Evaluating Approximate Nearest Neighbour Search Systems on Knowledge Graph Embeddings

Gaurav Pandit[1][0000−0002−6768−3403], Michael Röder[2][0000−0002−8609−8277], and Axel-Cyrille Ngonga Ngomo[2][0000−0001−7112−3516]

[1] Paderborn University, Germany
[2] Data Science Group (DICE), Heinz Nixdorf Institute, Paderborn University, Germany
{michael.roeder|axel.ngonga}@uni-paderborn.de

Abstract. Knowledge Graph Embeddings are often used to bridge symbolic representations of Knowledge Graphs and the sub-symbolic representations that modern Machine Learning algorithms operate on. While embedding models provide users with a mapping from symbolic entities and relations to their sub-symbolic representations, a mapping in the opposite direction typically relies on a nearest neighbor search. Due to the computational complexity of this task, a plethora of approaches for approximate nearest neighbor search have been developed. The majority of these approaches outperform the default brute-force-based approach while providing a high recall. However, previous evaluations of these approaches focused on image data and word embeddings but did not consider Knowledge Graph Embeddings. We close this gap by carrying out a detailed comparison of 22 Approximate Nearest Neighbor Search systems on 16 datasets. In contrast to the state of the art, we fine-tune each approach in each experiment by using Bayesian optimization to ensure the fairness of our experiments. Our results suggest that the overall performance of approaches with respect to runtime and recall is contingent upon the similarity measure used to compare embeddings. Our source code, datasets and results are available at https://github.com/MichaelRoeder/ann-benchmarks/tree/main.

Keywords: Knowledge Graph Embeddings · k-Nearest Neighbor Search · Approximate Nearest Neighbor Search.

1 Introduction

Knoweldge Graph Embeddings (KGEs) have a wide area of applications [8] ranging from link prediction [11] to Knowledge-Graph-based question answering [5,6] and recommendation systems [37,16]. In several of the aforementioned applications, one or several entities found in the embedded Knoweldge Graph (KG) have to be identified based on the similarity (or distance) of their embedding vector to a given query vector. A brute force comparison of the query vector to all known embedding vectors is a simple, straight-forward solution. However, it comes with a linear time complexity in the number of entities in the input KG

and in the number of dimensions of the embeddings. Consequently, brute-force approaches yield a high runtime if

1. there are many vectors in the embedding model and
2. the embedding vectors have a high dimensionality.

This is known as the k nearest neighbors (k-NN) search problem. In recent years, a plethora of algorithms has been proposed that try to approximate the solution for this search problem [2,26,34]. These algorithms are called approximate nearest neighbors search (ANNS) algorithms. The main idea behind ANNS approaches is to accept a small error in the result in exchange for a runtime or memory consumption reduction when compared to exact k-NN search algorithms [26]. Muja et al. [28] report that the choice of the optimal ANNS algorithm highly depends on factors including the data dimensionality and the size and structure of the dataset. Due to their wide applicability, ANNS algorithms have been compared on many different datasets including audio, image, video, text and word embedding datasets in the literature [2,26]. However, to the best of our knowledge, their performance on KGEs has never been studied so far and it is unclear whether previous results hold in this different application area.[3]

Our work tackles this research gap via two main contributions.

1. We provide the first comprehensive evaluation of ANNS algorithms on KGEs. Relying on an ANN benchmarking framework proposed by Aumüller et al. [2], we compare 22 ANNS systems on 16 datasets. These datasets are created using 2 KGs, 2 distance metrics and 4 KGEs models.
2. For the first time, we address the challenge of finding suitable parameters for ANNS systems by using Bayesian optimization. Through this superior approach for parameter tuning, we achieve a fairer comparison of the systems we evaluate than previous grid-search-based works [2].

In the following, we describe related work with a focus on the systems that we evaluate. Section 3 gives a formal problem definition of the problem we tackle and describes the extensions of the ANN benchmarking framework necessary for our experiments. Section 4 comprises a description of our experiments, including the used data, they key performance indicators and the gathered results. We discuss these results in Section 5 before we conclude the article in Section 6.

2 Related Work

A large number of exact and approximate k-NN search algorithms have been proposed [2,26,34]. In recent years, several articles have been published that compare these algorithms with respect to their effectiveness and efficiency on different types of data. For example, Wen et al. [26] compare 8 algorithms on 20

[3] KGEs tend to contain millions of entities but with less dimensions than the image embeddings used in the related work. At the same time, Alshahrani et. al [1] showed that different embedding algorithms use the embedding space in very different ways.

datasets with a focus on the Euclidean distance. Their datasets comprise mainly image data but also include audio, video, text and synthetically generated data. Wang et al. [34] compare 15 algorithms on 8 datasets with a focus on on graph-based ANNS algorithms. Their evaluation also relies on audio, image, text and video data. Aumüller et al. [2] compare 14 algorithms on 7 datasets and include the angular and Hamming distance. None of these works targets the effectiveness and efficiency of ANNS systems on KGEs.

Aumüller et al. [2] also propose an open-source ANN benchmarking framework that they used for their evaluations.[4] We use this framework as basis for our experiments since it provides the general workflow for a repeatable evaluation and comes with a large number of integrated ANNS systems. Consequently—in contrast to Wen et al. [26]—, we do not alter the algorithm implementations to reduce the impact of optimizations the developers may have integrated into their solutions. Instead, we evaluate ANNS systems that are ready for use. We also include several systems that implement the same base algorithm to see whether the implementation of an algorithm has an impact on our evaluation results.[5]

While there are many ANNS systems available, we ensure the practical feasibility of our study by focusing our evaluation on systems that

1. can handle floating point numbers,
2. support at least the one of the angular and Euclidean distance metrics,
3. have been integrated into the ANN benchmarking framework,
4. come with a working Docker image, and
5. are suggested for benchmarking by the authors of the ANN benchmarking framework.[6]

Table 1 lists the 22 chosen systems, which are briefly described in the following subsections.[7] Note that the list does neither contain hash-based nor neural algorithms. Systems of the former type rely on a hash algorithm like locality sensitive hashing [20,29] to create a list of possible neighbors while the latter are a set of approaches that utilize neural networks to improve existing search algorithms further [25,33], e.g., by learning a hash function [7]. We have to exclude these algorithms since none of them fulfills the criteria above at the time of our evaluation.

[4] The framework is available at `https://github.com/erikbern/ann-benchmarks/`.

[5] Note that some of the systems offer additional features, e.g., the support of a variety of distance metrics, support for a distributed environment, or the usage of the computational power of GPUs [13]. We won't take these features into account for our experiments as they go beyond the focus of this paper.

[6] Some algorithms, like Annoy (`https://github.com/spotify/annoy`) have been marked as "disabled" since they have a successor algorithm (in this case, Voyager) that typically performs better.

[7] We also include the pyglass library into our experiments (`https://github.com/zilliztech/pyglass`). However, it does not achieve a Recall above 0 for any configuration that we try. Hence, we won't discuss it within this article.

Table 1. The ANN systems that we evaluate, their main algorithm, and links to their project page.

Type	System	Algorithm	Link
Graph-based	Elasticsearch	HNSW [27]	
	FaissHNSW	HNSW [27]	
	Hnswlib	HNSW [27]	
	LuceneKNN	HNSW [27]	
	N2	HNSW [27]	
	NGT-PANNG	PANNG [22]	
	NGT-QG	QG [23]	
	NN-Descent	NN-Descent [13]	
	OpenSearch k-NN	HNSW [27]	
	pg-embedding	HNSW [27]	
	PyNNDescent	NN-Descent [13]	
	QSGNGT	QG [23]	
	Vald	QG [23]	
	Vespa	HNSW [27]	
	Voyager	HNSW [27]	
	Weaviate	HNSW [27]	
Tree-based	FLANN [28]	kd-trees / k-means tree	
	MRPT	MRPT [19]	
	Scann	SOAR [32] + VQ [18]	
IVF-based	FaissIVF	IVF	
	FaissIVFPQ	IVF + PQ	
	TinyKNN	IVF + PQ	

2.1 Graph-based Approaches

Graph-based ANNS approaches create a k-NN neighborhood graph, in which data points of the original dataset are the nodes that are connected to their k neighboring vertices with an edge [34]. However, creating an exact k-NN graph is expensive [21]. The NN-Descent algorithm [13] can be used to generate an approximate nearest neighbor graph (ANNG) and is based on the idea that two neighboring vertices are very likely to have the same neighbors. Given an initial ANNG, the algorithm traverses the neighbors of nodes and checks whether their neighbors are also within the k nearest neighbors of the current node. Based on this concept, the algorithm creates an updated version of the given graph. PyNNDescent and NN-Descent are implementations of the NN-Descent algorithm in Python an C++, respectively. Both implementations extend the original algorithm with an initialization based on random projection trees [9] to generate the initial version. Several algorithms of the NGT library are built upon a similar approach to create ANNG [21]. NGT-PANNG [22] is an implementation that includes the pruning of excessive edges in the ANNG. NGT-QG and QSGNGT are further extensions based on quantization graphs (QG), i.e., they make use of product quantization to store approximations of the given vectors that are

shorter and, hence, faster to process [23]. Vald is a variant thereof with a focus on distributed setups. One approach to search in an ANNG with a given query vector is to start at one or several seed nodes and to compare the distances of this node and its neighbors to the query node. The closest node is chosen and the search continues until no closer neighbor can be found. Another approach is to further explore nodes within a search space that is narrowed down during the search process [22].

The Hierarchical Navigable Small World (HNSW) algorithm [27] separates the neighborhood graph into several hierarchically connected graphs. The highest layer only contains a small number of randomly chosen points with larger distances while the lower layers contain neighborhoods with small distances. A search in the graph is performed in the same way as before but includes steps from the top layers to the bottom layers. In each layer, the number of neighbors per vertex is limited to ensure a search in logarithmic time. The HNSW algorithm has been widely adopted and in this article, we look at Elasticsearch, FaissHNSW, Hnswlib, LuceneKNN, N2 [27], OpenSearch k-NN, pg-embedding, Vespa, Voyager, and Weaviate, that all rely on this algorithm.[8]

2.2 Tree-based Approaches

Tree-based approaches separate the high-dimensional space into smaller sub spaces. The separated parts are then organized in a tree structure. MRPT [19] uses multiple random projection trees to separate the search space. The search results from the different trees are combined using voting. FLANN [28] is a system that is based on two different algorithms and decides which to use based on the given data. It can use the multiple randomized kd-trees [3,15] or the priority search k-means tree [17]. The latter is chosen in cases in which the vectors have many dimensions since the multiple randomized kd-trees are known to have a decreasing performance with high dimensional data [28]. Scann is based on SOAR [32], an approach that uses spill trees with multiple representations of the search space to reduce the probability that a nearest neighbor is missed due to the separation of the search space. However, in contrast to previous works, SOAR does not create the trees independently but uses an orthogonality-amplified residual loss. The goal is to optimize each tree to cover cases where other trees perform poorly. In addition, Scann uses a quantization method called vector quantization [18].

2.3 Inverted-file-based Approaches

Inverted-file-based approaches cluster the set of given vectors into inverted lists. When a search is performed on an inverted file (IVF) index, only vectors from

[8] According to the documentation, OpenSearch k-NN also provides a FaissIVF-based implementation for an ANN search index (`https://opensearch.org/docs/2.6/search-plugins/knn/approximate-knn/`). However, only the HNSW implementation seems to be integrated into the ANN Benchmarking framework.

a small number of lists are compared to the given query vector [24]. Faiss-IVF implements this algorithm while FaissIVFPQ and TinyKNN combine the IVF approach with product quantization (PQ).

3 Benchmarking Framework

We define the k nearest neighbor problem as follows: Given a set of vectors V in a space X and a distance function $\delta : X \times X \to \mathbb{R}^+$, find the set π that contains the k closest vectors to a given query vector u according to δ [2]. Formally, we define π as a k-subset of V such that $\forall v \in \pi \; \forall v' \in V \setminus \pi : \delta(u, v) \leq \delta(u, v')$. An algorithm that provides an approximation of π is called approximate nearest neighbor search (ANNS) algorithm. Within this article we focus on the angular and Euclidean distance functions for δ. The space X is defined by the used Knowledge Graph embedding model.

A dataset for the evaluation of an ANNS system comprises 1. a set of vectors that are to be indexed, 2. a set of query vectors, and 3. the ground truth which of the indexed vectors are the k-NNs of the single query vectors. Aumüller et al. [2] provide an ANN benchmarking framework that already provides the main functionalities that are necessary to carry out our experiments. Given a set of high-dimensional vectors, the framework generates a dataset by splitting the vectors into indexed and query vectors and generates the ground truth based on a brute force search. This search is also used as a baseline for comparison.

We propose two extensions of this framework. First, we use Bayesian optimization to choose parameters of the benchmarked systems. Second, we generate synthetic queries based on the indexed data instead of splitting the data into a train and a test split to account for the comparatively small size of the KGE models we use in our experiments.

3.1 Bayesian Optimization

Nearly all of the ANN systems integrated into the benchmarking framework have one or several parameters that can be used to adapt the system to the given data. The ANN benchmarking framework supports the configuration of a grid search to identify system configurations with different performance characteristics. However, grid search is known to be a disadvantageous option when optimizing parameters in comparison to a random search [4] or hyper parameter optimization [31]. We integrate the latter into the framework using the implementation of [30]. We use the upper confidence bound as acquisition function for the optimizer [35] and use the Recall value achieved by a system's configuration as quality metric for the optimization.

3.2 Query Generation

By default, the ANN benchmarking framework randomly selects 10,000 vectors from the given set of vectors and uses them as query vectors while the remaining

vectors will be indexed by the benchmarked system. However, this would remove more than 68% of the vectors in our evaluation, thus leading to a small number of indexed vectors and correspondingly unrealistic evaluation results. At the same time, using several thousand queries supports the reliability of the evaluation results. To circumvent the split of the embedding model vectors, we implement the generation of synthetic queries based on the given embedding vectors. For each vector dimension, we calculate the mean and standard deviation of the pairwise distances of the given vectors. Then, we generate a synthetic query vector by randomly sampling a vector from the set of given vectors. After that, for each vector dimension we sample a value from a Gaussian distribution that is defined by the previously determined mean and standard deviation for this dimension. Then, we randomly choose to add or subtract the sampled value from the sampled vector to calculate the value for the new query vector.

4 Evaluation

4.1 Datasets

We evaluate the 22 systems from Table 1 on 16 datasets. These 16 datasets are created using the angular and the Euclidean distance measures on 8 KGE models. These models are created for the FB15k-237 and Yago3-10 datasets. For each of these two Knowledge Graphs, we reuse 4 Knowledge Graph embedding models that have been pre-calculated using the embedding algorithms DistMult [36], QuatE [38], OMult [10] and Keci [11]. We choose these embedding algorithms because they operate with different vector spaces. The first three algorithms are based on $\mathbb{R}$, $\mathbb{H}$, and $\mathbb{O}$, respectively. In contrast, Keci determines the appropriate Clifford algebra depending on the KG [11]. All embedding models use vectors comprising 128 floats. Each model for FB15k-237 and Yago3-10 contains one vector for each of the Knowledge Graphs' 14,541 and 123,182 entities, respectively. We generate the datasets using these entity vectors as described in Section 3. So in addition to the entity vectors, we generate 10,000 synthetic query vectors and create the ground truth.

4.2 Key Performance Indicators

The goal of our analysis is to evaluate the effectiveness and efficiency of the benchmarked systems. The effectiveness is measured as Recall, i.e., we divide the number of correctly retrieved neighbors by $k = 10$.[9] With respect to the system's efficiency, we put our focus on the number of queries that a system is able to queries per second (QpS). However, we also report the size of the created index and the time the system needs to create it.

[9] Note that related work may call the measure Precision, Recall@k or Recall@10. The ANN benchmarking framework also reports the approximative recall. These results can be found in our result files but we will not discuss them in this article.

As suggested by the related work, we focus on those configurations that lead to the best performances of the benchmarked systems [2,14]. Let S be the set of the benchmarked systems listed in Table 1. Let Θ_s be the set of all possible configurations of the system $s \in S$. Let d be a dataset and M_d be the set of all results that we gathered for all systems on this dataset. A single result m_i in this set is represented as the tuple $m_i = (s_i, \theta_i, r_i, q_i, t_i, b_i)$, where s_i is a system, $\theta_i \in \Theta_s$ is the system's configuration that leads to the measured results r_i, q_i, t_i, and b_i that represent the Recall, QpS, the time needed to create the index and the index size, respectively. We define a comparison operator that identifies configurations that show a lower performance than other configurations with respect to either effectiveness or efficiency and are not worse with respect to the other. Let m_1 and m_2 be two results in M_d and let r_1, r_2, q_1, and q_2 be their recall and QpS scores. We define the comparator $\underset{r,q}{>}$ based on Recall and QpS as follows:

$$m_1 \underset{r,q}{>} m_2 \iff (r_1 > r_2 \wedge q_1 \geq q_2) \vee (r_1 \geq r_2 \wedge q_1 > q_2)\,. \tag{1}$$

Based on this operator, we define a subset of measurements $M_{d,r,q}$ that represent the best performance of the single systems with respect to Recall and QpS as follows:

$$M_{d,r,q} = \left\{ m_i \,\middle|\, m_i \in M_d \wedge \nexists m_j \in M_d : m_j \neq m_i \wedge s_j = s_i \wedge m_j \underset{r,q}{>} m_i \right\}, \tag{2}$$

where s_i and s_j are the two systems for which the results with m_i and m_j have been measured, respectively. This subset $M_{d,r,q}$ contains only the best configurations with respect to Recall and QpS that have been identified during our evaluation for all systems for a particular dataset. The curve that they create can be understood as the Pareto front of the systems on the dataset [14].

4.3 Setup

We evaluate the 22 systems listed in Table 1. All experiments are carried out with $k = 10$ on a single machine.[10] For all systems, for which the ANN benchmarking framework would run a grid search, we configure the Bayesian optimizer to search for the configuration with the best Recall in the parameter range that would have been used by the grid search. To keep the comparison of systems fair, we configured the benchmarking framework to use a maximum of 35 runs for each system regardless of the number of parameters the systems may have. We run the queries in batch mode, i.e., systems can answer queries in parallel. This has the following two benefits. First, it is closer to real applications in which for example multiple users interact with a KG-based system in parallel. Second, it enables us to run more experiments and, hence, test more configurations.

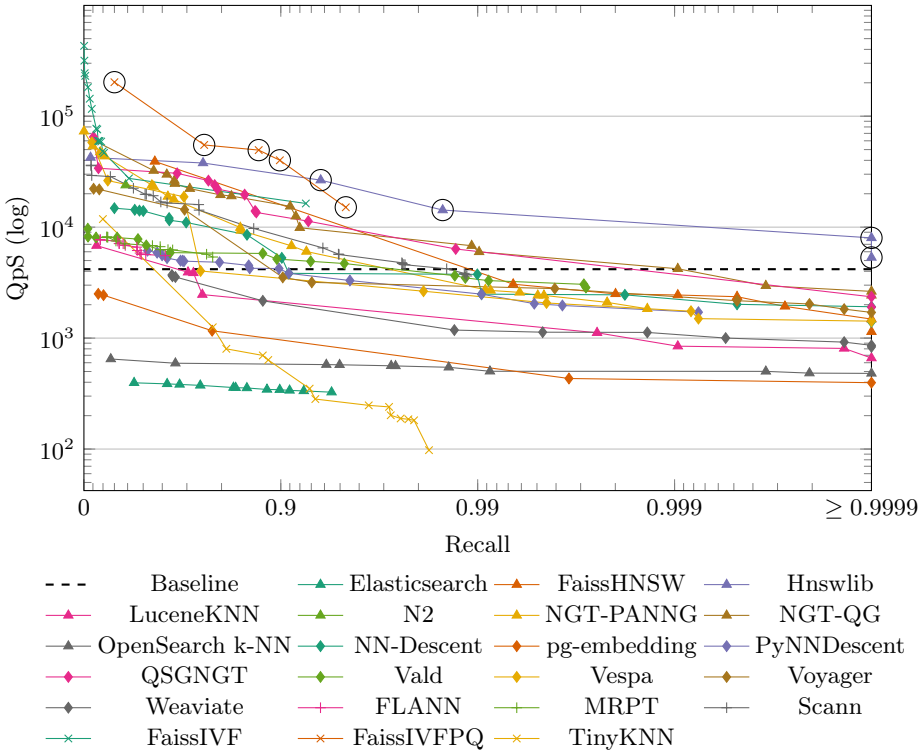

Fig. 1. The best configurations for all tested systems and the performance of the baseline on the FB15k-237 DistMult embeddings with angular distance. The best configurations for this dataset have been encircled (◯). Note that the Recall axis is not linear.

4.4 Results

Our experiment results comprise more than 750 data points per dataset.[11] In previous works, the results have been presented in Recall-QpS plots. However, these plots tend to be crowded when comparing all 22 evaluated systems as it can be seen in Figure 4.4. Instead, we depict only the Pareto front of those systems that have at least one configuration that is part of the Pareto front for a particular dataset in Figures 2 and 3. FaissIVFPQ occurs in every plot. It typically achieves very high QpS scores while the Recall is quite high as well. However, only for some datasets, the Bayesian optimization was able to find configurations of FaissIVFPQ that let to a Recall very close to 1.0. On angular-distance-based datasets, implementations like NGT-QG, QSGNGT or Hnswlib dominate this region of the plots. Many configurations of FaissIVF are

[10] A VM with 4 64-bit CPUs and 32GB RAM. Note that the VM does not have a GPU that could be used by the ANNS systems.

[11] All our results are available at `https://github.com/MichaelRoeder/ann-benchmarks/tree/main`.

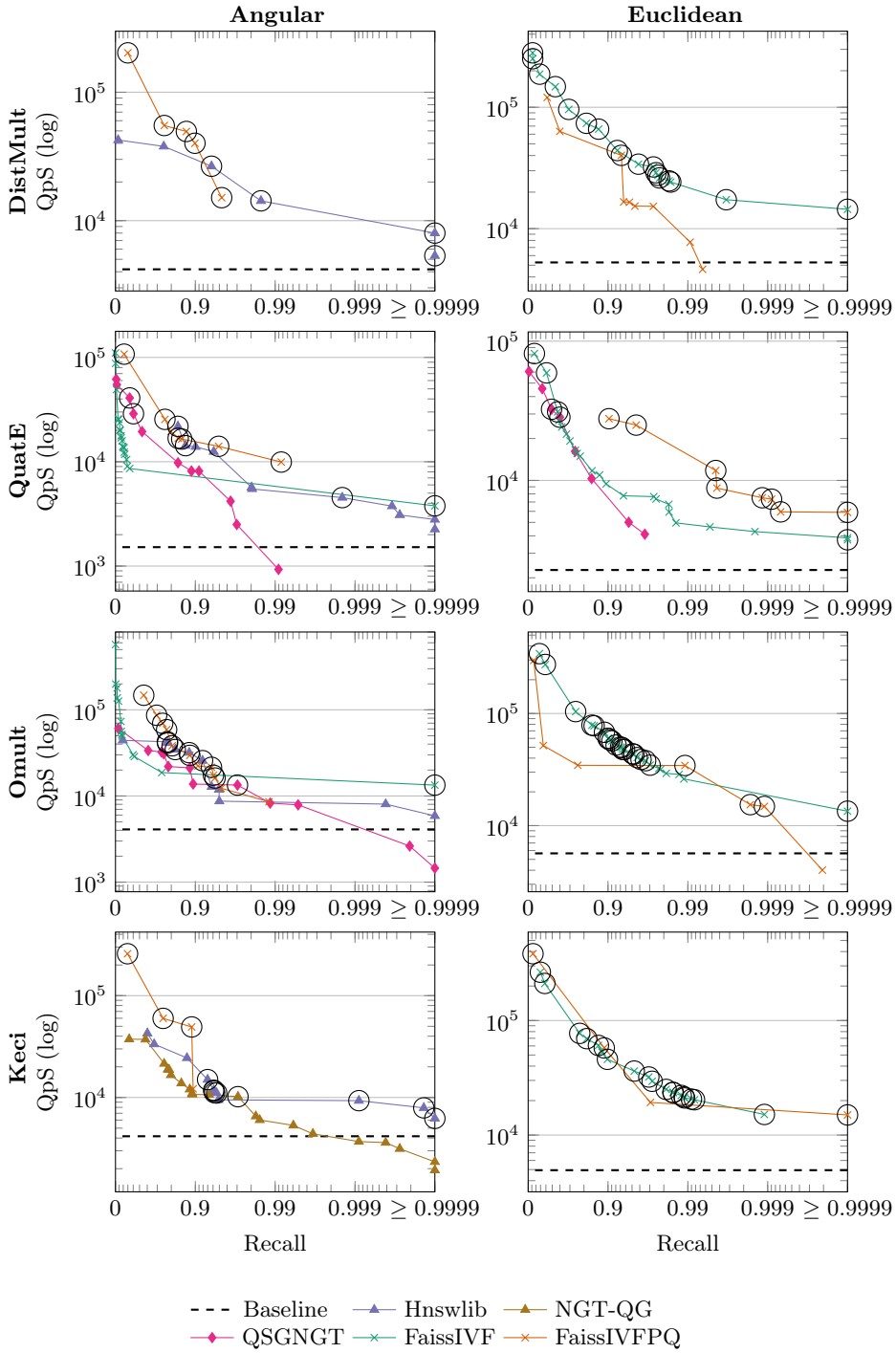

Fig. 2. The Recall-QpS curve of the best-performing systems on the FB15k-237 datasets. Angular and Euclidean datasets can be found in the left and right columns, respectively. The best configurations for the datasets have been encircled (◯).

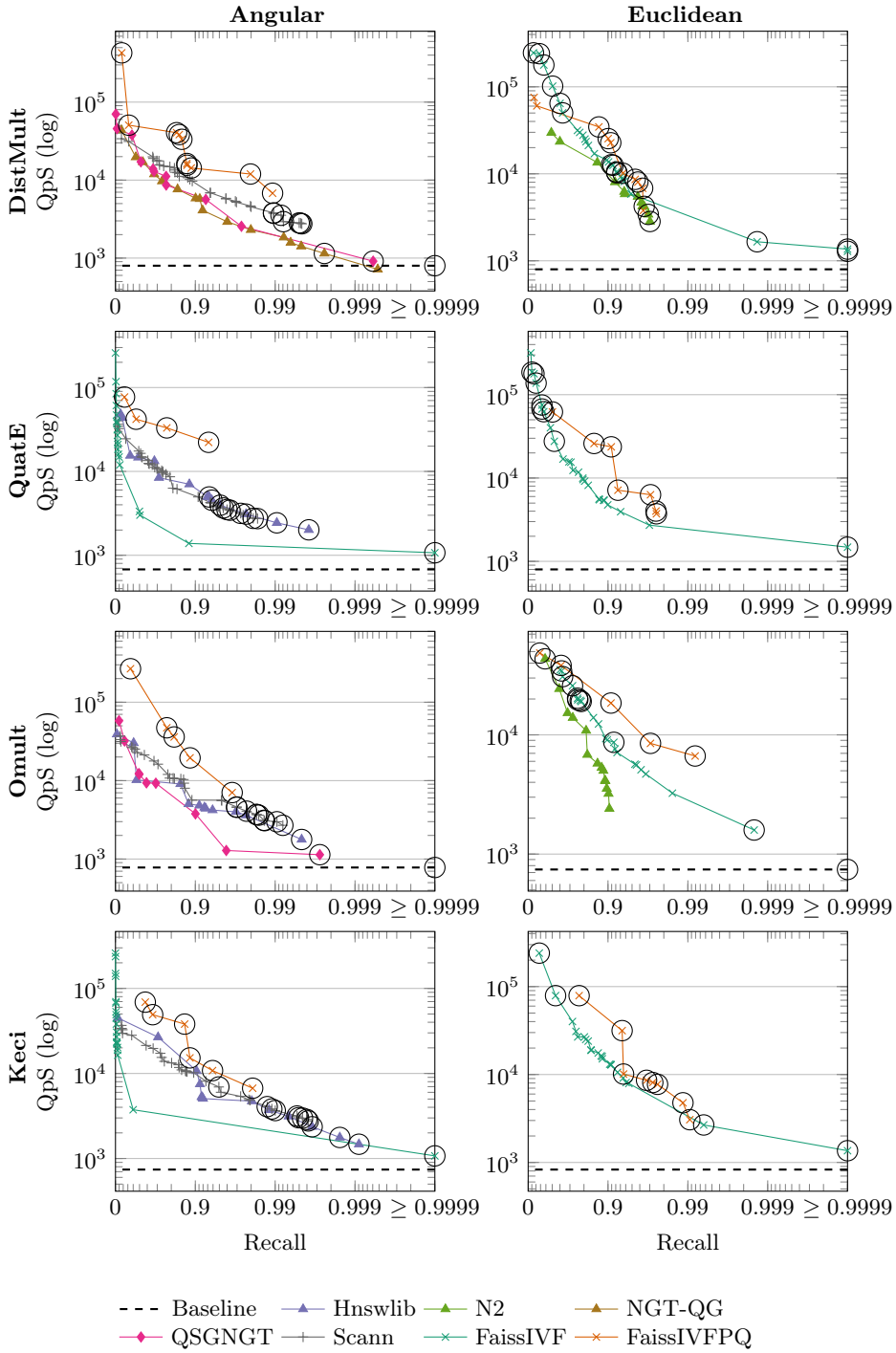

Fig. 3. The Recall-QpS curve of the best-performing systems on the Yago3-10 datasets. Angular and Euclidean datasets can be found in the left and right columns, respectively. The best configurations for the datasets have been encircled (◯).

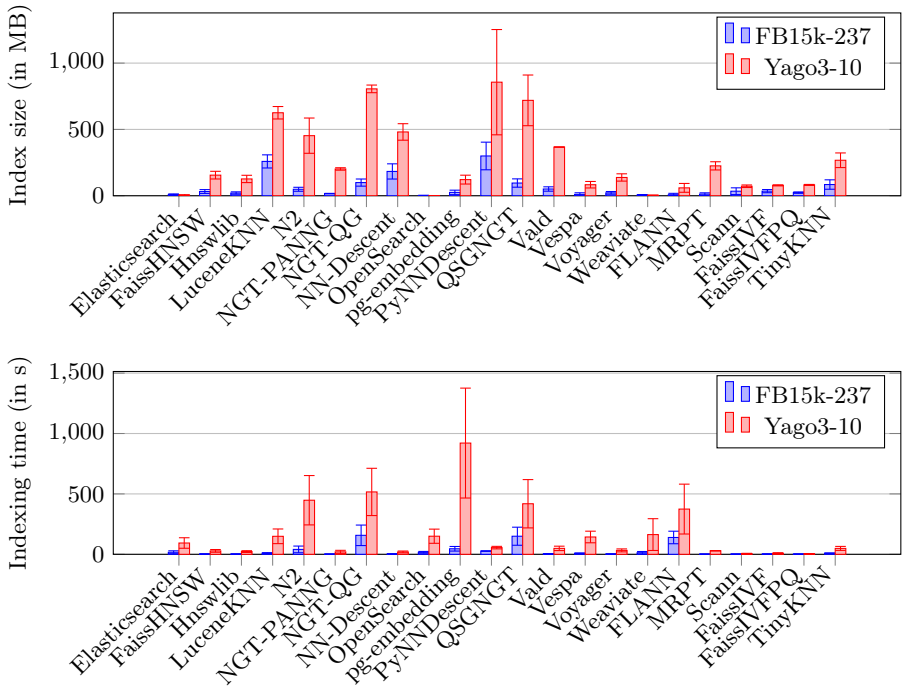

Fig. 4. The average size of the generated indexes on the two knowledge graphs (top) and the average time it took to generate the index (bottom).

not competitive when used with the angular distance. However, in half of the plots for the angular-distance-based datasets, FaissIVF occurs since it is able to achieve a Recall of 1.0, which rarely is achieved by other approaches. The plots for the Euclidean-distance-based experiments are dominated by FaissIVF and FaissIVFPQ with a similar pattern as before. In many cases, FaissIVFPQ achieves a high QpS rate and a good Recall, but FaissIVF achieves a better Recall.

Figure 4 summarizes the results with respect to the generation of the search indexes. Especially for the bigger KG Yago3-10, the systems show quite different numbers. While the majority of systems stays close or even below the 60MB of the Parquet file, in which the Knowledge Graph embeddings are provided, LuceneKNN, N2, NGT-QG, NN-Descent, PyNN-Descent, and QSGNGT create indexes with a size close to or beyond the 500MB mark. With respect to the time that is needed to generate the index, pg-embedding is the slowest system with an average runtime of 921 seconds for Yago3-10 datasets, followed by NGT-QG, N2 and QSGNGT.

Table 2. Pairwise comparisons of systems: the number of datasets on which the system in the row outperformed the system in the column. The row #Worse and the column #Better contain the sum in how many comparisons a system has been worse or better than any other system, respectively.

	BL	Elasticsearch	FaissHNSW	Hnswlib	LuceneKNN	N2	NGT-PANNG	NGT-QG	NN-Descent	OpenSearch-kNN	pg-embedding	PyNNDescent	QSGNGT	Vald	Vespa	Voyager	Weaviate	FLANN	MRPT	Scann	FaissIVF	FaissIVFPQ	TinyKNN	#Better
BL		16	0	0	3	0	0	0	0	8	6	0	0	1	0	0	6	0	0	0	0	0	4	44
Elasticsearch	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FaissHNSW	0	16		0	5	4	0	0	0	7	3	0	0	0	0	0	7	1	3	0	1	0	8	55
Hnswlib	4	16	6		14	5	2	0	4	9	5	6	0	11	9	7	11	7	6	4	1	0	11	138
LuceneKNN	0	16	0	0		0	0	0	0	6	3	0	0	0	0	0	0	0	0	0	0	0	2	27
N2	0	8	3	0	8		0	0	0	8	0	0	1	4	6	5	6	0	0	0	1	0	5	55
NGT-PANNG	0	16	0	0	4	3		0	0	6	1	0	0	0	0	0	2	4	3	0	1	0	13	53
NGT-QG	0	16	4	0	13	6	3		2	13	5	5	0	8	9	10	12	5	7	2	1	0	14	135
NN-Descent	0	16	0	0	12	1	0	0		15	8	2	0	0	0	0	9	5	6	0	1	0	16	91
OpenSearch k-NN	0	16	0	0	0	0	0	0	0		2	0	0	0	0	0	0	0	0	0	0	0	0	18
pg-embedding	0	6	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	6
PyNNDescent	0	16	0	0	7	1	0	0	0	9	0		0	0	0	0	5	2	1	0	1	0	14	56
QSGNGT	0	16	1	0	11	5	1	0	0	9	4	3		5	3	3	9	5	6	1	1	0	13	96
Vald	0	16	0	0	1	0	0	0	0	4	0	0	0		0	0	2	1	1	0	0	0	10	35
Vespa	0	16	0	0	14	0	0	0	0	9	6	0	0	0		0	10	2	2	0	1	0	11	71
Voyager	0	16	0	0	13	1	0	0	0	10	8	0	0	0	0		10	3	2	0	0	0	11	74
Weaviate	0	16	0	0	0	0	0	0	0	8	3	0	0	0	0	0		0	0	0	0	0	1	28
FLANN	3	12	0	0	5	0	0	0	0	6	3	0	0	2	0	0	4		0	0	0	0	9	44
MRPT	0	13	1	0	6	0	0	0	0	7	1	0	0	4	0	0	6	0		0	0	0	10	48
Scann	0	15	1	0	1	3	2	0	0	1	0	0	0	6	1	1	0	5	6		1	0	8	51
FaissIVF	10	12	7	7	11	4	8	8	9	12	4	8	6	9	8	8	10	8	9	7		0	11	**176**
FaissIVFPQ	1	15	8	5	8	14	6	7	2	8	0	4	6	9	7	8	8	5	8	5	2		14	150
TinyKNN	0	10	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0		11
#Worse	18	315	31	12	136	47	22	15	17	156	62	28	13	59	43	42	117	53	60	19	12	**0**	185	

5 Discussion

For a detailed discussion, we analyse the results further. To this end, we remove the configurations from the systems' Pareto curves that have a Recall below 0.1, since although many of these results have a good efficiency (i.e., their QpS is high), they lack effectiveness. This for example removes many of the points in the upper left corner of the plots in Figures 2 and 3. With these updated curves, we compare the systems pair-wise on each dataset checking whether a system is able to outperform the other. We define that a system outperforms another system on a particular dataset, if there is no configuration of the second system on the common Pareto front of both systems according to the comparison operator defined in Section 4.2. Table 2 shows a summary of these pairwise comparisons over all datasets. For each dataset, we assign ranks to the single

Table 3. Average ranks according to the pairwise comparisons across the datasets. Best average ranks are marked bold. Systems that are significantly outperformed by the best system are underlined.

System	Distance		KG		Embedding				Overall
	Angular	*Euclidean*	*FB15k-237*	*Yago3-10*	*DistMult*	*QuatE*	*Omult*	*Keci*	
BL	11.19	11.31	10.38	12.13	10.88	12.88	10.00	11.25	11.25
Elasticsearch	23.00	23.00	23.00	23.00	23.00	23.00	23.00	23.00	23.00
FaissHNSW	8.31	15.94	11.56	12.69	12.00	14.13	11.00	11.38	12.13
Hnswlib	**2.25**	7.31	4.44	5.13	5.75	4.38	4.75	4.25	4.78
LuceneKNN	17.13	20.50	18.44	19.19	18.38	18.50	19.13	19.25	18.81
N2	17.88	4.06	11.31	10.63	11.63	11.25	10.25	10.75	10.97
NGT-PANNG	9.81	11.25	9.06	12.00	9.38	8.63	13.38	10.75	10.53
NGT-QG	3.44	5.81	5.81	**3.44**	**3.25**	5.25	6.38	**3.63**	4.63
NN-Descent	4.75	7.81	7.25	5.31	4.75	7.50	6.38	6.50	6.28
OpenSearch k-NN	17.88	20.88	21.13	17.63	18.75	20.25	19.50	19.00	19.38
pg-embedding	18.81	14.31	17.38	15.75	16.88	16.13	17.13	16.13	16.56
PyNNDescent	11.69	8.75	11.06	9.38	10.88	10.25	10.63	9.13	10.22
QSGNGT	5.13	7.69	6.50	6.31	5.13	6.75	6.50	7.25	6.41
Vald	14.63	15.19	16.06	13.75	14.38	14.25	16.13	14.88	14.91
Vespa	6.81	15.69	9.94	12.56	12.50	8.88	11.38	12.25	11.25
Voyager	7.56	14.63	10.31	11.88	10.63	10.88	11.38	11.50	11.09
Weaviate	16.31	19.50	17.56	18.25	18.50	17.00	17.25	18.88	17.91
FLANN	15.69	7.94	8.94	14.69	14.13	12.13	10.75	10.25	11.81
MRPT	17.88	6.56	12.25	12.19	11.88	14.25	10.38	12.38	12.22
Scann	7.94	14.50	13.69	8.75	12.00	11.38	9.38	12.13	11.22
FaissIVF	9.25	**1.25**	**4.19**	6.31	6.50	**2.13**	6.63	5.75	5.25
FaissIVFPQ	6.94	1.75	5.00	3.69	4.13	4.75	**4.00**	4.50	**4.34**
TinyKNN	21.75	20.38	20.75	21.38	20.75	21.50	20.75	21.25	21.06

systems according to the wins and losses that they achieve in the comparison. Table 3 shows the average ranks of the systems aggregated according to the distance measure, the underlying Knowledge Graph and the KGE used for the different datasets. A Friedman Test acknowledged for all columns in Table 3 that there are significant differences in the systems ranks [12].[12] The post-hoc Nimney Test [12] showed that in each column, several systems are significantly outperformed by the best system in that column.

These results allow several conclusions. The results from the first two columns suggest that in many cases, Hnswlib and FaissIVF are good choices when the angular and Euclidean distances are used, respectively. Similarly, several sys-

[12] We use $\alpha = 0.05$ for all significance tests.

tems should not be used for the one or the other distance. For example, N2 showed a comparably good performance on two Yago3-10 datasets when used with the Euclidean distance, but was often outperformed on datasets with an angular distance. Similarly, Vespa, Voyager, and Scann are significantly worse on Euclidean datasets while they are not significantly outperformed on angular datasets. Comparing the two distance columns with the other columns shows that the large differences of some of the average ranks that are observed in the first two columns do not repeat. This suggests that the distance measure could be the most important feature when deciding which system to take. However, especially with larger knowledge graphs, the growth of the index size and indexing time depicted in Figure 4 could become important for several algorithms that achieved good ranks, like N2, NGT-QG or QSGNGT. The four columns of the embeddings only show smaller differences between each other, suggesting that there is only a minor influence of the embedding algorithm on the systems' performance.

Another interesting insight comes from the comparison of the systems that rely on the same ANNS algorithm (see Table 1). For example, Hnswlib is the only HNSW-based implementation in our evaluation that is not significantly outperformed on either angular or Euclidean datasets. Other implementations are either outperformed in one, or like Elasticsearch, LucenKNN, OpenSearch k-NN, pg-embedding, and Weaviate, in both categories. Overall, these last 5 systems together with Vald and TinyKNN perform significantly worse in our evaluation than FaissIVFPQ. The latter achieves the best average rank over all 16 datasets. Hence, we can conclude that FaissIVFPQ can serve as an interesting alternative for Hnswlib and FaissIVF.

6 Conclusion

In this work, we carried out a detailed comparison of 22 Approximate Nearest Neighbor Search systems on 16 KGE-based datasets—a type of data that has been overlooked so far in evaluations of these systems. In contrast to the state of the art, we fine-tuned each approach in each experiment by using Bayesian optimization to ensure the fairness of our experiments. The evaluated systems showed significant performance differences in these experiments. Our results suggest that the overall performance of approaches with respect to runtime and recall is contingent upon the similarity measure used to compare embeddings.

Our future work will focus on the best performing systems in this evaluation. While our 16 datasets are based on FB15k-237 and Yago3-10—two Knowledge Graphs that are widely adopted in the research on KGE algorithms—we plan to include larger knowledge graphs to further look into the influence of the Knowledge Graph size on the system's performance.

Acknowledgments. This work has been supported by the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW) within the project SAIL (NW21-059D) and the Lamarr Fellow Network funded project WHALE (LFN 1-04).

References

1. Alshahrani, M., Thafar, M.A., Essack, M.: Application and evaluation of knowledge graph embeddings in biomedical data. PeerJ. Computer science (2021). https://doi.org/10.7717/peerj-cs.341
2. Aumüller, M., Bernhardsson, E., Faithfull, A.: ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. Information Systems **87**, 101374 (2020). https://doi.org/10.1016/j.is.2019.02.006, https://www.sciencedirect.com/science/article/pii/S0306437918303685
3. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (Sep 1975). https://doi.org/10.1145/361002.361007, https://doi.org/10.1145/361002.361007
4. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**(null), 281–305 (Feb 2012)
5. Bordes, A., Chopra, S., Weston, J.: Question answering with subgraph embeddings. In: Moschitti, A., Pang, B., Daelemans, W. (eds.) Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 615–620. Association for Computational Linguistics, Doha, Qatar (Oct 2014). https://doi.org/10.3115/v1/D14-1067, https://aclanthology.org/D14-1067
6. Bordes, A., Weston, J., Usunier, N.: Open question answering with weakly supervised embedding models. In: Calders, T., Esposito, F., Hüllermeier, E., Meo, R. (eds.) Machine Learning and Knowledge Discovery in Databases. pp. 165–180. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
7. Cao, Z., Long, M., Wang, J., Yu, P.S.: Hashnet: Deep learning to hash by continuation. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (Oct 2017)
8. Dai, Y., Wang, S., Xiong, N.N., Guo, W.: A survey on knowledge graph embedding: Approaches, applications and benchmarks. Electronics **9**(5) (2020). https://doi.org/10.3390/electronics9050750, https://www.mdpi.com/2079-9292/9/5/750
9. Dasgupta, S., Freund, Y.: Random projection trees and low dimensional manifolds. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing. p. 537–546. STOC '08, Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1374376.1374452, https://doi.org/10.1145/1374376.1374452
10. Demir, C., Moussallem, D., Heindorf, S., Ngonga Ngomo, A.C.: Convolutional hypercomplex embeddings for link prediction. In: Balasubramanian, V.N., Tsang, I. (eds.) Proceedings of The 13th Asian Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 157, pp. 656–671. PMLR (17–19 Nov 2021), https://proceedings.mlr.press/v157/demir21a.html
11. Demir, C., Ngonga Ngomo, A.C.: Clifford embeddings–a generalized approach for embedding in normed algebras. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 567–582. Springer (2023)
12. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (Dec 2006)
13. Dong, W., Moses, C., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In: Proceedings of the 20th International Conference on World Wide Web. p. 577–586. WWW '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/1963405.1963487, https://doi.org/10.1145/1963405.1963487

14. Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.E., Lomeli, M., Hosseini, L., Jégou, H.: The Faiss library (2024)
15. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An Algorithm for Finding Best Matches in Logarithmic Expected Time. ACM Trans. Math. Softw. **3**(3), 209–226 (Sep 1977). https://doi.org/10.1145/355744.355745, https://doi.org/10.1145/355744.355745
16. Fu, C., Zhou, M., Xuan, Q., Hu, H.X.: Expert recommendation in oss projects based on knowledge embedding. In: 2017 International Workshop on Complex Systems and Networks (IWCSN). pp. 149–155 (2017). https://doi.org/10.1109/IWCSN.2017.8276520
17. Fukunaga, K., Narendra, P.: A Branch and Bound Algorithm for Computing k-Nearest Neighbors. IEEE Transactions on Computers **C-24**(7), 750–753 (1975). https://doi.org/10.1109/T-C.1975.224297
18. Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., Kumar, S.: Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 3887–3896. PMLR (13–18 Jul 2020), https://proceedings.mlr.press/v119/guo20h.html
19. Hyvönen, V., Pitkänen, T., Tasoulis, S., Jääsaari, E., Tuomainen, R., Wang, L., Corander, J., Roos, T.: Fast nearest neighbor search through sparse random projections and voting. In: Big Data (Big Data), 2016 IEEE International Conference on. pp. 881–888. IEEE (2016)
20. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing. p. 604–613. STOC '98, Association for Computing Machinery, New York, NY, USA (1998). https://doi.org/10.1145/276698.276876, https://doi.org/10.1145/276698.276876
21. Iwasaki, M.: Proximity search in metric spaces using approximate k nearest neighbor graph. IPSJ Trans. Database **3**(1), 18–28 (2010)
22. Iwasaki, M.: Pruned bi-directed k-nearest neighbor graph for proximity search. In: Amsaleg, L., Houle, M.E., Schubert, E. (eds.) Similarity Search and Applications. pp. 20–33. Springer International Publishing, Cham (2016)
23. Iwasaki, M.: Fusion of graph-based indexing and product quantization for ANN search. Medium (online) (April 2023), https://medium.com/@masajiro.iwasaki/fusion-of-graph-based-indexing-and-product-quantization-for-ann-search-7d1f0336d0d0, accessed on december 5th, 2024.
24. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with gpus. IEEE Transactions on Big Data **7**(3), 535–547 (2021). https://doi.org/10.1109/TBDATA.2019.2921572
25. Li, C., Zhang, M., Andersen, D.G., He, Y.: Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. p. 2539–2554. SIGMOD '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3318464.3380600, https://doi.org/10.1145/3318464.3380600
26. Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., Lin, X.: Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement. IEEE Transactions on Knowledge and Data Engineering **32**(8), 1475–1488 (2020). https://doi.org/10.1109/TKDE.2019.2909204

27. Malkov, Y.A., Yashunin, D.A.: Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence **42**(4), 824–836 (2020). https://doi.org/10.1109/TPAMI.2018.2889473
28. Muja, M., Lowe, D.G.: Scalable Nearest Neighbor Algorithms for High Dimensional Data. Pattern Analysis and Machine Intelligence, IEEE Transactions on **36** (2014)
29. Naidan, B., Boytsov, L., Nyberg, E.: Permutation search methods are efficient, yet faster search is possible. CoRR **abs/1506.03163** (2015), http://arxiv.org/abs/1506.03163
30. Nogueira, F.: Bayesian Optimization: Open source constrained global optimization tool for Python (2014), https://github.com/bayesian-optimization/BayesianOptimization, last accessed on December 17, 2024.
31. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., de Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (2016). https://doi.org/10.1109/JPROC.2015.2494218
32. Sun, P., Simcha, D., Dopson, D., Guo, R., Kumar, S.: SOAR: Improved Indexing for Approximate Nearest Neighbor Search. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems. vol. 36, pp. 3189–3204. Curran Associates, Inc. (2023), https://proceedings.neurips.cc/paper_files/paper/2023/file/0973524e02a712af33325d0688ae6f49-Paper-Conference.pdf
33. Wang, G., Ke, X., Hu, J., Li, Q., Shao, M., Fan, J.: Learning to prune: General and efficient approximate nearest neighbor search with direction navigating graph. In: Proceedings of the 2022 5th International Conference on Data Storage and Data Engineering. p. 50–58. DSDE '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3528114.3528123, https://doi.org/10.1145/3528114.3528123
34. Wang, M., Xu, X., Yue, Q., Wang, Y.: A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. Proc. VLDB Endow. **14**(11), 1964–1978 (Jul 2021). https://doi.org/10.14778/3476249.3476255, https://doi.org/10.14778/3476249.3476255
35. Wu, J., Chen, X.Y., Zhang, H., Xiong, L.D., Lei, H., Deng, S.H.: Hyperparameter optimization for machine learning models based on bayesian optimizationb. Journal of Electronic Science and Technology **17**(1), 26–40 (2019). https://doi.org/https://doi.org/10.11989/JEST.1674-862X.80904120, https://www.sciencedirect.com/science/article/pii/S1674862X19300047
36. Yang, B., Yih, S.W.t., He, X., Gao, J., Deng, L.: Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In: Proceedings of the International Conference on Learning Representations (ICLR) 2015 (May 2015), https://www.microsoft.com/en-us/research/publication/embedding-entities-and-relations-for-learning-and-inference-in-knowledge-bases/
37. Zhang, F., Yuan, N.J., Lian, D., Xie, X., Ma, W.Y.: Collaborative Knowledge Base Embedding for Recommender Systems. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 353–362. KDD '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939673, https://doi.org/10.1145/2939672.2939673
38. Zhang, S., Tay, Y., Yao, L., Liu, Q.: Quaternion knowledge graph embeddings. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran

Associates, Inc. (2019), https://proceedings.neurips.cc/paper_files/paper/2019/file/d961e9f236177d65d21100592edb0769-Paper.pdf