Comparative Analysis of Data Structures for Approximate Nearest Neighbor Search

Alexander Ponomarenko, Nikita Avrelin

National Research University
Higher School of Economics
Nizhny Novgorod, Russia
Email: aponomarenko@hse.ru

Bilegsaikhan Naidan

Department of Computer and
Information Science
Norwegian University of
Science and Technology,
Trondheim, Norway
Email: bileg@idi.ntnu.no

Leonid Boytsov

Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA, USA
Email: srchvrs@cs.cmu.edu

***Abstract*—Similarity searching has a vast range of applications in various fields of computer science. Many methods have been proposed for exact search, but they all suffer from the curse of dimensionality and are, thus, not applicable to high dimensional spaces. Approximate search methods are considerably more efficient in high dimensional spaces. Unfortunately, there are few theoretical results regarding the complexity of these methods and there are no comprehensive empirical evaluations, especially for non-metric spaces. To fill this gap, we present an empirical analysis of data structures for approximate nearest neighbor search in high dimensional spaces. We provide a comparison with recently published algorithms on several data sets. Our results show that small world approaches provide some of the best tradeoffs between efficiency and effectiveness in both metric and non-metric spaces.**

Keywords–nearest neighbor search; metric space; non-metric search, approximate search; small world graphs

I. Introduction

Similarity searching is a fundamental topic of computer science, which naturally appears in the different fields such as pattern recognition [1], computer vision [2], collaborative filtering [3], and so on. The goal of a similarity search is to find points from a data set that are sufficiently similar to a search pattern q, also known as a *query*.

The similarity of two data points (x and y) is computed using a distance function $d(x, y)$. The smaller the value of the distance function, the more similar (close) are the points. When d is (1) a symmetric non-negative function; (2) satisfies the triangle inequality; (3) and is equal to zero only for identical points, it is called a *metric*. If d violates any of these properties, then it is called non-metric.

In this paper, we focus on the nearest-neighbor search, where one needs to find the points whose distance from the query is the smallest among all points in the data set. A k-nearest-neighbor (k-NN) search is a generalization of the nearest-neighbors search. This generalization aims to find k points closest to the query, i.e., its k nearest neighbors. In an exact version of the problem, one is required to find all k nearest neighbors. Many exact nearest-neighbor search methods were proposed. Yet, they work well only in a low dimensional metric space. (A dimensionality of a vector space is simply a number of coordinates necessary to represent a vector: This notion can be generalized to spaces without coordinates [4]).

Experiments showed that exact methods can rarely outperform the sequential scan when dimensionality exceeds ten [5]. In a literature this problem has been dubbed as "the curse of dimensionality". Using approximate search methods, which do not guarantee retrieval of all neighbors, allows one to lift the "curse".

Non-metric spaces represent another domain where most of the proposed methods are not applicable. Compared to metric spaces, it is much harder to design exact methods for arbitrary non-metric spaces, most importantly, because the triangle inequality is violated. Whenever exact search methods for non-metric spaces do exist, they also seem to suffer from the curse of dimensionality [6][7].

Thus, the goal of approximation is two-fold: It allows us to (1) reduce the search time while obtaining reasonably accurate results; (2) answer queries for data points drawn from non-metric spaces, where properties such as the triangle inequality do not hold.

Approximate search methods can be much more efficient than exact ones, but this additional efficiency comes at the expense of a reduced search accuracy. More specifically, k points obtained by an approximate nearest-neighbor search methods might not be the k closest points to the query point. One common measure of the search accuracy is a *recall*. The recall is equal to the fraction of nearest neighbors returned by a search method.

There is a lack of evaluations that compare approximate search methods for both metric and non-metric spaces. Thus, we carry out this experimental comparison by testing several efficient benchmarks on metric and non-metric data sets. These benchmarks are compared against recently proposed method based on navigable small worlds graphs [8][9]. We measure efficiency and effectiveness for complete data sets as well as study how these characteristics depend on the number of data points.

There are several surveys covering exact nearest neighbor and range search in metric spaces, in particular, a work by Chávez et al. [4]. Many classic exact methods for metric spaces are implemented in the *Metric Spaces Library* [10]. Skopal and Bustos [11] surveyed search methods for non-metric spaces.

A *Non-Metric Space Library* is an evaluation toolkit and a similarity search library that contains efficient benchmarks for both metric (e.g., Euclidean) and non-metric spaces [12][7]. In

particular, the library has an approximate version of the VP-tree that was shown to be competitive [7] against the multi-probe locality sensitive hashing [13] in the Euclidean (i.e., metric) space, as well as against the bbtree [6] in the case of KL-divergence [14] and Itakura-Saito distance [15] (which are both non-metric distance functions).

The paper is organized as follows: In Section II, we describe the selected benchmarks (implemented in the *Non-Metric Space Library*); In Section III, we present evaluation results; Section IV concludes the paper.

II. Implemented Methods

A. Vantage Point Tree

The Vantage Point Tree is a hierarchical space partitioning method which uses the triangle inequality to discard partitions that cannot contain nearest neighbors [16][17]. The classic version of this method supports only an exact search in metric spaces. Yet, by stretching, i.e., relaxing, the triangle inequality [18], it is possible to support approximate nearest neighbor searching in both metric and non-metric spaces [7].

Optimal stretching coefficients were found using a simple grid search. We indexed a small database sample, executed the 10-NN search for various values of stretching coefficients and measured performance. Then, we selected coefficients resulting in the fastest search at a given recall value.

B. Permutation Methods

Permutation methods are dimensionality-reduction approaches, where each point is represented by a low-dimensional integer-valued vector called a *permutation*. To obtain the permutation, we need to select m pivots π_i (e.g., by randomly sampling data points). Then, for every point x we arrange pivots π_i in the order of increasing distance $d(\pi_i, x)$. An i-th element of the permutation vector is simply a position of the pivot i in this arrangement. For the pivot closest to the data point the value the vector element is one, while for the most distance pivot the value is m. Some of the first permutation methods were independently proposed by Chavez et al. [19], as well as by Amato and Savino [20].

A basic version of this method randomly samples pivots from the data set. Then, it computes permutations for every data point and stores permutations as an array. During the search, it scans permutations of the data points sequentially and computes a distance (usually Euclidean) between the query permutation and each retrieved permutation (representing a data point). This step generates a list of candidate data points.

Afterwards, the search method sorts candidate data points based on the distances between their permutations and the permutation of the query. A fraction of points which represent the smallest distances are compared directly against the query, using the original distance function d. The underlying idea is that while computation of the original distance can be expensive, comparing low-dimensional integer-valued vectors, i.e., permutations, is an inexpensive operation.

This basic method was improved in several ways. First of all, we need only a small fraction of the data points that have permutation closest to the query permutation. Thus, computing the complete ordering of permutations is wasteful. Instead, one can resort to incremental sorting [19]. Second, one can index permutations rather than searching them sequentially: It is possible to employ a permutation prefix tree [21], an inverted index [20], or an index designed for metric spaces, e.g., a VP-tree [22].

More recently, it was proposed to index pivot neighborhoods: For each data point, we select $numPrefix \ll m$ pivots (out of m existing pivots) that are closest to the data point. Then, we associate these $numPrefix$ closest pivots with the data point via an inverted file [23]. One can hope that for similar points two pivot neighborhoods will have a non-zero intersection.

To exploit this observation, our implementation of the pivot neighborhood indexing method retrieves all points that share at least $minTimes$ nearest neighbor pivots (using an inverted file). Then, these candidates points are compared directly against the query.

Preliminary experiments showed that, depending on a data set, one of the following permutations method was the most efficient: the basic permutation method with incremental sorting, the approximate version of VP-tree index built over a set of permutations, or a pivot neighborhood index.

C. Small World

A small world method is a variant of a navigable small world graph data structure [9]. The small world graph represents an approximation of the Delaunay triangulation [24] and its respective Voronoi partitioning [24]. In a small world graph, data points are graph nodes and edges connect close data points. Ideally, it should be possible to find nearest neighbors of any data point by following just a few graph edges.

The nearest neighbor search algorithm is, thus, a greedy search procedure that carries out several sub-searches. A sub-search starts at a random node and proceeds to expanding the set of traversed nodes by following neighboring links. The sub-search stops when we cannot find points that are closer than already found M nearest points (M is a search parameter).

Indexing is a bottom-up procedure that relies on the previously described greedy search procedure. We add points, one by one. For each data point, we find N closest points using an already constructed index. Then, we create an edge between a new graph node (representing a new point) and nodes that represent N closest points found by the greedy search. Note that the greedy search is only approximate and does not necessarily return all N nearest neighbors. Empirically, it was shown that this method often creates a navigable small world graph, where most nodes are separated by only a few edges. In that, the number of edges is typically logarithmic in the size of the data set [8].

The indexing algorithm is rather expensive and we accelerate it by running parallel searches in multiple threads. The graph updates are synchronized: If a thread needs to add edges to a node or obtain the list of node edges, it first locks a node-specific mutex. Because, different threads rarely update the same node, such synchronization creates little contention and, consequently, our parallelization approach is efficient. It is also necessary to synchronize updates for the list of graph nodes, but this operation takes little time compared to searching for N neighboring points.

D. Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is a class of methods employing hash functions that tend to have the same hash

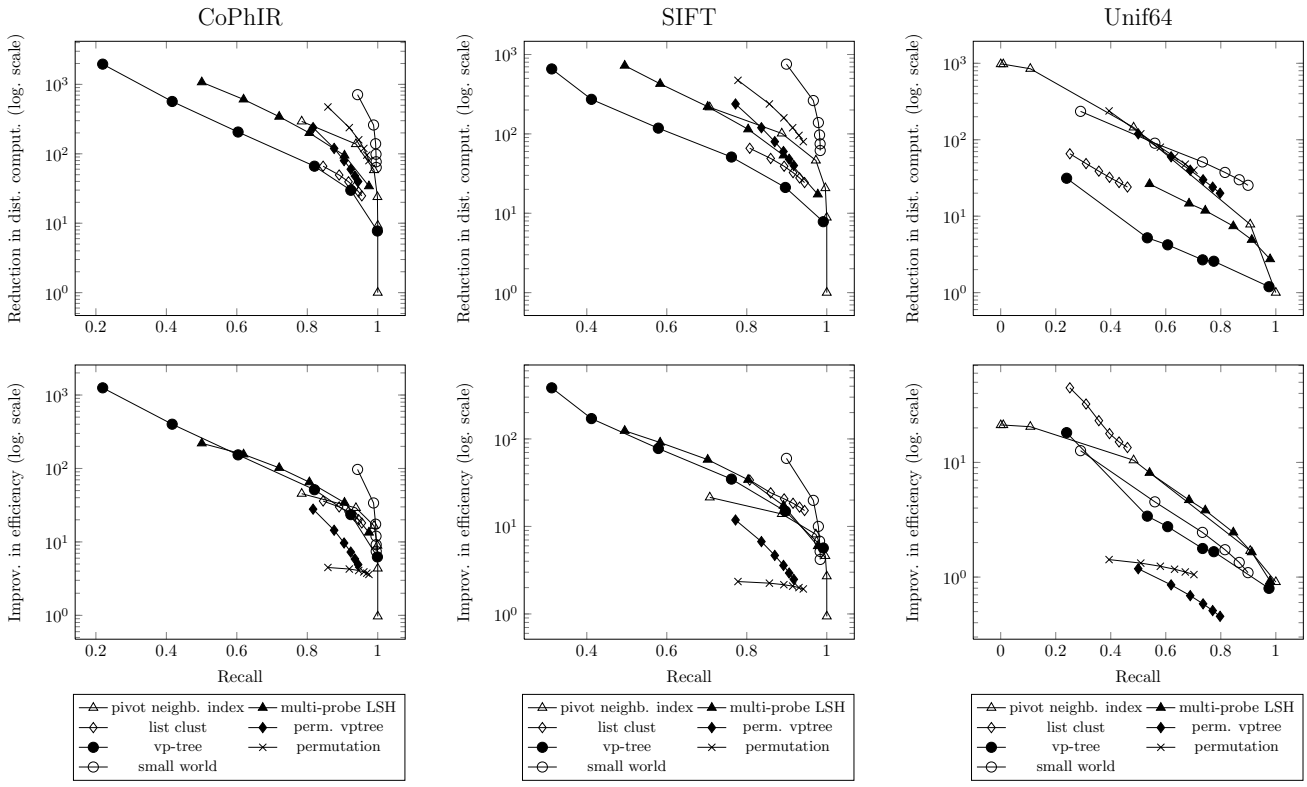

Figure 1. Performance of a 10-NN search for L_2 : plots in the same column correspond to the same data set

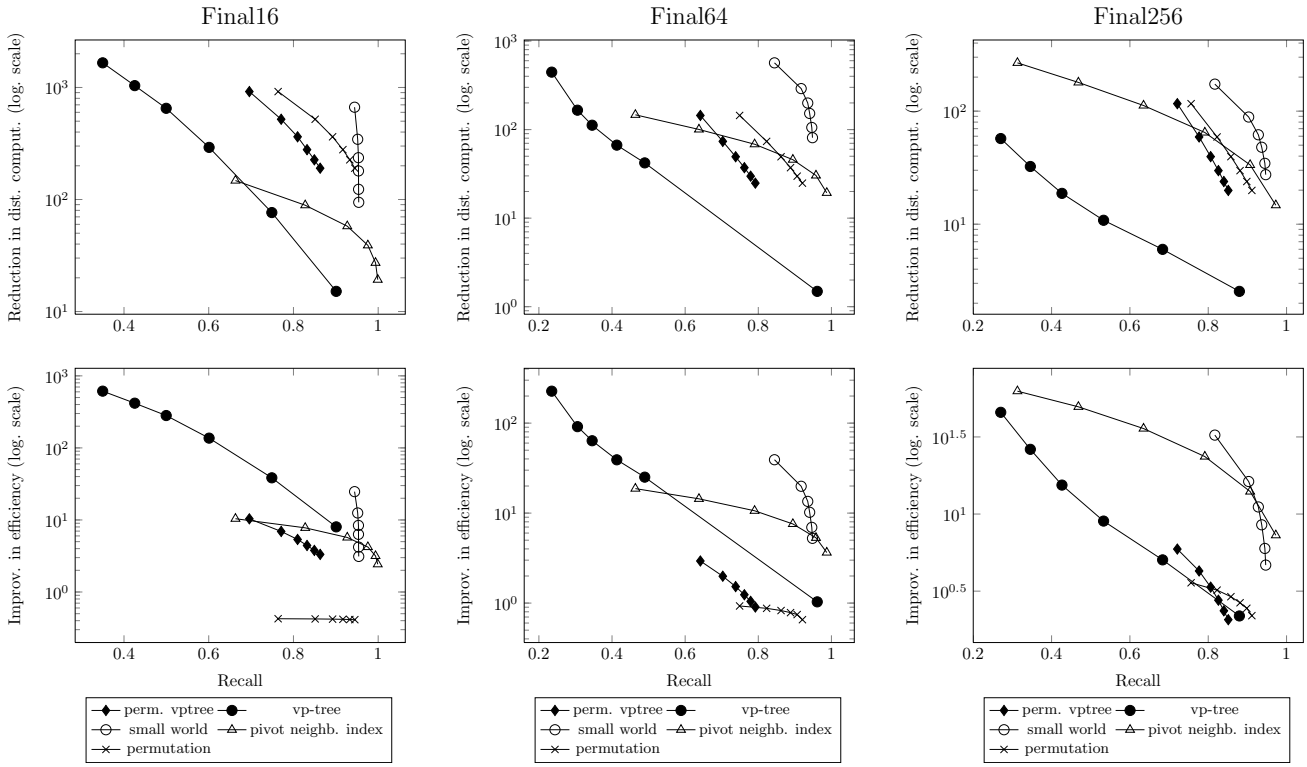

Figure 2. Performance of a 10-NN search for the KL-divergence: plots in the same column correspond to the same data set

values for close points and different hash values for distant points. It is a probabilistic method in which the probability of having the same hash value is a monotonically decreasing function of the distance between two points (that we compare). A hash function that possesses this property is called locality sensitive. The first LSH method was proposed by Indyk and Motwani in [25].

One drawback of this method is that it is hard to design a locality sensitive hash function for an arbitrary non-metric space. Yet, it is a very strong benchmark in the case of the Euclidean distance. This is why we used it in our experiments. More specifically, we employed a memory-efficient multi-probe LSH due to Dong et al. [13], which is implemented as a part of the LSHKIT library [13].

E. List of Clusters

The list of clusters [26] is an exact search method for metric spaces, which relies on flat (i.e., non-hierarchical) clustering. Clusters are created sequentially starting by selecting an arbitrary cluster center. Then, close points are assigned to the cluster and the clustering procedure is applied to the remaining points. Closeness is defined either in terms of the maximum distance R from the cluster center (points with distances larger than R are not included into the cluster) or in terms of the number of points N closest to the cluster center. In our work, we rely on the latter strategy and select cluster centers randomly.

The search algorithm iterates over the constructed list of clusters and checks if answers can potentially belong to the currently selected cluster (using the triangle inequality). If the cluster can contain an answer, each cluster element is compared directly against the query. Next, we use the triangle inequality to verify if answers can be outside the current cluster. If this is not possible, the search is terminated.

We modified this exact algorithm by introducing an early termination condition. The clusters are visited in the order of increasing distance from the query to a cluster center. The search process stops after vising a certain number (a method parameter) of clusters.

III. Experiments

A. Data Sets

Overall, three different distance functions were used:

- The Euclidean metric distance (L_2);
- The non-metric distance function KL-divergence [14]: $d(x,y) = \sum x_i \log \frac{x_i}{y_i}$;
- The non-metric cosine similarity: $1 - \frac{\sum x_i y_i}{\sqrt{\sum x_i^2}\sqrt{\sum y_i^2}}$.

In what follows, we summarize employed data sets and respective distance functions.

1) CoPhIR (L_2): data set is the collection of 208-dimensional vectors extracted from images in MPEG7 format [27]. Vectors are composed of five different MPEG7 features.

2) SIFT (L_2): is a part of the TexMex data set collection [28]. It has one million 128-dimensional vectors. Each vector corresponds to descriptor extracted from image data using Scale Invariant Feature Transformation (SIFT) [29].

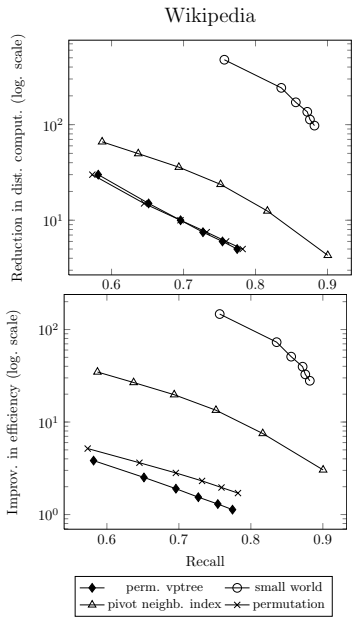

Figure 3. Performance of a 10-NN search for the 3.2 million points from the Wikipedia sparse-vector data set.

3) Wikipedia (cosine similarity): is a data set that contains 3.2 million vectors represented in a sparse format. Each vector corresponds to the TF-IDF vector of the Wikipedia page extracted using the *gensim* library [30]. This set has an extremely high dimensionality (more than 100 thousand elements). Yet, the vectors are sparse: On average only about 600 elements are non-zero.

4) Unif64 (L_2): is a synthetic data set of 64-dimensional vectors. The vectors were generated randomly, independently and uniformly in the unit hypercube.

5) Final16, Final64, and Final256 (KL-divergence): are sets of 0.5 million topic histograms generated using the Latent Dirichlet Allocation (LDA) [31]. The numeric suffix of a data set name indicates the dimensionality, which is also equal to the number of LDA topics. This data set was created by Cayton [6].

B. Evaluation

Experiments were carried out on an Linux Intel Xeon server (3.60 GHz, 32GB memory) in a single threaded mode using the *Non-Metric Space Library* as an evaluation toolkit [12]. The code was written in C++ and compiled using GNU C++ 4.7 (-Ofast optimization option).

We relied on optimized distance functions implemented with a help of SSE 4.2 SIMD instructions. In the case of the KL-divergence, further speed up are achieved by precomputing logarithms of vector elements at index time [7]. An implementation of the cosine similarity used the all-against-all comparison instruction `_mm_cmpistrm`. This implementation (inspired by the set intersection algorithm of Schlegel et al. [32]) is about 2.5 times faster than a pure C++ implementation based on the merge-sort approach.

We randomly divided a data set into two subsets. A smaller subset contained only 1000 points and was used as a query set. The remaining points were indexed. After indexing, we

evaluated performance of a 10-NN search (all indexes were memory-resident). To this end, a search was repeated several times to produce results at different recall values (method parameters were selected manually). This procedure was repeated five times and evaluation results were averaged over five data set splits. The variance in query times was low and, hence, we report only point estimates.

Most methods were evaluated on all data sets. Yet, the multi-probe LSH and the list of clusters were used only with the Euclidean distance. The VP-tree was not used for Wikipedia, because, due to extremely high dimensionality of this data set, the VP-tree was only marginally better than sequential searching.

Evaluation results for the Euclidean distance and for the KL-divergence are presented in Figures 1 and 2, respectively. The graphs in the first row show reduction in the number of distance computations (compared to sequential, i.e., brute force searching without an index) against the search accuracy measured by the recall (equal to the fraction of nearest neighbors returned by a method). An exact method has an ideal recall of one, which means that the exact method finds all nearest neighbors. The graphs in the second row show the overall improvement in efficiency (again, compared to sequential searching). Note that the permutation method with incremental sorting is denoted as simply *permutation* in the plots' legends.

As can be seen from the Figures 1 and 2, the small world algorithm provides the best tradeoffs between reduction in the number of distance computations and effectiveness for all three data sets. Consider, for example, the CoPhIR data in Figure 1. At the recall value of 0.9, the improvement in the number of distance computations for the small world is 1000. For all the other methods, the improvement in the number of distance computations is less than 100. To obtain a comparable improvement in the number of distance computations for, e.g., LSH, one has to tolerate the recall as low as 0.4.

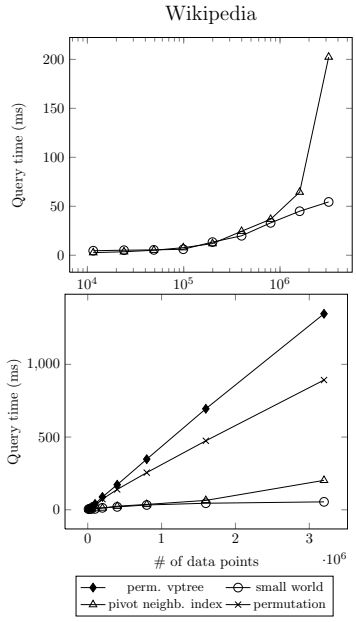

Figure 4. Dependence of the query time on the data set size. The upper plot represents only the small world method and the pivot neighborhood index. Query times are computed at roughly fixed recall values: 0.9 for the small world and 0.8 for other methods.

Typically, the larger is the reduction in the number of distance computations performed during the search, the more efficient is the method. Yet, for inexpensive distance functions (such as the Euclidean distance), the reduction in the number of distance computations does not directly translate into the overall improvement in performance. Consider the Unif64 data in Figure 1 and Final256 data in Figure 2: Despite that the small world method performs fewer distance computations than other methods in almost all the cases, the bookkeeping cost related to traversal of the small world graph can be high. As a result, the pivot neighborhood index or the multi-probe LSH are sometimes more efficient (at same recall values).

Note that both the small world and the pivot neighborhood index work well in the case of the KL-divergence (see Figure 2). For all three KL-divergence data sets, it is possible to achieve a ten-fold speed up over sequential searching while keeping the recall as high as 0.9.

The results for the complete Wikipedia data set are presented in Figure 3. The upper graph shows the reduction in the number of distance computations against the recall, while the lower graph shows the improvement in efficiency against the recall. Despite our optimized SIMD implementation of the cosine similarity is 2.5 times faster than the pure C++ version, it is still quite expensive to compute the scalar product between sparse TF-IDF vectors. As a result, in most cases, reduction in the number of distance computations maps well to the overall improvement in efficiency.

Note that the small world method is substantially better than the other methods. For example, at the recall value 0.87 it is about 40 times faster than sequential searching. The next fastest method (the pivot neighborhood index) achieves this speedup only at a significantly lower recall value of 0.6.

To measure how performance depends on the size of a data set, we also obtained results for Wikipedia subsets whose sizes varied from 12.5 thousand to 3.2 million data points. For each step and for each method we ran a search procedure several times with different options to measure performance at various values of recall (again, we manually tweaked method parameters to achieve different recall values). In the case of the small world, we selected runs that resulted in recall values closest to 0.9, while for other methods we selected runs with recall closest to 0.8. The results are presented in Figure 4, where the lower plot includes all the tested method, while the upper plot includes only the small world method and the pivot neighborhood index.

As can been seen from the Figure 4, all permutation methods have a near linear dependency for the number of distance computations on the number of data points. For the small world method, the dependence is close to being logarithmic (see the upper plot in Figure 4). In that, the small world method exhibits a greater reduction in the number of distance computations at higher recall values (0.9 vs 0.8). Compared to other permutation methods, performance of the pivot neighborhood scales much better as the number of data points increases. Yet, this method is still substantially slower and/or less accurate than the small world method.

IV. Conclusion and Future Work

We carried out an extensive experimental comparison using several large data sets. Our experiments involve both metric and non-metric distance functions including the challenging KL-divergence: The KL-divergence is not symmetric and does not satisfy the triangle inequality. To ease reproduction of results, we make our code publicly available, as a part of the open-source Non-Metric Space Library [12]. All data sets except CoPhIR are publicly available as well.

Our experiments show that the small world method outperforms the other methods for most recall values. Experiments with the sparse-vector Wikipedia data set demonstrate that the small world method has a near logarithmic dependence for the number of distance computation on the number of data points, which confirms previous findings [8]. That is, despite dealing with an extremely high-dimensional data set, it is possible to obtain accurate results (recall 0.9) quickly. We hypothesize that small world graph approaches are some of the most efficient high-accuracy methods in both metric and non-metric spaces.

The small world method is almost always superior in terms of the reduction in the number of distance computations. In the case of inexpensive distance functions, this does not always result in better overall performance, because traversing the small world graph can be expensive (note that the small world method is still the fastest in most cases). In the future, we plan to design a more efficient version of the small world method.

Acknowledgment

The first author is partially supported by LATNA Laboratory, NRU HSE, RF government grant, ag. 11.G34.31.0057.

References

[1] T. Cover and P. Hart, "Nearest neighbor pattern classification," Information Theory, IEEE Transactions on, vol. 13, no. 1, 1967, pp. 21–27.

[2] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic et al., "Query by image and video content: The qbic system," Computer, vol. 28, no. 9, 1995, pp. 23–32.

[3] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in Proceedings of the 10th international conference on World Wide Web. ACM, 2001, pp. 285–295.

[4] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín, "Searching in metric spaces," ACM computing surveys (CSUR), vol. 33, no. 3, 2001, pp. 273–321.

[5] R. Weber, H. J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in Proceedings of the 24th International Conference on Very Large Data Bases. Morgan Kaufmann, August 1998, pp. 194–205.

[6] L. Cayton, "Fast nearest neighbor retrieval for bregman divergences," in ICML, 2008, pp. 112–119.

[7] L. Boytsov and B. Naidan, "Learning to prune in metric and non-metric spaces," in NIPS, 2013, pp. 1574–1582.

[8] 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. Springer, 2012, pp. 132–147.

[9] ——, "Approximate nearest neighbor algorithm based on navigable small world graphs," Information Systems, vol. 45, 2014, pp. 61–68.

[10] K. Figueroa, G. Navarro, and E. Chávez, "Metric spaces library," 2007, available at http://www.sisap.org/Metric_Space_Library.html [retrieved: Jun 2014].

[11] T. Skopal and B. Bustos, "On nonmetric similarity search problems in complex domains," ACM Comput. Surv., vol. 43, no. 4, Oct. 2011, pp. 34:1–34:50.

[12] L. Boytsov and B. Naidan, "Engineering efficient and effective non-metric space library," in SISAP, 2013, pp. 280–293, available at https://github.com/searchivarius/NonMetricSpaceLib [retrieved: Jun 2014].

[13] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, "Modeling lsh for performance tuning," in Proceedings of the 17th ACM conference on Information and knowledge management, ser. CIKM '08. New York, NY, USA: ACM, 2008, pp. 669–678.

[14] S. Kullback and R. A. Leibler, "On information and sufficiency," The Annals of Mathematical Statistics, vol. 22, no. 1, 03 1951, pp. 79–86. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177729694

[15] F. Itakura and S. Saito, "Analysis synthesis telephony based on the maximum likelihood method," in Proceedings of the 6th International Congress on Acoustics, vol. 17. pp. C17–C20, 1968, pp. C17–C20.

[16] J. K. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," Information processing letters, vol. 40, no. 4, 1991, pp. 175–179.

[17] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, ser. SODA '93. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1993, pp. 311–321.

[18] E. Chávez and G. Navarro, "Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces," Information Processing Letters, vol. 85, no. 1, 2003, pp. 39–46.

[19] E. C. Gonzalez, K. Figueroa, and G. Navarro, "Effective proximity retrieval by ordering permutations," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 30, no. 9, 2008, pp. 1647–1658.

[20] G. Amato and P. Savino, "Approximate similarity search in metric spaces using inverted files," in Proceedings of the 3rd international conference on Scalable information systems. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 28.

[21] A. Esuli, "Use of permutation prefixes for efficient and scalable approximate similarity search," Inf. Process. Manage., vol. 48, no. 5, Sep. 2012, pp. 889–902.

[22] K. Figueroa and K. Fredriksson, "Speeding up permutation based indexing with indexing," in Proceedings of the 2009 Second International Workshop on Similarity Search and Applications, ser. SISAP '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 107–114.

[23] E. S. Tellez, E. Chávez, and G. Navarro, "Succinct nearest neighbor search," Information Systems, vol. 38, no. 7, 2013, pp. 1019–1030.

[24] H. Samet, Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc., 2005.

[25] 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. ACM, 1998, pp. 604–613.

[26] E. Chávez and G. Navarro, "A compact space decomposition for effective metric indexing," Pattern Recognition Letters, vol. 26, no. 9, 2005, pp. 1363–1376.

[27] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli, and F. Rabitti, "Cophir: a test collection for content-based image retrieval," arXiv preprint arXiv:0905.4627, 2009.

[28] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 33, no. 1, 2011, pp. 117–128.

[29] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," International journal of computer vision, vol. 60, no. 2, 2004, pp. 91–110.

[30] R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora," in Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks. Valletta, Malta: University of Malta, 2010, pp. 46–50.

[31] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," the Journal of machine Learning research, vol. 3, 2003, pp. 993–1022.

[32] B. Schlegel, T. Willhalm, and W. Lehner, "Fast sorted-set intersection using simd instructions." in ADMS@ VLDB, 2011, pp. 1–8.