm are a good base for building many real-world extreme dataset size high dimensionality similarity search applications.

8 References

1. Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. Information Theory, IEEE Transactions on **13**(1), 21-27 (1967).
2. Flickner, M., et.al.: Query by image and video content: the QBIC system Computer **28**(9), 23-32 (1995).
3. Cost, S., Salzberg, S.: A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. Machine Learning **10**(1), 57-78 (1993).
4. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: , New York, USA 2001. Proceedings of the 10th international conference on World Wide Web, pp. 285-295.
5. Rhoads, R., Rychlik, W.: A computer program for choosing optimal oligonudeotides for filter hybridization, sequencing and in vitro amplification of DNA. Nucletic Acids Research **17**(21), 8543-8551 (1989).

6. Deerwester, S., et.al.: Indexing by Latent Semantic Analysis. J. Amer. Soc. Inform. Sci. **41**, 391-407 (1990).
7. Kleinberg, J.: The Small-World Phenomenon: An Algorithmic Perspective. Annual ACM Symposium on Theory of Computing **32**, 163-170 (2000).
8. Aurenhammer, F.: Voronoi diagrams — a survey of a fundamental geometric data structure. ACM Computing Surveys (CSUR) **23**(3), 345-405 (1991).
9. Navarro, G.: Searching in metric spaces by spatial approximation. Paper presented at the String Processing and Information Retrieval Symposium, Cancun, Mexico.
10. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM **18**(9), 509-517 (1975).
11. Finkel, R.A., Bentley, J.L.: Quad Trees: A Data Structure for Retrieval on Composite Keys. Acta Informatica **4**(1), 1-9 (1974).
12. Lee, D.T., Wong, C.K.: Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. Acta Informatica **9**(1), 23-29 (1977).
13. Samet, H.: The design and analysis of spatial data structures. Addison-Wesley Pub, (1989).
14. Arya, S.: Accounting for boundary effects in nearest-neighbor searching. Discrete & Computational Geometry **16**(2), 155-176 (1996).
15. Chávez, E., et.al.: Searching in metric space. Journal ACM Computing Surveys (CSUR) **33**(3), 273-321 (2001).
16. Arya, S., Mount, D.: Approximate nearest neighbor queries in fixed dimensions. In: SODA '93 Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms, Philadelphia, PA, USA 1993, pp. 271-280.
17. Kleinberg, J.: Two algorithms for nearest-neighbor search in high dimensions. In: STOC '97 New York, USA 1997. Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pp. 599 - 608.
18. Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: STOC '98 New York, USA 1998. Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 604-613.
19. Kushilevitz, E., Ostrovsky, R., Rabani, Y.: Efficient search for approximate nearest neighbor in high dimensional spaces. In: STOC '98, New York, USA 1998. Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 614 - 623.
20. Gionis, A., Indyk, P., Motwani, R.: Similarity Search in High Dimensions via Hashing. In: VLDB '99 San Francisco, USA 1999. Proceedings of the 25th International Conference on Very Large Data Bases, pp. 518-529.
21. Andoni, A., Indyk, P.: Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In: FOCS'06, Berkeley, USA 2006. Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06) pp. 459-468.
22. Houle, M.E., Sakuma, J.: Fast Approximate Similarity Search in Extremely High-Dimensional Data Sets. In: ICDE 2005.

23. Chávez, E., Figueroa, K., Navarro, G.: Effective Proximity Retrieval by Ordering Permutations. Pattern Analysis and Machine Intelligence, IEEE Transactions on **30**(9), 1647 - 1658 (2008).
24. Cai, M., Frank, M., Chen, J., Szekely, P.: MAAN: A Multi-Attribute Addressable Network for Grid Information Services. Journal of Grid Computing **2**(1), 3-14 (2004).
25. Ganesan, P., Yang, B., Garcia-Molina, H.: One torus to rule them all: multi-dimensional queries in P2P systems. In, New York, USA 2004. Proceedings of the 7th International Workshop on the Web and Databases, pp. 19-24
26. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. In, New York, USA 2004. Proceedings of Applications, technologies, architectures, and protocols for computer communication, pp. 353-366.
27. Beaumont, O., Kermarrec, A.-M., Marchal, L., Riviere, E.: VoroNet: A scalable object network based on Voronoi tessellations. In, Long Beach, US 2007. Proceedings of International Parallel and Distributed Processing Symposium, p. 20.
28. Novak, D., Zezula, P.: M-Chord: A Scalable Distributed Similarity Search Structure. In, San Diego 2001. Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 149-160.
29. Batko, M., Gennaro, C., Zezula, P.: Similarity Grid for Searching in Metric Spaces. Lecture Notes in Computer Science **3664** (2005).
30. Haghani, P., Michel, S., Aberer, K.: Distributed similarity search in high dimensions using locality sensitive hashing. Paper presented at the 12th International Conference on Extending Database Technology: Advances in Database Technology, New York, USA.
31. Beaumont, O., Kermarrec, A.-M., Rivière, É.: Peer to peer multidimensional overlays: approximating complex structures. In, Berlin, Heidelberg 2007. Proceedings of the 11th international conference on Principles of distributed systems.
32. Krylov, V., Ponomarenko, A., Logvinov, A., Ponomarev, D.: Single-attribute Distributed Metrized Small World Data Structure. Paper presented at the IEEE International Conference on Intelligent Computing and Intelligent Systems (CAS).
33. Wang, Y., Xiao, J., Suzek, T.O., Zhang, J., Wang, J., Bryant, S.H.: PubChem: a public information system for analyzing bioactivities of small molecules. Nucl. Acids Res. **37**, W623-W633 (2009).
34. James, C.A., Weininger, D., Delaney, J.: Fingerprints-Screening and Similarity, http://www.daylight.com/dayhtml/doc/theory/theory.toc.html (1997).