The Role of Local Intrinsic Dimensionality in Benchmarking Nearest Neighbor Search

Martin Aumüller[0000−0002−7212−6476] and Matteo Ceccarello[0000−0003−2783−0218]

IT University of Copenhagen
Copenhagen, Denmark
{maau,mcec}@itu.dk

Abstract. This paper reconsiders common benchmarking approaches to nearest neighbor search. It is shown that the concept of local intrinsic dimensionality (LID) allows to choose query sets of a wide range of difficulty for real-world datasets. Moreover, the effect of different LID distributions on the running time performance of implementations is empirically studied. To this end, different visualization concepts are introduced that allow to get a more fine-grained overview of the inner workings of nearest neighbor search principles. The paper closes with remarks about the diversity of datasets commonly used for nearest neighbor search benchmarking. It is shown that such real-world datasets are not diverse: results on a single dataset predict results on all other datasets well.

1 Introduction

Nearest neighbor (NN) search is a key primitive in many computer science applications, such as data mining, machine learning and image processing. For example, Spring and Shrivastava very recently showed in [25] how nearest neighbor search methods can yield large speed-ups when training neural network models. In this paper, we study the classical k-NN problem. Given a dataset $S \subseteq \mathbb{R}^d$, the task is to build an index on S to support the following type of query: For a query point $\mathbf{x} \in \mathbb{R}^d$, return the k closest points in S under some distance measure D.

In many practical settings, a dataset consists of points represented as high-dimensional vectors. For example, word representations generated by the `glove` algorithm [23] associate with each word in a corpus a d-dimensional real-valued vector. Common choices for d are between 50 and 300 dimensions. Finding the true nearest neighbors in such a high-dimensional space is difficult, a phenomenon often referred to as the "curse of dimensionality" [8]. In practice, it means that finding the true nearest neighbors, in general, cannot be solved much more efficiently than by a linear scan through the dataset (requiring time $O(n)$ for n data points) or in space that is exponential in the dimensionality d, which is impractical for large values of d.

While we cannot avoid these general hardness results [1], most datasets that are used in applications are not *truly* high-dimensional. This means that the dataset can be embedded onto a lower-dimensional space without too much distortion. Intuitively, the intrinsic dimensionality (ID) of the dataset is the

minimum number of dimensions that allows for such a representation [11]. There exist many explicit ways of finding good embeddings for a given dataset. For example, the Johnson-Lindenstrauss transformation [16] allows us to embed n data points in $\mathbb{R}^d$ into $\Theta((\log n)/\varepsilon^2)$ dimensions such that all pairwise distances are preserved up to a $(1+\varepsilon)$ factor with high probability. Another classical embedding often employed in practice is given by principal component analysis (PCA), see [17].

In this paper, we put our focus on "local intrinsic dimensionality" (LID), a measure introduced by Houle in [11]. We defer a detailed discussion of this measure and its estimation to Section 2. Intuitively, the LID of a data point $\mathbf{x}$ at a distance threshold $r > 0$ measures how difficult it is to distinguish between points at distance r and distance $(1+\varepsilon)r$ in a dataset. Most importantly for this study, LID is a *local* measure that can be associated with a single query. It was stated in [12] that the LID might serve as a characterization of the difficulty of k-NN queries. One purpose of this paper is to shed light on this statement.

A focus of this paper is an empirical study of how the LID influences the performance of NN algorithms. To be precise, we will benchmark four different implementations [18] which employ different approaches to NN search. Three of them (`HNSW` [21], `FAISS-IVF` [15], `Annoy` [6]) stood out as most performant in the empirical study conducted by Aumüller et al. in [4]. Another one (`ONNG`) was proposed very recently [13] and shown to be competitive to these approaches. We base our experiments on [4] and describe their benchmarking approach and the changes we made to their system in Section 3. We analyze the LID distribution of real-world datasets in Section 4. We will see that there is a substantial difference between the LID distributions among datasets. We will next conduct two experiments: First, we fix a dataset and choose as query set the set of points with smallest, medium, and largest estimated LIDs. In addition, we choose a set of "diverse" query points w.r.t. their LID estimates. As we will see, there is a clear tendency such that the larger the LID, the more difficult the query for all implementations. Next, we will study how the different LID distributions between datasets influence the running time performance. In a nutshell, it cannot be concluded that LID by itself is a good indicator for the relative performance of a fixed implementation over datasets. These statements will be made precise in the evaluation that is discussed in Section 5.

In the first part of our evaluation, we work in the "classical evaluation setting of nearest neighbor search". This means that we relate a performance measure (such as the achieved throughput measured in queries per second) to a quality measure (such as the average fraction of true nearest neighbors found over all queries). While this is the most commonly employed evaluation method, we reason that this way of representing results in fact hides interesting details about the inner workings of an implementation. Using non-traditional visualization techniques provide new insights into their query behavior on real-world datasets. As one example, we see that reporting average recall on the graph-based approaches from [21,13] hides an important detail: For a given query, they either find all true nearest neighbors or not a single one. This behavior is not shared by the

two other approaches that we consider; both yield a continuous transition from "finding no nearest neighbors" to "finding all of them".

As a final point, we want, ideally, to benchmark on a collection of "interesting" datasets that show the strengths and weaknesses of individual approaches [24]. We will conclude that there is little diversity among the considered real-word datasets: While the individual performance observations change from dataset to dataset, the relative performance between implementations stays the same.

Our Contributions. The main contributions of this paper are

- a detailed evaluation of the LID distribution of many real-world datasets used in benchmarking frameworks,
- an evaluation of the influence of the LID on the performance of NN search implementations,
- considerations about the result diversity, and
- an exploration of different visualization techniques that shed light on individual properties of certain implementation principles.

A preliminary workshop version of this paper appeared as [5]. In this paper we expand the experimental study with the correlation between LID and recall; we also consider different ways of generating synthetic datasets to investigate the relationship between LID and performance.

Related Work on Benchmarking Frameworks for NN. We use the benchmarking system described in [4] as the starting point for our study. Different approaches to benchmarking nearest neighbor search are described in [9,10,20]. We refer to [4] for a detailed comparison between the frameworks.

2 Local Intrinsic Dimensionality

We consider a distance-space $(\mathbb{R}^d, D)$ with a distance function $D\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. As described in [2], we consider the distribution of distances within this space with respect to a reference point $\mathbf{x}$. Such a distribution is induced by sampling n points from the space $\mathbb{R}^d$ under a certain probability distribution. We let $F\colon \mathbb{R} \to [0,1]$ be the cumulative distribution function of distances to the reference point $\mathbf{x}$.

Definition 1 ([11]). *The local continuous intrinsic dimension of F at distance r is given by*

$$ID_F(r) = \lim_{\varepsilon \to 0} \frac{\ln(F((1+\varepsilon)r)/F(r))}{\ln((1+\varepsilon)r/r)},$$

whenever this limit exists.

The measure relates the increase in distance to the increase in probability mass (the fraction of points that are within the ball of radius r and $(1+\varepsilon)r$ around the query point). Intuitively, the larger the LID, the more difficult it is to distinguish true nearest neighbors at distance r from the rest of the dataset. As described in [12], in the context of k-NN search we set r as the distance of the k-th nearest neighbor to the reference point $\mathbf{x}$.

Estimating LID We use the Maximum-Likelihood estimator (MLE) described in [19,2] to estimate the LID of $\mathbf{x}$ at distance r. Let $r_1 \leq \ldots \leq r_k$ be the sequence of distances of the k-NN of $\mathbf{x}$. The MLE $\hat{\mathrm{ID}}_{\mathbf{x}}$ is then

$$\hat{\mathrm{ID}}_{\mathbf{x}} = -\left(\frac{1}{k}\sum_{i=1}^{k} \ln \frac{r_i}{r_k}\right)^{-1}. \tag{1}$$

Amsaleg et al. showed in [2] that MLE estimates the LID well. We remark that in very recent work, Amsaleg et al. proposed in [3] a new MLE-based estimator that works with smaller k values than (1).

3 Overview over the Benchmarking Framework

We use the `ann-benchmarks` system described in [4] to conduct our experimental study. Ann-benchmarks is a framework for benchmarking NN search algorithms. It covers dataset creation, performing the actual experiment, and storing the results of these experiments in a transparent and easy-to-share way. Moreover, results can be explored through various plotting functionalities, e.g., by creating a website containing interactive plots for all experimental runs.

Ann-benchmarks interfaces with a NN search implementation by calling its preprocess (index building) and search (query) methods with certain parameter choices. Implementations are tested on a large set of parameters usually provided by the original authors of an implementation. The answers to queries are recorded as the indices of the points returned. Ann-benchmarks stores these parameters together with further statistics such as individual query times, index size, and auxiliary information provided by the implementation. See [4] for more details.

Compared to the system described in [4], we added tools to estimate the LID based on Equation (1), pick "challenging query sets" according to the LID of individual points, and added new datasets and implementations. Moreover, we implemented a mechanism that allows an implementation to provide further query statistics after answering a query. To showcase this feature, all implementations in this study report the number of distance computations performed to answer a query.[1]

4 Algorithms and Datasets

4.1 Algorithms

Nearest neighbor search algorithms for high dimensions are usually graph-, tree-, or hashing-based. We refer the reader to [4] for an overview over these principles and available implementations. In this study, we concentrate on the three implementations considered most performant in [4], namely `HNSW` [21],

[1] We thank the authors of the implementations for their help and responsiveness in adding this feature to their library.

Dataset	Data Points	Dimensions	LID avg	LID median	Metric
SIFT [14]	1 000 000	128	21.9	19.2	Euclidean
MNIST	65 000	784	14.0	13.2	Euclidean
Fashion-MNIST [26]	65 000	784	15.6	13.9	Euclidean
GLOVE [23]	1 183 514	100	18.0	17.8	Angular/Cosine
GLOVE-2M [23]	2 196 018	300	26.1	23.4	Angular/Cosine
GNEWS [22]	3 000 000	300	21.1	20.1	Angular/Cosine

Table 1. Datasets under consideration with their average local intrinsic dimensionality (LID) computed by MLE [2] from the 100-NN of all the data points.

`Annoy` [6] and `FAISS-IVF` [15] (`IVF` from now on). We consider the very recent graph-based approach `ONNG` [13] in this study as well.

`HNSW` and `ONNG` are graph-based approaches. This means that they build a k-NN graph during the preprocessing step. In this graph, each vertex is a data point and a directed edge (u, v) means that the data point associated with v is "close" to the data point associated with u in the dataset. At query time, the graph is traversed to generate candidate points. Algorithms differ in details of the graph construction, how they build a navigation structure on top of the graph, and how the graph is traversed.

`Annoy` is an implementation of a random projection forest, which is a collection of random projection trees. Each node in a tree is associated with a set of data points. It splits these points into two subsets according to a chosen hyperplane. If the dataset in a node is small enough, it is stored directly and the node is a leaf. `Annoy` employs a data-dependent splitting mechanism in which a splitting hyperplane is chosen as the one splitting two "average points" by repeatedly sampling dataset points. In the query phase, trees are traversed using a priority queue until a predefined number of points is found.

`IVF` builds an inverted file based on clustering the dataset around a predefined number of centroids. It splits the dataset based on these centroids by associating each point with its closest centroid. During query it finds the closest centroids and checks points in the dataset associated with those.

We remark we used both `IVF` and `HNSW` implementations from `FAISS`[2].

4.2 Datasets

Table 1 presents an overview over the datasets that we consider in this study. We restrict our attention to datasets that are usually used in connection with Euclidean distance and Angular/Cosine distance. For each dataset, we compute the LID distribution with respect to the 100-NN as discussed in Section 2. SIFT, MNIST, and GLOVE are among the most-widely used datasets for benchmarking

[2] `https://github.com/facebookresearch/faiss`

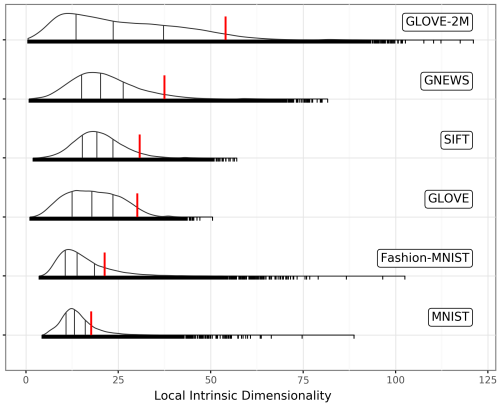

Fig. 1. LID distribution for each dataset. Ticks below the distribution curves represent single queries. Lines within each distribution curve correspond to the 25, 50 and 75 percentile. The red line marks the 10 000 largest estimated LID, which we use as a threshold value to define *hard* query sets.

nearest neighbor search algorithms. Fashion-MNIST is considered as a replacement for MNIST, which is usually considered too easy for machine learning tasks [26].

Figure 1 provides a visual representation of the estimated LID distribution of each dataset, for $k = 100$. While the datasets differ widely in their original dimensionality, the median LID ranges from around 13 for MNIST to about 23 for GLOVE-2M. The distribution of LID values is asymmetric and shows a long tail behavior. MNIST, Fashion-MNIST, SIFT, and GNEWS are much more concentrated around the median compared to the two `glove`-based datasets.

5 Evaluation

This section reports on the results of our experiments. Due to space constraints, we only present some selected results. More results and plots can be explored via interactive plots at `http://ann-benchmarks.com/sisap19/`, which also contains a link to the source code repository. For a fixed implementation, the plots presented here consider the Pareto frontier over all parameter choices [4]. Tested parameter choices and the associated plots are available on the website.

Experimental Setup Experiments were run on 2x 14-core Intel Xeon E5-2690v4 (2.60GHz) with 512GB RAM using Ubuntu 16.10 (kernel 4.4.0). Index building was multi-threaded, queries where answered in a single thread.

Quality and Performance Metrics As quality metric we measure the individual recall of each query, i.e., the fraction of points reported by the implementation that are among the true k-NN. As performance metric, we record individual query times and the total number of distance computations needed to answer all queries. We usually report on the throughput (the average number of queries that can be answered in one second, in the plots denoted as QPS for *queries per second*), but we will also inspect individual query times.

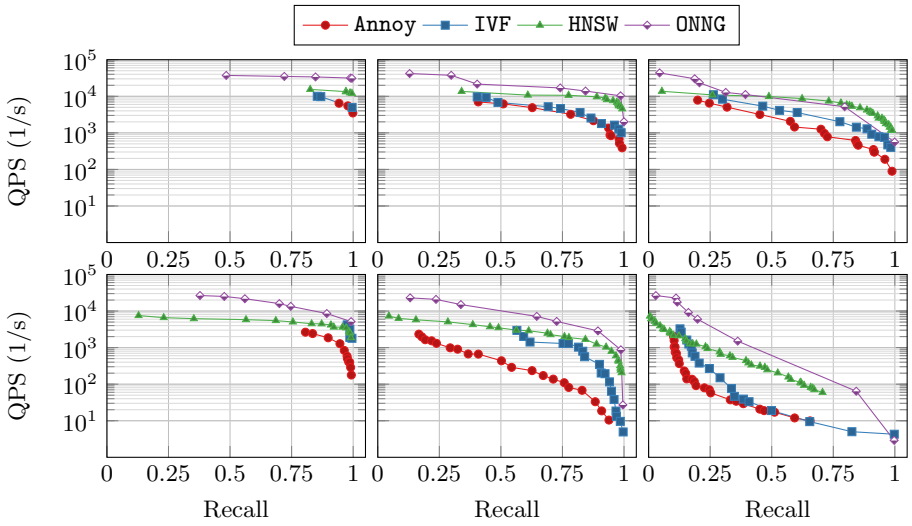

Fig. 2. Recall-QPS (1/s) tradeoff – up and to the right is better; top: SIFT, bottom: GLOVE-2M; left: easy, middle: middle, right: hard.

Objectives of the Experiments Our experiments are tailored to answer the following questions:

(Q1) How does the LID of a query set influence the running time performance? (Section 5.1)
(Q2) How diverse are measurements obtained on datasets? Do relative differences between the performance of different implementations stay the same over multiple datasets? (Section 5.2)
(Q3) How well does the number of distance computations reflect the relative running time performance of the tested implementations? (Section 5.2)
(Q4) How concentrated are quality and performance measures around their mean for the tested implementations? (Section 5.3)

Choosing Query Sets For each dataset, we select four different query sets: The query set that contains the 10 000 points with the lowest estimated LID (which we denote *easy*), 10 000 points around the data point with median estimated LID (denoted *medium*), 10 000 points with the largest estimated LID (dubbed *hard*), and 5 000 points chosen uniformly according to (integer) LID values (denoted *diverse*). For the latter, we split all data points up into buckets, where bucket i represents all data points that have an estimated LID of i (rounded down). For each query, we pick a non-empty bucket uniformly at random, and inside the bucket we pick a random point (with repetition). Figure 1 marks with a red line the LID used as a threshold to build the *hard* queryset.

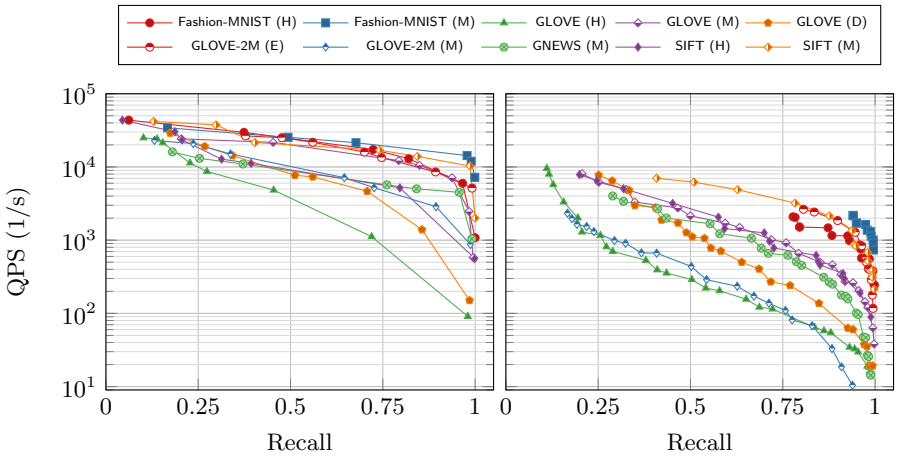

Fig. 3. Recall-QPS (1/s) tradeoff – up and to the right is better; left: `ONNG`, right: `Annoy`; (E) — easy, (M) — medium, (H) — hard, (D) — diverse.

5.1 Influence of LID on Performance

Figure 2 shows results for the influence of using only points with low, middle, and large estimated LID as query points, in SIFT and GLOVE-2M. We observe a clear influence of the LID of the query set on the performance: the larger the LID, the more down and to the left the graphs move. This means that, for higher LID, it is more expensive, in terms of time, to answer queries with good recall. For all datasets except GLOVE-2M, all implementations were still able to achieve close to perfect recall with the parameters set. This means that all but one of the tested datasets do not contain too many "noisy queries". Already the queries around the median prove challenging for most implementations. For the most difficult queries (according to LID), only `IVF` and `ONNG` achieve close to perfect recall on GLOVE-2M.

Figure 3 reports on the results of `ONNG` and `Annoy` on selected datasets. Comparing results to the LID measurements depicted in Figure 1, the estimated median LID gives a good estimate on the relative performance of the algorithms on the data sets. As an exception, SIFT (M) is much easier than predicted by its LID distribution. In particular for `Annoy`, the hard SIFT instance is as challenging as the medium GLOVE version. The easy version of GLOVE-2M turns out to be efficiently solvable by both implementations (taking about the same time as it takes to answer the hard instance of Fashion-MNIST, which has a much higher LID). From this, we cannot conclude that LID as a single indicator explains performance differences of an implementation across different datasets. However, more careful experimentation is need before drawing a final conclusion. In our setting, the LID estimation is conducted for $k = 100$, while queries are only searching for the 10 nearest neighbors. Moreover, the estimation using MLE might not be accurate enough on these datasets, since it is very dependent on

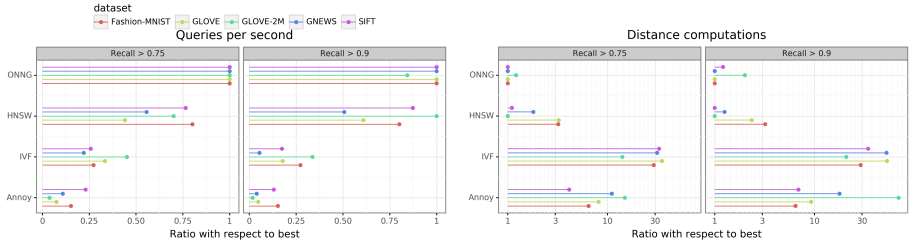

Fig. 4. Ranking of algorithms on five different datasets, according to recall ≥ 0.75 and ≥ 0.9, and according to two different performance measures: number of queries per second (left) and number of distance computations (right). Both plots report the ratio with the best performing algorithm on each dataset: for the queries per second metric a larger ratio is better, for the number of distance computations metric, a smaller ratio is better.

the parameter k being used. We leave the investigation of these two directions as future work.

In general, the diverse query set is more difficult than the medium query set. In particular, at high recall it generally becomes nearly as difficult as the difficult dataset. The reason for this behavior is that none of the implementations can adapt to the difficulty of a query. They only achieve high average recall when they can solve sufficiently many queries with high LID. The parameter settings that allow for such guarantees slow down answering the easy queries by a lot. We believe that the "diverse" query sets thus allow for challenging benchmarking datasets for adaptive query algorithms.

As a side note, we remark that Fashion-MNIST is as difficult to solve as MNIST for all implementations, and is by far the easiest dataset for all implementations. Thus, while there is a big difference in the difficulty of solving the classification task [26], there is no measurable difference between these two datasets in the context of NN search.

5.2 Diversity of Results

Figure 4 gives an overview over how algorithms compare to each other among all "medium difficulty" datasets. We consider two metrics, namely the number of queries per second (left plot), and the number of distance computations (right plot). For two different average recall thresholds (0.75 and 0.9) we then select, for each algorithm, the best performing parameter configuration that attains at least that recall. For each dataset, the plots report the ratio with the best performing algorithm on that dataset, therefore the best performer is reported with ratio 1. Considering different dataset, we see that there is little variation in the ranking of the algorithms. Only the two graph-based approaches trade ranks,

all other rankings are stable. Interestingly, `Annoy` makes much fewer distance computations but is consistently outperformed by `IVF`.[3]

Comparing the number of distance computations to running time performance, we see that an increase in the number of distance computations is not reflected in a proportional decrease in the number of queries per second. This means that the candidate set generation is in general more expensive for graph-based approaches, but the resulting candidate set is of much higher quality and fewer distance computations have to be carried out. Generally, both graph-based algorithms are within a factor 2 from each other, whereas the other two need much larger candidate lists to achieve a certain recall. The relative difference usually ranges from 5x to 30x more distance computations for the non-graph based approaches, in particular at high recall. This translates well into the performance differences we see in this setting: consider for instance Figure 2, where the lines corresponding to `HNSW` and `ONNG` upper bound the lines relative to the other two algorithms.

5.3 Reporting the Distribution of Performance

In the previous sections, we made extensive use of recall/queries per second plots, where each configuration of each algorithm results in a single point, namely the average recall and the inverse of the average query time. As we shall see in this section, concentrating on averages can hide interesting information in the context of k-NN queries. In fact, not all queries are equally difficult to answer. Consider the plots in Figure 5, which report performance of the four algorithms[4] on the GLOVE-2M dataset, medium difficulty. The top four plots report the recall versus the number of queries per second, and black dots correspond to the averages. Additionally, for each configuration, we report the distribution of the recall scores: the baseline of each recall curve is positioned at the corresponding queries per second performance. Similarly, the bottom plots report on the inverse of the individual query times (the average of these is the QPS in the left plot) against the average recall. In both plots, the best performance is achieved towards the top-right corner.

Plotting the distributions, instead of just reporting the averages, uncovers some interesting behaviour that might otherwise go unnoticed, in particular with respect to the recall. The average recall gradually shifts towards the right as the effect of more and more queries achieving good recalls. Perhaps surprisingly, for graph-based algorithms this shift is very sudden: most queries go from having recall 0 to having recall 1, taking no intermediate values. Taking the average recall as a performance metric is convenient in that it is a single number to compare algorithms with. However, the same average recall can be attained with very different distributions: looking at such distributions can provide more insight.

[3] We note that `IVF` counts the initial comparisons to find the closest centroids as distance computations, whereas `Annoy` did not count the inner product computations during tree traversal.

[4] In order not to clutter the plots, we fixed parameters as follows: `IVF` — number of lists 8192; `Annoy` — number of trees 100; `HNSW` — efConstruction 500, M 8; `ONNG` — edge 100, outdegree 10, indegree 120.

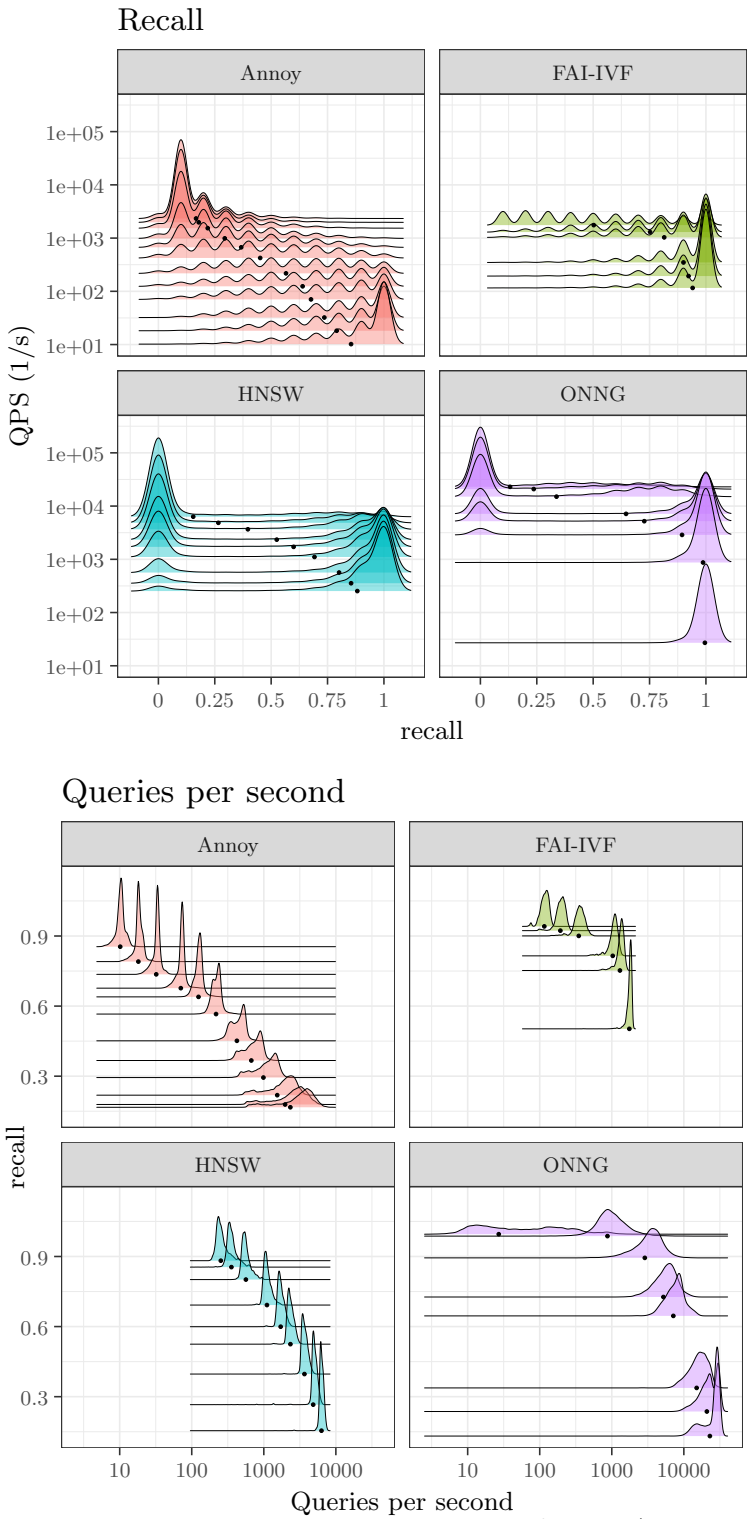

Fig. 5. Distribution of performance for queries on the GLOVE-2M (medium difficulty) dataset. Looking just at the average performance can hide interesting behaviour.

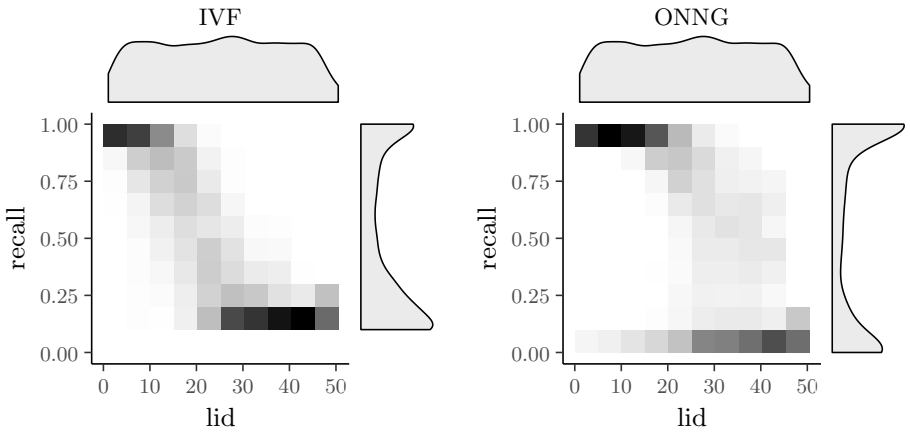

Fig. 6. Distribution of Recall vs. LID plot on the GLOVE datset. Intensity reflects number of queries that achieve a combination of recall vs. LID.

For the bottom plots, we observe that individual query times of all the algorithms are well concentrated around their mean.

Figure 6 gives another distributional view on the achieved result quality. The plots shows two runs of `IVF` and `ONNG` with fixed parameters on the GLOVE dataset with diverse queries. On the top we see the distribution of estimated LID values for the diverse query set, on the right we see the distribution of recall values achieved by the implementation. Each of the queries corresponds to a single data point in the recall/LID plot and data points are summarized through hexagons, where the color intensity of a hexagon indicates the number of data points falling into this region. The plots show that the higher the LID of a query, there is a clear tendency for the query to achieve lower recall.

For space reasons, we do not report other parameter configurations and datasets, which nonetheless show similar behaviours. All of them can be accessed at the website.

6 Summary

In this paper we studied the influence of LID to the performance of nearest neighbor search algorithms. We showed that LID allows to choose query sets of a wide range of difficulty from a given dataset. We also showed how different LID distributions influence the running time performance of the algorithms. In this respect, we could not conclude that the LID alone can predict running time differences well. In particular, SIFT is usually easier for the algorithms, while GLOVE's LID distribution would predict it to be the easier dataset of the two.

With regard to challenging query workloads, we described a way to choose diverse query sets. They have the property that for most implementations it is easy to perform well for most of the query points, but they contain many more

easy and difficult queries than query workloads chosen randomly from the dataset. We believe this is a very interesting benchmarking workload for approaches that try to adapt to the difficulty of an individual query.

We introduced novel visualization techniques to show the uncertainty within the answer to a set of queries, which made it possible to show a clear difference between the graph-based algorithms and the other approaches.

We hope that this study initiates the search for more diverse datasets, or for theoretical reasoning why certain algorithmic principles are generally better suited for nearest neighbor search. On a more practical side, Casanova et al. showed in [7] how dimensionality testing can be used to speed up reverse k-NN queries. We would be interested in seeing whether the LID can be used at other places in the design of NN algorithms to guide the search process or the parameter selection. While we know from [2] that the LID estimation of MLE with $k = 100$ works well on their datasets, it would be interesting to see whether the other estimators mentioned there are also able to characterize the relative performance of queries.

Acknowledgements The authors would like to thank the anonymous reviewers for their useful suggestions, which helped to improve the presentation of the paper. The research leading to these results has received funding from the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331.

References

1. Alman, J., Williams, R.: Probabilistic polynomials and hamming nearest neighbors. In: FOCS'15. pp. 136–150
2. Amsaleg, L., Chelly, O., Furon, T., Girard, S., Houle, M.E., Kawarabayashi, K.I., Nett, M.: Estimating local intrinsic dimensionality. In: KDD'15. pp. 29–38. ACM (2015)
3. Amsaleg, L., Chelly, O., Houle, M.E., Kawarabayashi, K.i., Radovanović, M., Treeratanajaru, W.: Intrinsic dimensionality estimation within tight localities. In: Proceedings of the 2019 SIAM International Conference on Data Mining. pp. 181–189. SIAM (2019)
4. Aumüller, M., Bernhardsson, E., Faithfull, A.: ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. In: SISAP'17. pp. 34–49 (2017). https://doi.org/10.1007/978-3-319-68474-1_3
5. Aumüller, M., Ceccarello, M.: Benchmarking nearest neighbor search: Influence of local intrinsic dimensionality and result diversity in real-world datasets. In: 1st Workshop on Evaluation and Experimental Design in Data Mining and Machine Learning (EDML 2019) (2019), `https://imada.sdu.dk/Research/EDML/`
6. Bernhardsson, E.: Annoy, `https://github.com/spotify/annoy`
7. Casanova, G., Englmeier, E., Houle, M.E., Kröger, P., Nett, M., Schubert, E., Zimek, A.: Dimensional testing for reverse k-nearest neighbor search. PVLDB **10**(7), 769–780 (2017)
8. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. **33**(3), 273–321 (Sep 2001). https://doi.org/10.1145/502807.502808

9. Curtin, R.R., Cline, J.R., Slagle, N.P., March, W.B., Ram, P., Mehta, N.A., Gray, A.G.: MLPACK: A scalable C++ machine learning library. J. of Machine Learning Research **14**, 801–805 (2013)
10. Edel, M., Soni, A., Curtin, R.R.: An automatic benchmarking system. In: NIPS 2014 Workshop on Software Engineering for Machine Learning (2014)
11. Houle, M.E.: Dimensionality, discriminability, density and distance distributions. In: Data Mining Workshops (ICDMW). pp. 468–473. IEEE (2013)
12. Houle, M.E., Schubert, E., Zimek, A.: On the correlation between local intrinsic dimensionality and outlierness. In: SISAP'18. pp. 177–191 (2018). https://doi.org/10.1007/978-3-030-02224-2_14
13. Iwasaki, M., Miyazaki, D.: Optimization of Indexing Based on k-Nearest Neighbor Graph for Proximity Search in High-dimensional Data. ArXiv e-prints (Oct 2018)
14. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell. **33**(1), 117–128 (2011). https://doi.org/10.1109/TPAMI.2010.57
15. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with gpus. CoRR **abs/1702.08734** (2017)
16. Johnson, W.B., Lindenstrauss, J., Schechtman, G.: Extensions of Lipschitz maps into Banach spaces. Israel Journal of Mathematics **54**(2), 129–138 (1986)
17. Jolliffe, I.: Principal component analysis. Springer (2011)
18. Kriegel, H., Schubert, E., Zimek, A.: The (black) art of runtime evaluation: Are we comparing algorithms or implementations? Knowl. Inf. Syst. **52**(2), 341–378 (2017)
19. Levina, E., Bickel, P.J.: Maximum likelihood estimation of intrinsic dimension. In: NIPS'15. pp. 777–784 (2005)
20. Li, W., Zhang, Y., Sun, Y., Wang, W., Zhang, W., Lin, X.: Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement (v1.0). CoRR **abs/1610.02455** (2016)
21. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. ArXiv e-prints (Mar 2016)
22. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)
23. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Empirical Methods in Natural Language Processing (EMNLP). pp. 1532–1543 (2014)
24. Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R.: Towards objective measures of algorithm performance across instance space. Computers & Operations Research **45**, 12 – 24 (2014)
25. Spring, R., Shrivastava, A.: Scalable and sustainable deep learning via randomized hashing. In: KDD'17. pp. 445–454 (2017). https://doi.org/10.1145/3097983.3098035
26. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. CoRR **abs/1708.07747** (2017)