Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs

Article *in* IEEE Transactions on Pattern Analysis and Machine Intelligence · March 2016

DOI: 10.1109/TPAMI.2018.2889473

CITATIONS
1,748

READS
6,764

2 authors:

Yu A Malkov
DeepMind
51 PUBLICATIONS **3,379** CITATIONS

SEE PROFILE

Dmitry A Yashunin
Institute of Applied Physics of the Russian Academy of Sciences (IAP RAS), Nizhny …
20 PUBLICATIONS **1,830** CITATIONS

SEE PROFILE

Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs

Yu. A. Malkov[1], D. A. Yashunin

1. Federal state budgetary institution of science Institute of Applied Physics of the Russian Academy of Sciences, 46 Ul'yanov Street, 603950 Nizhny Novgorod, Russia

Abstract

We present a new algorithm for the approximate K-nearest neighbor search based on navigable small world graphs with controllable hierarchy (Hierarchical NSW). The proposed approach is fully graph-based, without any need for additional search structures which are typically used at coarse search stage of the most proximity graph techniques. Hierarchical NSW incrementally builds multi-layer structure consisting from hierarchical set of proximity graphs (layers) for nested subsets of the stored elements. The maximum layer in which an element is present is selected randomly with exponentially decaying probability distribution. This allows producing graphs similar to the previously studied Navigable Small World (NSW) structures while additionally having the links separated by their characteristic distance scales. Starting search from the upper layer together with utilizing the scale separation boosts the performance compared to the NSW and allows a logarithmic complexity scaling. Additional employment of a heuristic for selecting proximity graph neighbors significantly increases performance at high recall and in case of highly clustered data. Performance evaluation has demonstrated that the proposed general metric space method is able to strongly outperform many previous state-of-art vector-only approaches. Similarity of the algorithm to the skip list structure allows straightforward balanced distributed implementation.

Introduction

Constantly growing amount of the available information resources leads to high demand in scalable and efficient similarity search data structures. One of the generally used approaches for information search is the K-Nearest Neighbor Search (K-NNS). The K-NNS assumes you can measure distance between the data elements and aims at finding the *K* elements from the dataset which minimize the distance to a given query. Such algorithms are used in many applications such as non-parametric machine learning algorithms [1], image features matching in large scale databases [2] and semantic document retrieval [3]. A naïve approach to solve the problem is to compute distances between the query and every element in the dataset and select elements with minimal distance. However the complexity of the naïve approach scales linearly with the size of the dataset making it infeasible for big data. This has led to a high interest in development of fast and scalable K-NNS algorithms.

Exact solutions for the K-NNS [4-6] may offer a substantial search speedup only in case of relatively low dimensional data due to “curse of dimensionality”. To overcome this problem a concept of Approximate Nearest Neighbors search (k-ANNS) was proposed, which relaxes the condition of exact search by allowing a small number of errors. The quality of inexact search (the recall) is usually defined as the ratio between the number of found true nearest neighbors

and *K*. Most popular K-ANNS solutions are based on approximated versions of tree algorithms [7-9] and hashing techniques [10, 11].

Proximity graph K-ANNS algorithms [12-19] have recently gained popularity offering a better performance compared to tree techniques in some cases. In the vast majority of studied graph algorithms searching the nearest neighbors takes a form of greedy routing k-Nearest Neighbor (k-NN) Graphs. One of the main drawbacks of this approach are the power law scaling of the number of hops during the routing process [20, 21] as well as a possible loss of global connectivity in such graphs. To overcome this problems many hybrid approaches have been proposed that use auxiliary algorithms applicable only for vector data (such as kd-trees [12, 14] and Cartesian concatenation [13]) to find candidate seeds by doing a coarse search.

The first works to consider networks with polylogarithmic greedy routing hops scaling (such networks are called *navigable*) were done by J. Kleinberg [22, 23] as social network models for the famous Milgram experiment [24]. Kleinberg studied a variant of random Watts-Strogatz networks [25] on a regular lattice with a specific long link length distribution $r^{-\alpha}$. For α=d (where d is the dimensionality of the lattice) the number of hops to get to the target by greedy routing scales polylogarithmically (instead of a power law for any other value of α). The idea has led to development of many K-NNS and K-ANNS algorithms based on the navigation effect [26-29]. Kleinberg's navigability criterion in principle can be extended for more general spaces, however, unfortunately, in order to build a Kleinberg's navigable network one has to know the data distribution, which strongly limits the approach.

In [30-32] authors proposed a new proximity graph K-ANNS algorithm called Navigable Small World (NSW, also known as Metricized Small World, MSW), which utilized *navigable* graphs with long range links constructed by a much simpler model. The model is based on growth and connection to the approximate nearest neighbors and studied as Growing Homophilic (GH) networks in [33]. The GH networks are constructed by consecutive insertion of elements in random order by connecting them to *M* closest neighbors from the previously inserted elements. Links to the closest neighbors of the elements inserted in the beginning of the construction later become bridges connecting different parts of the network and allow logarithmic scaling of the greedy algorithm hop number. It was suggested [33] that the mentioned network formation mechanism based on growth and homophily may be responsible for navigability of large-scale biological neural networks (presence of which is disputable): similar models were able to describe growth of small brain networks, while the GH mechanism predicts several high level features observed in large scale neural networks.

The NSW algorithm uses a variant of greedy search with overall polylogarithmic time complexity and can outperform rival algorithms on many real-world datasets [34, 35]. However, the polylogarithmic complexity scaling of the algorithm causes notable performance degradation on large datasets, especially in the case of low dimensional data [35].

In this paper we propose a new algorithm based on ideas close to NSW, which offers a much better logarithmic complexity scaling. The main contributions are smart selection of the graph's

enter-point node, separation of links by different scales and using a slightly more complicated heuristic to select the neighbors. Alternatively the Hierarchical NSW algorithm can be seen as an extension of the probabilistic skip list structure [36] with proximity graphs instead of the linked lists.

Core idea

The base NSW algorithm builds a graph by consecutive insertion of elements through connecting them to *M* previously inserted closest neighbors. Links to the closest neighbors of elements inserted in the beginning of the construction of the NSW graph later become long-range links connecting distant network parts allowing logarithmic scaling of the number of hops [33].

The NSW algorithm performs reasonably fast, however, it still has several drawbacks such as low performance at low dimensional data and polylogarithmic scalability of the total number of distance calculations at best. There are well-known alternative construction models for navigable small-world networks. They, however, do not provide a better scaling: Kleinberg's model as well as its derivatives also offers a polylogarithmic scalability at best, while the scale-free models [37-39] have an even worse power law scaling [33]. It is not clear whether a single layer graph can have a logarithmic scalability in principle.

The process of routing in navigable small-world networks with strong correlation between the degree and characteristic connections distance was studied in detail in [33, 37] and can be divided into two phases: "zoom-out" and "zoom-in" [37]. The algorithm starts in the "zoom-out" phase from a low degree node and traverses the graph increasing the node's degree until the characteristic radius of the node links length reaches the scale of the distance to the query. Before the latter happens, the average degree of a node can stay relatively small, which leads to an increased probability of being stuck in a distant false local minimum. Obviously one can avoid this problem in the NSW by starting the search from a node with the maximum degree (good candidates are the first nodes inserted in the NSW structure [33]), directly going to the "zoom-in" phase of the search. Simulations show that setting hubs as starting points substantially increases probability of successful routing in the structure and offers significantly better performance at low dimensional data. However it still has only a polylogarithmic complexity scalability of a single greedy search and performs worse on high dimensional data compared to the unmodified NSW.

The reason for the polylogarithmic complexity scaling of a single greedy search in the NSW is that the overall number of distance computations is roughly proportional to a product of the average number of greedy algorithm hops and the average degree of the nodes on the greedy path. The average number of hops scales logarithmically [32, 33], while the average degree of the nodes on the greedy path also growths logarithmically due to the facts that: 1) the greedy search tends to go through the hubs [33, 37]; 2) the number of hub connections is growing logarithmically with an increase of the network size. Thus we get an overall polylogarithmic dependence of the resulting complexity.

The idea of the Hierarchical NSW algorithm is to separate the links according to their length scale, producing a multilayer graph. In this case we can evaluate only a needed portion of connections for each element independent of the networks size, thus getting a logarithmic scalability at the "zoom-in" phase (see fig. 1 for illustration). The search starts from the upper layer greedily selecting elements only from the layer until a local minimum is reached. After that the search switches to the lower layer and restarts from the element which was the local minimum in the previous layer. The average number of connections per element in all layers can be made constant thus allowing getting a logarithmic complexity scaling.

One way to form such a layered structure is to explicitly set links with different distance scales by artificially introducing layers. For every element we define an integer *level* which defines the maximum layer which element belongs to. For all elements that are present in a layer a proximity graph (i.e. graph containing only "short" links which approximate Delaunay graph) is incrementally built. If we set exponentially decaying probability of item's *level* we get a logarithmic scaling of the number of layers in the structure. The search procedure is an iterative greedy search starting from the top layer. In case we merge connections from all layers, the structure becomes similar to the NSW graph (in this case the *level* can be put in correspondence to the node degree in NSW). Note that in contrast to the NSW, the Hierarchical NSW construction algorithm does not require the elements to be inserted in random order.

The Hierarchical NSW idea is also very similar to a well-know 1D probabilistic skip list structure [36] and can be described using its terms. The major difference is that we generalize the structure by replacing the linked list with proximity graphs. Hierarchical NSW approach thus can utilize the same methods for making the distributed approximate search/overlay structures [40].

For selection of proximity graph connections we utilized a heuristic that uses the distances between the candidate elements to create diverse direction connections (a similar algorithm was utilized in the spatial approximation tree [5] to select the tree children) instead of just using the closest neighbors. The heuristic examines the candidates starting from the closest and creates a connection to a candidate only if it is closer to the base element compared to any of the already connected elements (see Algorithm section for details). When the number of candidates is large enough the heuristic allows getting the exact relative neighborhood graph [41] (a minimal subgraph of the Delaunay graph deducible using only distances between the nodes) as a subgraph, thus easily keeping a global connected component, even in case of highly clustered data (see fig. 2 for illustration). Note that the heuristic creates many extra connections compared to the relative neighborhood graph, allowing control of the connections number which is important for search performance. For the case of 1D the heuristic allows getting the exact Delaunay graph (which coincides with the relative neighborhood in this case) by using only distances between the elements, thus making a direct transition from the Hierarchical NSW to the 1D probabilistic skip list algorithm.

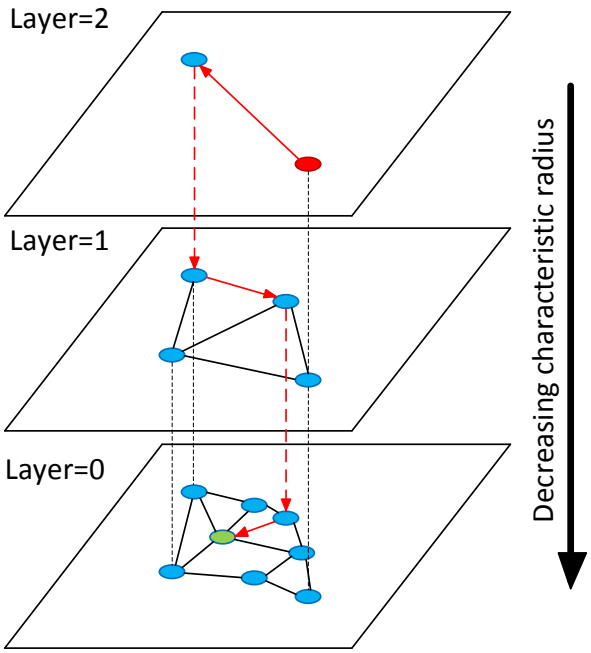

Fig. 1. Illustration of the Hierarchical NSW idea. The search starts from an element from the top layer (shown red). Red arrows show direction of the greedy algorithm from the entry point to the query (shown green).

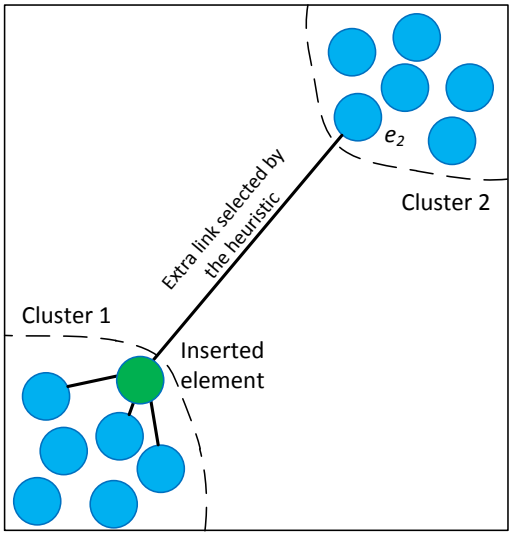

Fig. 2. Illustration of the heuristic used to select the neighbors in the proximity graph. The data in the example consists from two isolated clusters. A new element is being inserted on the boundary of the first cluster. All of the closest neighbors of the new element are belonging to the same (first) cluster thus missing the Delaunay graph links between the clusters. The heuristic, however, selects an element e_2 from another cluster maintaining the global connectivity in case the inserted element is the closest to e_2 compared to any other element from Cluster 1.

Algorithm

Network construction algorithm is based on sequential insertions of the metric elements into the structure. For every inserted element an integer maximum layer *level* is randomly selected with an exponentially decaying probability distribution (normalized by a *levelMult* parameter, see alg. 1).

The first phase of the insertion process starts from the top layer by greedily traversing the graph in order to find the closest neighbor in the layer. After the algorithm finds a local minimum in a layer, it continues the search from the next layer using the found closest neighbors from previous layer as enter points, and the process repeats. Closest neighbors at each layer are found by a variant of the greedy search algorithm described in alg. 2, which is updated compared to described in [32]. To obtain approximate *K* nearest neighbors in some layer *level*, a dynamic list of *ef* closest of the found elements (initially filled with a list of enter points) is kept during the search. The list is updated at each step by evaluating the

neighborhood of the closest previously non-evaluated element in the list until the neighborhood of every element from the list is evaluated. Such stop condition allows avoiding bloating of priority queues by discarding candidate elements that are not fit to the list. The distinctions from the algorithm described in [32] (along with queue optimizations) is that: 1) the enter point is a fixed parameter; 2) instead of changing the number of multi-searches, the quality of the search is controlled by different parameter *ef* (which was set to *K* in ref. [32]). During the first phase of the search the *ef* parameter is set to 1 (simple greedy search).

When the search reaches layer equal or less than level, the second phase of the construction algorithm is started which differs in two points: 1) the ef parameter is increased from 1 to `efConstruction` in order to control the recall of the greedy search procedure; 2) the found closest neighbors on each level are also used as candidates for connections of the inserted element.

Two methods for selection of *M* neighbors from the candidates were considered for the algorithm: simple connection to the closest elements (alg. 3) and the heuristic that uses the distances between the candidate elements to create connections in diverse directions (alg. 4), described in the "Core idea" section. The maximum number of connections that an element can have per layer is defined by a parameter M_{max} for every layer higher than zero (special parameter M_{max0} is used for the ground layer separately). If a node is already full at the moment of making of a new connection, then its connection list gets shrunken excluding a neighbor by the same heuristic that described in algs. 3-4.

The insertion procedure ends when the connections of the inserted elements are established on the ground (zero) layer.

Algorithm 1

```
Insertion (object q, integer: M, efConstruction, levelMult)
1 Set [object] tempRes, candidates, visitedSet, enterPoints=enterpoint
2 integer level=floor(-log(random(0..1))*levelMult) // Selecting a random level
3 for i=maxLayer to level-1 do:
4      tempRes=SearchAtLayer (q, enterPoints, M, 1, i)
5      enterPoints=closest elements from tempRes
6 for i=min(maxLayer,level) downto 0 do:
7      tempRes=SearchAtLayer (q, enterPoints, M, efConstruction, i)
8      select best M elements from tempRes by using a heuristic // alg. 3 or alg. 4
9      bidirectionally connect best M elements from tempRes to q
10     shrink lists of connected elements
11     enterPoints=closest elements from tempRes
13 if (level> maxLayer) do: // update the enterpoint
14     maxLayer=level
15     enterpoint=q
```

Algorithm 2

```
SearchAtLayer (object q, Set[object] enterPoints, integer: M, ef, layer)
1 Set [object] visitedSet
2 priority_queue [object] candidates (closer - first), result (further - first)
3 candidates, visitedSet, result ← enterPoints
```

```
repeat:
    object c =candidates.top()
    candidates.pop()
    //check stop condition:
    if d(c,q)>d(result.top(),q) do:
          break
    //update list of candidates:
    for_each object e from c.friends(layer) do:
          if e is not in visitedSet do:
                add e to visitedSet
                if d(e, q)< d(result.top(),q) or result.size()<ef do:
                      add e to candidates, result
                      if result.size()>ef do:
                            result.pop()
return best k elements from result
```

Algorithm 3

```
SelectNeighbors_simple(object baseElement, Set [object] candidates, integer M)
return M closest elements from candidates
```

Algorithm 4

```
SelectNeighbors_heuristic(object baseElement, Set [object] candidates, integer M)
Set [object] result, tempList
extend the neighborhood of candidates (optional)
sort candidates so that items closer to baseElement come first
for_each object e from candidates do:
    if e is closer to baseElement compared to any element from result do:
          add e to result
    else do:
          add e to tempList
    if result.size()>M do:
          break
// (optionally) add some discarded connections:
sort tempList so that items closer to baseElement come first
for_each object e from tempList do:
    add e to result
    if(result.size()>=M) do:
          break
return result
```

The K-ANNS algorithm used in the Hierarchical NSW is presented in alg. 5. It is roughly equivalent to the insertion algorithm for an item with *level*=0. The difference is that the closest neighbors found at ground layer which were used as candidates for the connections are now returned as the search result. The quality of the result is controlled by the *ef* parameter (corresponding to *efConstruction* in the construction algorithm).

Algorithm 5

```
K-NNSearch (object query, integer: ef)
Set [object] tempRes, enterPoints=[enterpoint]
for i=maxLayer downto 1 do:
    tempRes=SearchAtLayer (query, enterPoints, M, 1, i)
    enterPoints =closest elements from tempRes
tempRes=SearchAtLayer (query, enterPoints, M, ef, 0)
return best K of tempRes
```

Performance evaluation

Influence of parameters

Algorithm construction parameters *levelMult* and M_{max0} are responsible for maintaining the small world navigability in the constructed graphs. Setting *levelMult* to zero (this corresponds to a single layer in graph) and M_{max0} to M leads to production of directed k-NN graphs with power law search complexity well studied before for K-ANN search [16, 21] (assuming using the alg. 3 for neighbor selection). Setting *levelMult* to zero and M_{max0} to infinity leads to production of the NSW graphs with polylogarithmic complexity [30, 32]. Finally, setting *levelMult* to some non-zero value leads to emergence of controllable hierarchy graphs with logarithmic search complexity by introduction of layers (see the Algorithm section).

To achieve the optimum performance advantage of the controllable hierarchy, the overlap between neighbors on different layers (i.e. percent of element neighbors that are also belong to other layers) has to be small. In order to decrease the overlap we need to decrease the *levelMult*. However, at the same time, decrease of the *levelMult* leads to an increase of average hop number during a greedy search on each layer, which negatively affects the performance. This leads to existence of the optimal value for the *levelMult* parameter.

An obvious choice for the optimal *levelMult* is setting it to 1/log(M): this makes the generated *level* a dimensionless quantity (assuming that M has some specific units). The simulations done on an Intel Core i5 2400 CPU agree well with this assumption, demonstrating a very large speedup on low dimensional data when increasing the *levelMult* from zero (see fig. 3 for 1-NN searches, K=1 on 10M random d=4 vectors, a suggested value for *levelMult* is shown by an arrow). It is hard to expect the same behavior for high dimensional data since in this case the k-NN graph already has very short greedy algorithm paths [20]. Surprisingly, increasing the *levelMult* from zero leads to a measurable increase in speed on very high dimensional data (100k dense random d=1024 vectors, see plot in fig. 4), and does not introduce any penalty for the Hierarchical NSW approach. For mid-dimensional data, such as SIFT vectors [2], the performance advantage of increasing the *levelMult* is moderate (see fig. 5 for 10-NN search performance on 5 million 128-dimensional SIFT vectors from the learning set of BIGANN [42]).

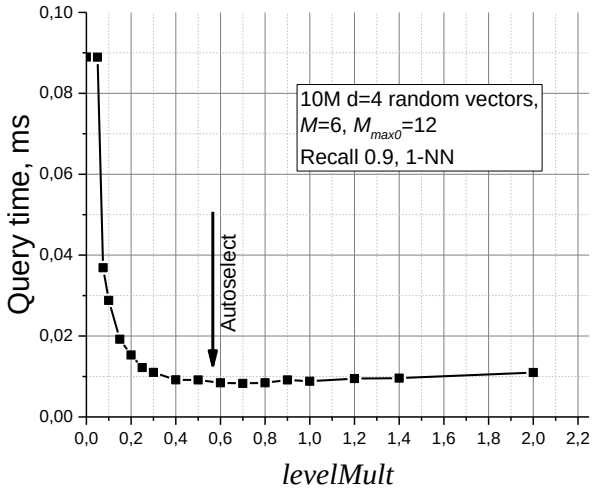

Fig. 3. Plots for query time vs *levelMult* parameter for 10M random vectors with d=4. The autoselected value for *levelMult* is shown by an arrow.

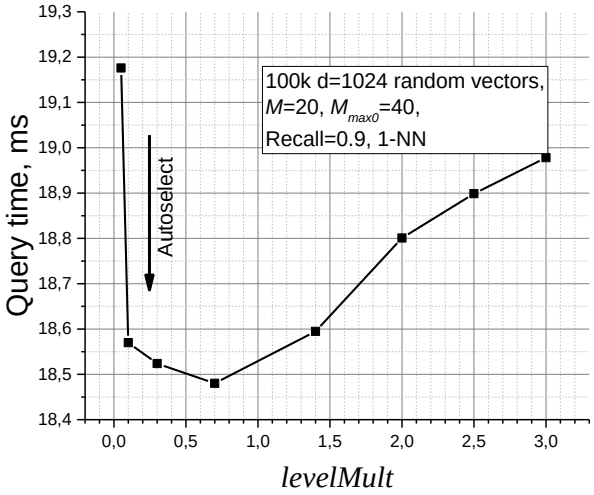

Fig. 4. Plots for query time vs *levelMult* parameter for 100k random vectors with d=1024. The autoselected value for *levelMult* is shown by an arrow.

Selection of the M_{max0} also has a strong influence on the search performance, especially in case of high quality (high recall) search. Simulations show that setting M_{max0} to M (this corresponds to k-NN graphs on each layer if the neighbors selection heuristic is not used) leads to a very strong performance penalty at high recall, while setting too large M_{max0} leads to excessive long-range links in the base (zero) layer. Simulations also suggest that $2 \cdot M$ is a good choice for M_{max0}. In fig. 6 there are presented results of 10-NN search performance for the 5M SIFT dataset depending on the M_{max0} parameter. The suggested value gives performance close to optimal at different recalls.

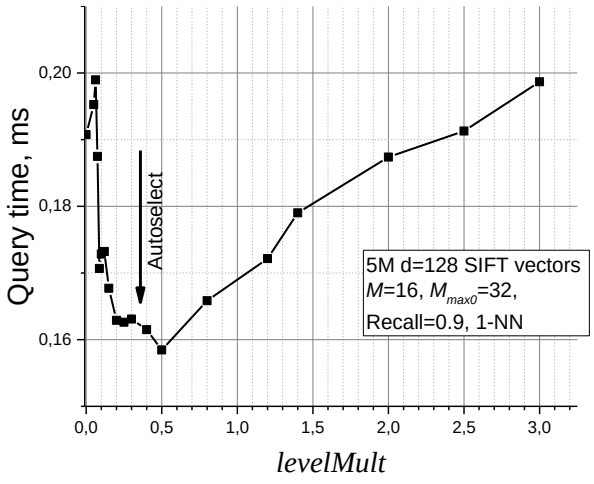

Fig. 5. Plots for query time vs *levelMult* parameter for 5M SIFT. The autoselected value for *levelMult* is shown by an arrow.

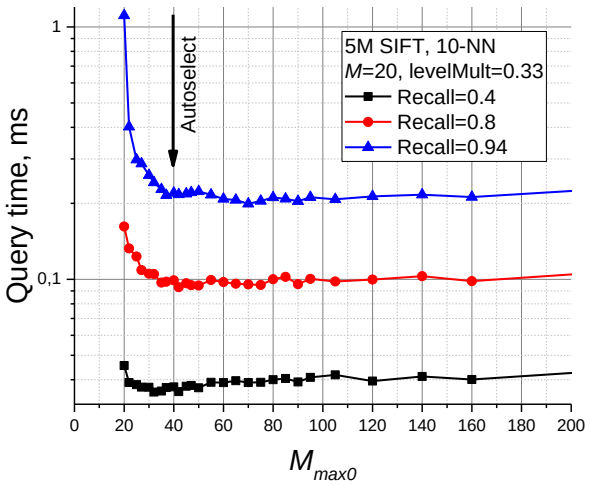

Fig. 6. Plots for query time vs M_{max0} parameter for 5M SIFT. The autoselected value for M_{max0} is shown by an arrow.

Selection of the *efConstruction* is straightforward. As it was suggested in [32] it has to be large enough to produce K-ANNS recall close to unity during the construction process (0.95 is enough for the most use-cases). And just like in [32] this parameter can possibly be auto-configured by using sample data.

In all of the considered cases, use of the heuristic for proximity graph neighbors selection (alg. 4) leads to a higher or equal search performance compared to the naïve connection to the nearest neighbors (alg. 3). The effect is very strong for low dimensional data, at high recall for mid-dimensional data and for the case of highly clustered data (ideologically discontinuity can be regarded as a local low dimensional feature), see the comparison in fig. 7. When using the closest neighbors as connections for the proximity graph, the Hierarchical NSW algorithm fails to achieve a high recall for clustered data because the search stucks on the clusters boundaries. While for the heuristic the introduced clustering leads to even higher performance. For uniform and very high dimensional data there is a little difference between the neighbors selecting methods, possibly due to the fact that in this case almost all of the nearest neighbors are selected by the heuristic.

The only meaningful construction parameter left for the user is the *M* parameter. A reasonable range of *M* is from 5 to 48. Simulations show that smaller *M* produces better results for lower recalls and/or lower dimensional data, while bigger *M* is better for high recall and/or high dimensional data (see fig. 8 for illustration).

The construction process can be easily parallelized with very few synchronization points. Building a high quality index (efConstruction=200, *M*=20) in multithreading regime for 1M SIFT data from [42] with 40 parallel threads on four Xeon E5-4650 v2 CPUs takes about 1-2 minutes at current implementation, which does not have many of optimizations used in the search algorithm.

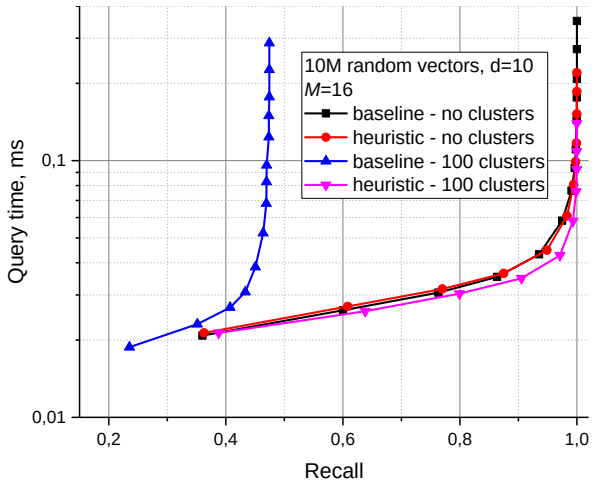

Fig. 7. Effect of the method of neighbor selections (baseline corresponds to alg. 3, heuristic to alg. 4) on clustered (100 random isolated clusters) and non-clustered d=10 random vector data.

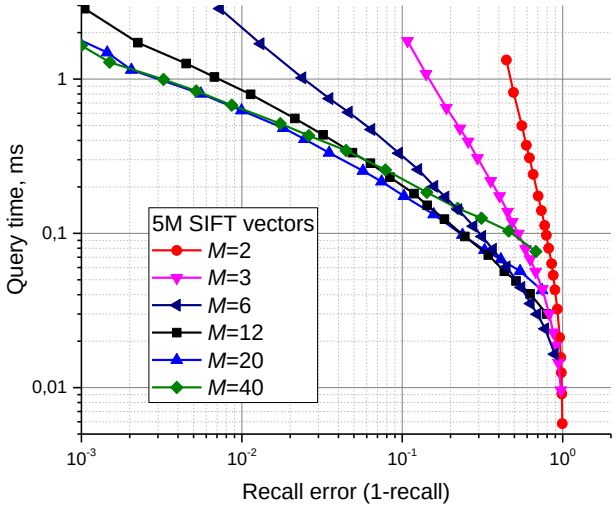

Fig. 8. Plots for query time vs recall for different parameters of *M* for the Hierarchical NSW on 5M SIFT dataset.

Comparison with basic NSW

The Hierarchical NSW algorithm is implemented on top of the Non Metric Space Library. Due to several limitations posed by the library, to achieve a better performance the implementation uses its own versions of distance functions together with C-style memory management at the

search phase. For the baseline NSW algorithm, we used the version from NMSLIB 1.1 which is slightly faster compared to the implementation tested in [34, 35] to demonstrate improvements in speed and algorithmic complexity.

Figure 9 presents a comparison of the Hierarchical NSW to the basic NSW algorithm for *d*=4 random hypercube data made on an i5-2400 Intel CPU (10-NN search). The *M* parameter was set to 6 for both algorithms. The Hierarchical NSW algorithm uses much less distance computations during a search on the dataset while the advance in actual performance is even higher (more than two orders of magnitude for high recall values) mostly due to better algorithm implementation.

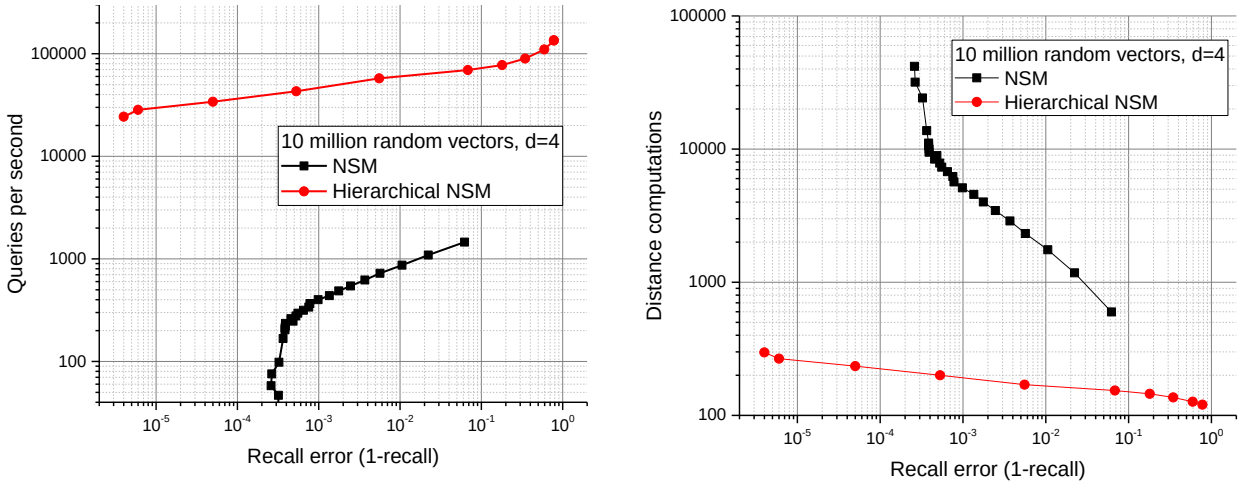

Fig. 9. Performance comparison of the NSW and the Hierarchical NSW on a 10 million 4-dimensional random vectors dataset.

The scalings of the algorithms on a *d*=8 random hypercube dataset for a 10-NN search with a fixed recall of 0.95 are presented in fig. 10. The *M* parameter was set to 6 for both algorithms. It clearly follows that the Hierarchical NSW algorithm has a complexity scaling for this setting not worse than logarithmic and outperforms the NSW algorithm at any dataset size.

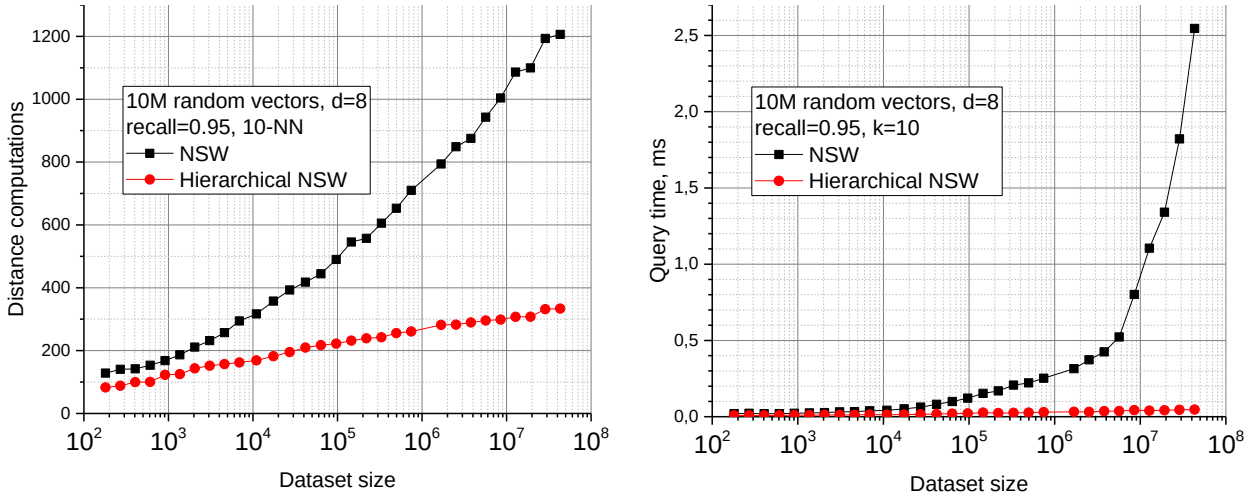

Fig. 10. Comparison between the NSW and the Hierarchical NSW in terms complexity scaling with the dataset size. The tests were performed on d=8 random vectors.

With a rise of the dataset dimensionality the scaling with the dataset size changes to a power-law at low size with a transition to logarithmic at some point. Scalings for random d=24 vectors (number of distance computations) and d=32 random vectors (query time in milliseconds) are presented in fig. 11, demonstrating the transition from power-law to logarithmic scaling at relatively high dimensions.

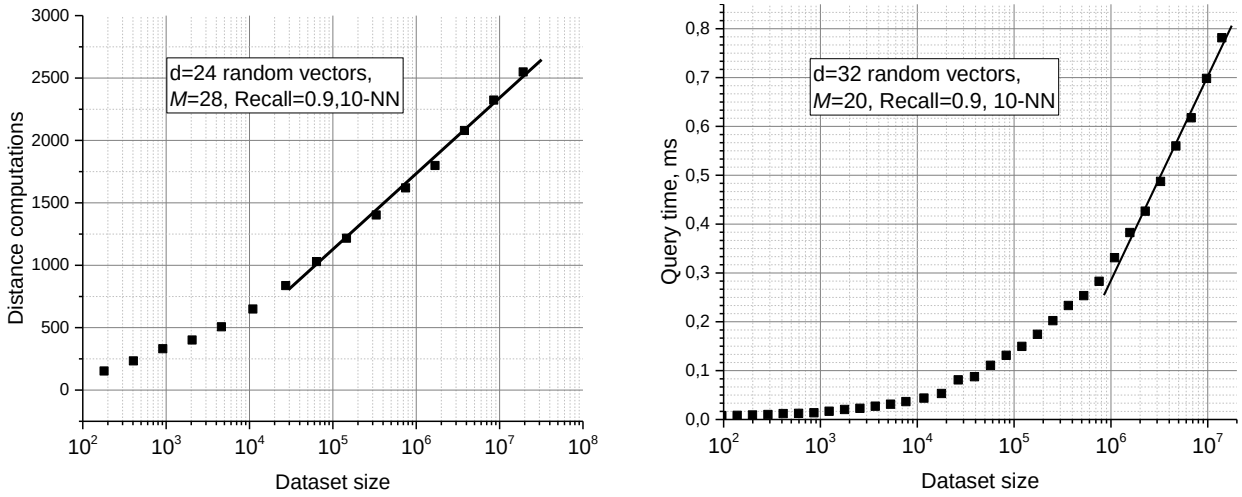

Fig. 11. Scalings for the number of distance computations (left, random d=24 vectors) and the query time (right, random d=32 vectors).

Comparison to rival methods

Comparing the performance of K-ANNS algorithms is a nontrivial task since the state-of-art is constantly changing as new algorithms and implementations are emerging. In this work we concentrated on comparison with the state-of-art algorithms that have open source implementations, which is beneficial for the users. An implementation of the Hierarchical NSW algorithm presented in this paper is also distributed as a part of the open source Non-Metric Space Library [43].

For comparison on vector data we used a popular K-ANNS benchmark from [44] as the base system of our comparison. The testing system utilizes python bindings of the algorithms and consequentially runs the K-ANN search for one thousand queries (randomly extracted from the initial dataset) with preset algorithm parameters producing an output containing recall and average time of a single search. The considered algorithms are:

1. FLANN 1.8.4 [7]. A popular library containing several algorithms, also used in OpenCV [45]. We used the built-in auto-tuning procedure with several reruns to infer the best parameters.
2. Annoy 02.02.2016 build [8]. A new but already popular algorithm based on random projection tree forest.
3. VP-tree. A general metric space algorithm implemented as a part of Non-Metric Space Library 1.1 [43].
4. FALCONN, version 1.2. A new efficient LSH algorithm for cosine similarity data [46].

The test parameters for VP-tree, FALCONN and Annoy were taken from [44]. The comparison was done on a 4-CPU Intel Xeon E5-4650 v2 system with 120 Gb of RAM under Debian OS. For

every algorithm we carefully chose the best results at every recall range to evaluate the best possible performance. All tests were done in a single thread regime. The Hierarchical NSW was compiled using the GCC 5.3 with -Ofast optimization flag.

Used datasets (vector data):

- SIFT dataset consisting of a million 128-dimensional vectors from [42].
- GloVe dataset consisting of 1.2 million 100-dimensional word embedding in vector space trained from tweets [47]. The cosine similarity distance was used as the distance function.
- CoPhIR dataset [48] consisting of 2 million 272-dimensional MPEG-7 features extracted from the images.
- Dataset of 30 million random points in unitary 4-dimensional cube with Euclid distance (to test the performance in a low dimensional case).
- Dataset containing 60 thousand handwritten digit images in 784-dimensional vector space from the MNIST database [49].

For all of the datasets except GloVe we used the L_2 distance. For GloVe we used the cosine similarity.

Results for the vector data are presented in fig. 12. For SIFT, GloVE and CoPhIR dataset the Hierarchical NSW algorithm clearly outperforms the rivals by a large margin. For low dimensional data (d=4) the Hierarchical NSW is slightly faster at high recall compared to the Annoy while strongly outperforming the other algorithms.

For comparison in a case of more general spaces with no constrains on the data, we have used the built-in testing system from Non Metric Space Library, repeating a subset of tests from the review [35]. The evaluated algorithms included the VP-tree, permutation techniques (NAPP and brutoforce filtering) [43, 50-52], the basic NSW algorithm and NNDescent-produced proximity graphs[21] (in pair with the NSW graph search algorithm). For every dataset the test includes the results of either NSW or NNDescent, depending on which structure performed better. No custom distance functions or special memory management were used in this case leading to some performance loss for the Hierarchical NSW.

Used datasets (Non-Metric data):

- Wiki-sparse dataset containing 4 million sparse 10^5-dimensional TF-IDF (term frequency–inverse document frequency) vectors (created via GENSIM [53]) with the sparse cosine distance.
- Wiki-128, Wiki-8 consisting of 2 million dense vectors of topic histograms created from sparse TF-IDF vectors of the wiki-sparse dataset (created via GENSIM [53]). Jensen–Shannon (JS) divergence was used as the distance function.
- ImageNet dataset contacting a million signatures extracted from LSVRC-2014 with SQFD (signature quadratic form) distance [54].

- 1M DNA (deoxyribonucleic acid) dataset sampled from the Human Genome 5. The employed distance function was the normalized Levenshtein distance.

Further details of the dataset origin and processing can be found in the original work [35]. The parameters of the rival methods were taken from the sample scripts of the Non Metric Space Library [43].

The results are presented in fig. 13. The Hierarchical NSW algorithm significantly improves the performance of the NSW algorithm and is a leader for any of the tested datasets. The strongest enhancement, almost by 3 orders of magnitude is observed for the dataset with the lowest dimensionality, the wiki-8 with JS-divergence. This is an important result that demonstrates the robustness of the Hierarchical NSW as for the original NSW this dataset was a stumbling block. Note that for the wiki-8 to nullify the effect of implementation results are presented for the distance computations number instead of the CPU time.

Indirect comparison to the competitive graph search techniques [13, 14] is presented in fig. 14. We were not able to make a direct comparison since there were no open source implementations of the rival algorithms [13, 14]. The timings of a 10-NN search on a one million SIFT dataset from [42] for different recall values were taken from [13], where a 3.4 GHz Intel CPU was used. For the test with the Hierarchical NSW we used a 3.4 GHz Intel i5-3570K CPU, which achieved FLANN performance close to the one from [13]. The latter is an additional correctness justification for such a comparison. The plot clearly shows that the Hierarchical NSW algorithm outperforms the algorithms [13, 14] for this setting, especially in case of high recall.

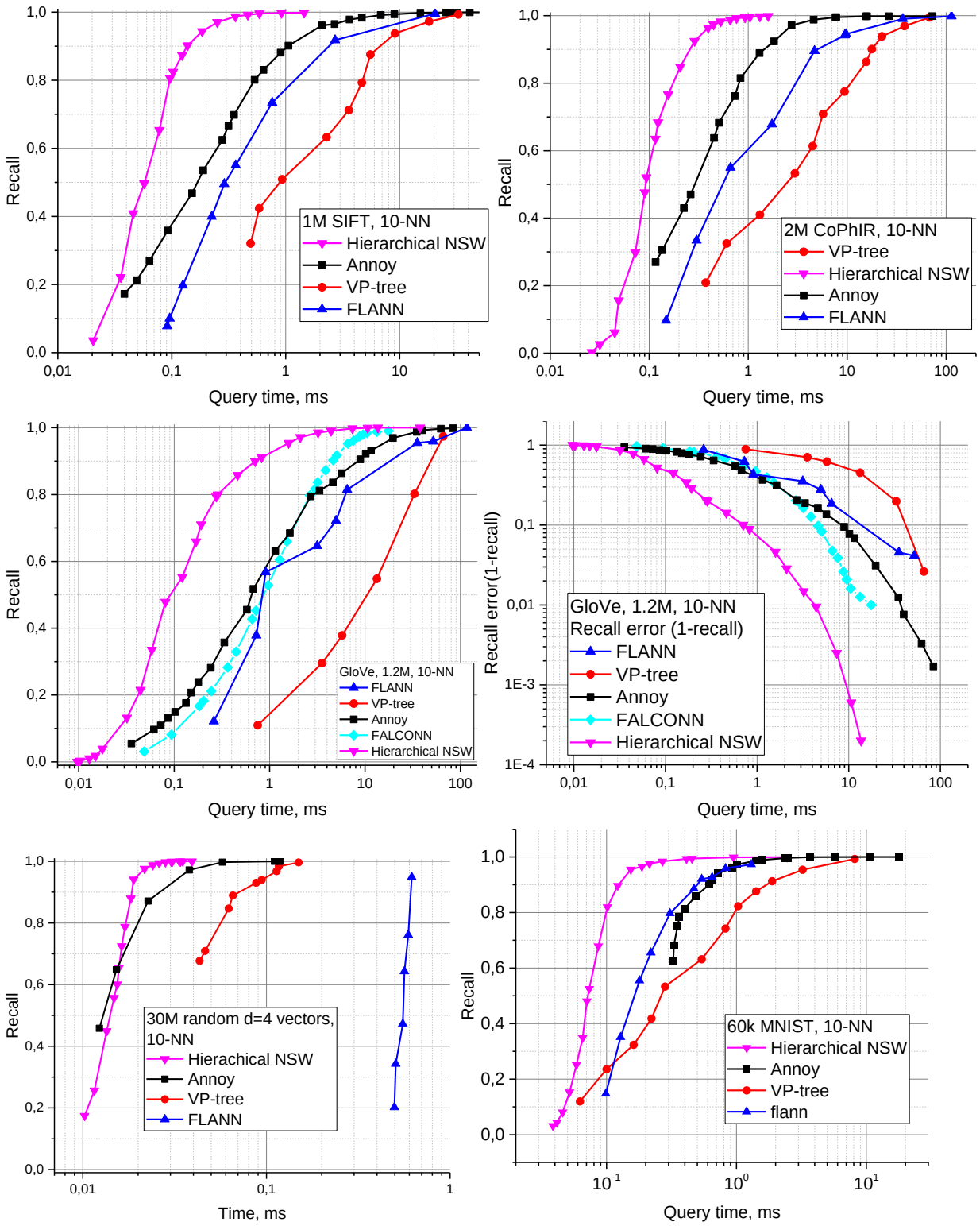

Fig. 12. Results of the comparison with open source implementations of K-ANNS algorithms on five datasets for 10-NN searches.

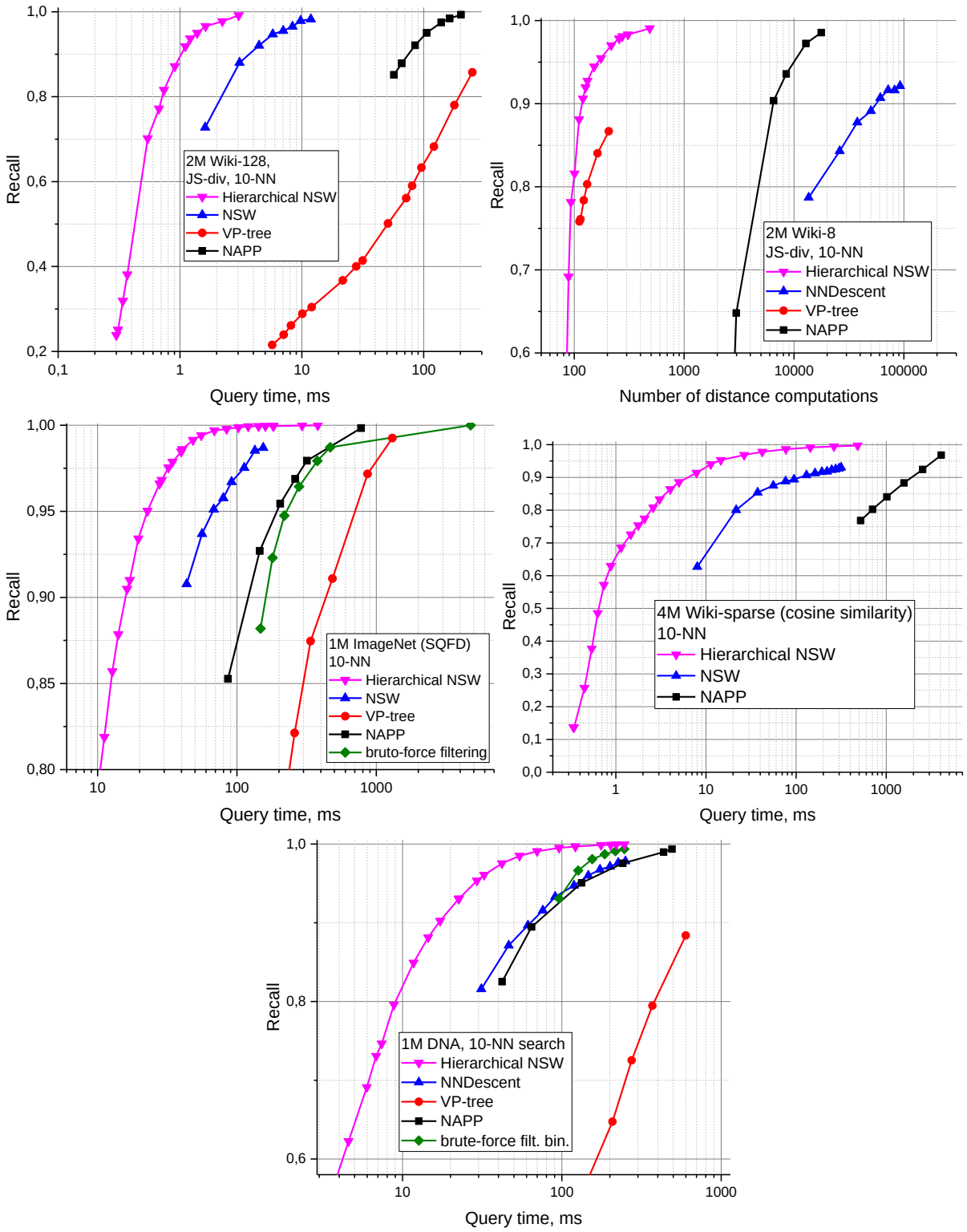

Fig. 13. Results of the comparison with general space K-ANNS algorithms from the Non Metric Space Library on five datasets for 10-NN searches.

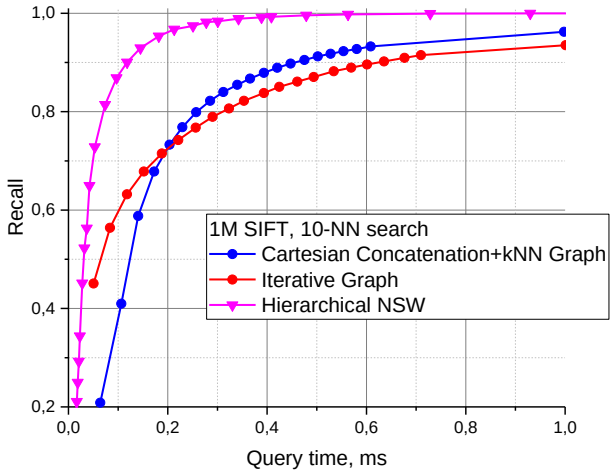

Fig. 14 Results of indirect comparison to the graph methods from [13, 14] on the 1M SIFT dataset from [42].

Complexity analysis

Search complexity

The complexity of a single search can be divided into complexity of two search phases. The first phase is used to coarsely find the closest element. The second phase corrects the error of the first phase and extracts the other *K*-1 nearest neighbors. During the first phase we make a constant number of steps on each layer (because the element *level* is uncorrelated with the distances) while the maximum layer number scales as log(N). Thus, the complexity of the first phase is logarithmical by the structure design. Intuitively, if the size of the dataset is large enough, the complexity of the second phase does not depend on the dataset size since we are extracting the graph only locally. Under this assumption the overall complexity scaling is log(N), in agreement with simulations on low dimensional datasets. For small-size large-dimensional datasets, the assumption of locality during the first stage may not hold and the search complexity is determined by the second stage, thus having at best sub-linear scalability.

Element removal complexity

If we remove an element from the structure we have to update the connections of its neighbors, which is roughly equivalent to the second phase of the search for each of the updated elements. Thus, under the same assumptions in the limit of large dataset element removal has a complexity independent of N.

Construction complexity

The construction complexity is defined by the search complexity since the insertion of an element is just a sequence of K-ANN-searches at different layers. The number of searches is equal to the element's *level*, which is close to unity on average. This means that the insertion cost is roughly equal to the search cost. And thus, for relatively low dimensional datasets the construction time scales as N·log(N).

Memory cost

The total memory cost for the structure is determined by the number of element connections and is about 2·*M*·number_of_bytes_per_link. If we limit the maximum total number of

elements by approximately four billions, we can use four-byte unsigned integers to store the connections. Tests suggest that typical close to optimal *M* values usually lie in a range between 6 and 48. This means that the typical memory requirements for the index (excluding the size of the data) are about 48-384 bytes per object.

Discussion

By using structure decomposition of navigable small world graphs together with the smart neighbor selection heuristics the proposed Hierarchical NSW approach overcomes several important problems of the basic NSW structure advancing the state of art in K-ANN search. The Hierarchical NSW offers an excellent performance and wins on a large variety of the datasets, surpassing the rivals by a large margin in case of high dimensional data. Even for the datasets where the previous version has lost by orders of magnitude, the Hierarchical NSW was able to come first. Hierarchical NSW can also be used as an efficient method for getting approximate K-NN and relative neighborhood graphs, which are byproducts of the structure construction.

Robustness of the approach is a strong feature which makes it very attractive for practical applications. The algorithm is applicable in generalized metric spaces and performs best on any of the datasets tested in this paper, thus eliminating the need for complicated selection of the best algorithm for a specific problem. We stress the importance of the algorithm's robustness since the data may have a complex structure with different effective dimensionality across the scales. For instance a high dimensional dataset can consist of large number of clusters arranged in a line, thus being low dimensional at large distance scale or, which is equivalent, at relatively small dataset size. In order to perform efficient search in such a dataset an approximate nearest neighbor algorithm has to work well for both cases of high and low dimensionality.

There are several ways to further increase the efficiency and applicability of the Hierarchical NSW approach. There is still one meaningful parameter left which strongly affects the construction of the index – the number of added connection per layer *M*. Potentially this parameter can be inferred directly by using different heuristics [5, 55]. It would also be interesting to compare the Hierarchical NSW at BIGANN dataset which consists from 1 billion SIFT feature vectors and is becoming a popular benchmark for the algorithms [13, 56-58].

One of the apparent shortcomings of the proposed approach compared to the NSW is the loss of the possibility of distributed search. The search in Hierarchical NSW structure always starts from the top layer, thus the structure cannot be made distributed by using the same techniques as described in [32] due to excessive load on the higher layer elements. Simple workarounds can be used to distribute the structure, such as partitioning the data across cluster nodes studied in [7], however in this case, the total parallel throughput of the system does not scale well with the number of computer nodes.

Still, there are other possible known ways to make this particular structure distributed. The Hierarchical NSW is ideologically very similar to the well-known one-dimensional exact search probabilistic skip list structure, and thus can use the same techniques to make the structure

distributed [40]. Potentially this can lead to even better distributed performance compared to the base NSW due to logarithmic scalability and ideally uniform load on the nodes.

Acknowledgements

We thank Leonid Boytsov for many helpful discussions, assistance with Non-Metric Space Library integration and comments on the manuscript. We also thank Valery Kalyagin for support of this work.

The reported study was funded by RFBR, according to the research project No. 16-31-60104 mol_a_dk.

References

[1] S. Cost and S. Salzberg, "A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features," *Machine Learning,* vol. 10, pp. 57-78, 1993.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision,* vol. 60, pp. 91-110, 2004.

[3] S. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by Latent Semantic Analysis," *J. Amer. Soc. Inform. Sci.,* vol. 41, pp. 391-407, 1990.

[4] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *SODA*, 1993, pp. 311-321.

[5] G. Navarro, "Searching in metric spaces by spatial approximation," *The VLDB Journal,* vol. 11, pp. 28-46, 2002.

[6] E. S. Tellez, G. Ruiz, and E. Chavez, "Singleton indexes for nearest neighbor search," *Information Systems,* 2016.

[7] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 36, pp. 2227-2240, 2014.

[8] (2016). *Annoy:Approximate Nearest Neighbors in C++/Python optimized for memory usage and loading/saving to disk*. Available: https://github.com/spotify/annoy

[9] M. E. Houle and M. Nett, "Rank-based similarity search: Reducing the dimensional dependence," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 37, pp. 136-150, 2015.

[10] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, "Practical and optimal LSH for angular distance," in *Advances in Neural Information Processing Systems*, 2015, pp. 1225-1233.

[11] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998, pp. 604-613.

[12] S. Arya and D. M. Mount, "Approximate Nearest Neighbor Queries in Fixed Dimensions," in *SODA*, 1993, pp. 271-280.

[13] J. Wang, J. Wang, G. Zeng, R. Gan, S. Li, and B. Guo, "Fast neighborhood graph search using cartesian concatenation," in *Multimedia Data Mining and Analytics*, ed: Springer, 2015, pp. 397-417.

[14] J. Wang and S. Li, "Query-driven iterated neighborhood graph search for large scale indexing," in *Proceedings of the 20th ACM international conference on Multimedia*, 2012, pp. 179-188.

[15] Z. Jiang, L. Xie, X. Deng, W. Xu, and J. Wang, "Fast Nearest Neighbor Search in the Hamming Space," in *MultiMedia Modeling*, 2016, pp. 325-336.

[16] E. Chávez and E. S. Tellez, "Navigating k-nearest neighbor graphs to solve nearest neighbor searches," in *Advances in Pattern Recognition*, ed: Springer, 2010, pp. 270-280.

[17] K. Aoyama, K. Saito, H. Sawada, and N. Ueda, "Fast approximate similarity search based on degree-reduced neighborhood graphs," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 1055-1063.

[18] G. Ruiz, E. Chávez, M. Graff, and E. S. Téllez, "Finding Near Neighbors Through Local Search," in *Similarity Search and Applications*, ed: Springer, 2015, pp. 103-109.

[19] R. Paredes, "Graphs for metric space searching," PhD thesis, University of Chile, Chile, 2008. Dept. of Computer Science Tech Report TR/DCC-2008-10. Available at http://www. dcc. uchile. cl/~ raparede/publ/08PhDthesis. pdf, 2008.
[20] C. C. Cartozo and P. De Los Rios, "Extended navigability of small world networks: exact results and new insights," *Physical review letters,* vol. 102, p. 238703, 2009.
[21] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proceedings of the 20th international conference on World wide web*, 2011, pp. 577-586.
[22] J. M. Kleinberg, "Navigation in a small world," *Nature,* vol. 406, pp. 845-845, 2000.
[23] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, 2000, pp. 163-170.
[24] J. Travers and S. Milgram, "An experimental study of the small world problem," *Sociometry,* pp. 425-443, 1969.
[25] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *Nature,* vol. 393, pp. 440-442, 1998.
[26] Y. Lifshits and S. Zhang, "Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design," in *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2009, pp. 318-326.
[27] A. Karbasi, S. Ioannidis, and L. Massoulie, "From Small-World Networks to Comparison-Based Search," *Information Theory, IEEE Transactions on,* vol. 61, pp. 3056-3074, 2015.
[28] O. Beaumont, A.-M. Kermarrec, and É. Rivière, "Peer to peer multidimensional overlays: Approximating complex structures," in *Principles of Distributed Systems*, ed: Springer, 2007, pp. 315-328.
[29] O. Beaumont, A.-M. Kermarrec, L. Marchal, and É. Rivière, "VoroNet: A scalable object network based on Voronoi tessellations," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1-10.
[30] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces," in *Similarity Search and Applications*, ed: Springer Berlin Heidelberg, 2012, pp. 132-147.
[31] A. Ponomarenko, Y. Malkov, A. Logvinov, and V. Krylov, "Approximate Nearest Neighbor Search Small World Approach," in *International Conference on Information and Communication Technologies & Applications*, Orlando, Florida, USA, 2011.
[32] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems,* vol. 45, pp. 61-68, 2014.
[33] Y. A. Malkov and A. Ponomarenko, "Growing homophilic networks are natural navigable small worlds," *arXiv preprint arXiv:1507.06529,* 2015.
[34] A. Ponomarenko, N. Avrelin, B. Naidan, and L. Boytsov, "Comparative Analysis of Data Structures for Approximate Nearest Neighbor Search," *In Proceedings of The Third International Conference on Data Analytics,* 2014.
[35] B. Naidan, L. Boytsov, and E. Nyberg, "Permutation search methods are efficient, yet faster search is possible," *VLDB Procedings,* vol. 8, pp. 1618-1629, 2015.
[36] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM,* vol. 33, pp. 668-676, 1990.
[37] M. Boguna, D. Krioukov, and K. C. Claffy, "Navigability of complex networks," *Nature Physics,* vol. 5, pp. 74-80, 2009.
[38] D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguná, "Hyperbolic geometry of complex networks," *Physical Review E,* vol. 82, p. 036106, 2010.
[39] A. Gulyás, J. J. Bíró, A. Kőrösi, G. Rétvári, and D. Krioukov, "Navigable networks as Nash equilibria of navigation games," *Nature Communications,* vol. 6, p. 7651, 2015.
[40] M. T. Goodrich, M. J. Nelson, and J. Z. Sun, "The rainbow skip graph: a fault-tolerant constant-degree distributed data structure," in *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, 2006, pp. 384-393.
[41] G. T. Toussaint, "The relative neighbourhood graph of a finite planar set," *Pattern recognition,* vol. 12, pp. 261-268, 1980.

[42] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 33, pp. 117-128, 2011.
[43] L. Boytsov and B. Naidan, "Engineering Efficient and Effective Non-metric Space Library," in *Similarity Search and Applications*, ed: Springer, 2013, pp. 280-293.
[44] *ANN benchmark*. Available: https://github.com/erikbern/ann-benchmarks
[45] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Real-time computer vision with OpenCV," *Communications of the ACM,* vol. 55, pp. 61-69, 2012.
[46] A. Andoni and I. Razenshteyn, "Optimal Data-Dependent Hashing for Approximate Near Neighbors," presented at the Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, Portland, Oregon, USA, 2015.
[47] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014),* vol. 12, pp. 1532-1543, 2014.
[48] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli*, et al.*, "CoPhIR: a test collection for content-based image retrieval," *arXiv preprint arXiv:0905.4627,* 2009.
[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, pp. 2278-2324, 1998.
[50] E. Chávez, M. Graff, G. Navarro, and E. Téllez, "Near neighbor searching with K nearest references," *Information Systems,* vol. 51, pp. 43-61, 2015.
[51] E. C. Gonzalez, K. Figueroa, and G. Navarro, "Effective proximity retrieval by ordering permutations," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 30, pp. 1647-1658, 2008.
[52] E. S. Tellez, E. Chávez, and G. Navarro, "Succinct nearest neighbor search," *Information Systems,* vol. 38, pp. 1019-1030, 2013.
[53] P. Sojka, "Software framework for topic modelling with large corpora," in *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010.
[54] C. Beecks, "Distance-based similarity models for content-based multimedia retrieval," Hochschulbibliothek der Rheinisch-Westfälischen Technischen Hochschule Aachen, 2013.
[55] A. Ponomarenko, "Query-Based Improvement Procedure and Self-Adaptive Graph Construction Algorithm for Approximate Nearest Neighbor Search," in *Similarity Search and Applications*, ed: Springer, 2015, pp. 314-319.
[56] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast exact search in hamming space with multi-index hashing," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 36, pp. 1107-1119, 2014.
[57] W. Zhou, C. Yuan, R. Gu, and Y. Huang, "Large Scale Nearest Neighbors Search Based on Neighborhood Graph," in *Advanced Cloud and Big Data (CBD), 2013 International Conference on*, 2013, pp. 181-186.
[58] A. Babenko and V. Lempitsky, "The inverted multi-index," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, pp. 3069-3076.