GraSP: Optimizing Graph-based Nearest Neighbor Search with Subgraph Sampling and Pruning

Minjia Zhang, Wenhan Wang, Yuxiong He
Microsoft
Bellevue, WA, USA
{minjiaz,wenhanw,yuxhe}@microsoft.com

ABSTRACT

Nearest Neighbor Search (NNS) has recently drawn a rapid growth of interest because of its core role in high-dimensional vector data management in data science and AI applications. The interest is fueled by the success of neural embedding, where deep learning models transform unstructured data into semantically correlated feature vectors for data analysis, e.g., recommending popular items. Among several categories of methods for fast NNS, graph-based approximate nearest neighbor search algorithms have led to the best-in-class search performance on a wide range of real-world datasets. While prior works improve graph-based NNS search efficiency mainly through exploiting the structure of the graph with sophisticated heuristic rules, in this work, we show that the frequency distributions of edge visits for graph-based NNS can be highly skewed. This finding leads to the study of pruning unnecessary edges to avoid redundant computation during graph traversal by utilizing the query distribution, an important yet under-explored aspect of graph-based NNS. In particular, we formulate graph pruning as a discrete optimization problem, and introduce a graph optimization algorithm GraSP that improves the search efficiency of similarity graphs by learning to prune redundant edges. GraSP enhances an existing similarity graph with a probabilistic model. It then performs a novel subgraph sampling and iterative refinement optimization to explicitly maximize search efficiency when removing a subset of edges in expectation over a graph for a large set of training queries. The evaluation shows that GraSP consistently improves the search efficiency on real-world datasets, providing up to 2.24X faster search speed than state-of-the-art methods without losing accuracy.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Vector management and search, search efficiency, graph sampling

WSDM ’22, February 21–25, 2022, Tempe, AZ, USA

ACM ISBN 978-1-4503-9132-0/22/02…$15.00
https://doi.org/10.1145/3488560.3498425

ACM Reference Format:
Minjia Zhang, Wenhan Wang, Yuxiong He. 2022. GraSP: Optimizing Graph-based Nearest Neighbor Search with Subgraph Sampling and Pruning. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM ’22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3488560.3498425

1 INTRODUCTION

Nearest neighbor search has been the core of canonical learning algorithms such as non-parametric classification/regression. The problem has recently become the focus of intense research activity due to its role in supporting semantic-based search of unstructured data such as images, texts, video, and speech using neural embedding models. In semantic-based search, the unstructured entities are embedded as dense continuous vectors such that the similarity between entities is expressed as the distance (e.g., Euclidean) between their embeddings [45, 47, 51, 53]. During inference, the search query is embedded into the same high dimensional space, and the application or service returns the entities whose embeddings are nearest to the embedded search query [40]. For example, e-commerce players such as Amazon [41] and Alibaba [56] build semantic-based search engines, which embed product catalog and the search query into high-dimensional vectors and recommends products whose embeddings that are closest to the embedded search query; Youtube [17] embeds videos to vectors for video recommendation; Web-scale search engines embed text (e.g., word2vec [37], doc2vec [30]) and images (e.g., VGG [47]) for text/image retrieval [16, 49]. Due to the success and continual advancement of neural embedding techniques that effectively capture the semantic relations of objects, we expect applications built on top of the embedding-based search to continue growing in the future.

A big challenge of the aforementioned applications is on performing fast near neighbor search, because the search happens for every query, and applications such as web search and recommendations are interactive and need to return near neighbors within a few or tens of milliseconds to avoid degrading user satisfaction and revenue [22]. Moreover, interactive services often need to handle massive request volumes and could require hundreds and even thousands of machines for serving similarity search: serving efficiency and thus the cost (e.g., the ownership cost and energy cost) are also of crucial importance.

A vast amount of theoretical and empirical literature has been proposed for fast near neighbor search, ranging from space partitioning [8] to hardware-based solutions [28, 54]. Among them, graph-based approaches such as HNSW [36] have emerged as a remarkably effective approach and achieved the best-in-class search performance on real-world datasets [20, 21, 32], outperforming existing algorithms such as tree-based approaches [39, 52] and locality-sensitive hashing [6]. For example, four of the top five search libraries on well-established ANN benchmarks use graph indices [9]. Graph indices have also been integrated with many large-scale production systems [3, 16, 48–50]. Therefore, optimizing graph-based near neighbor search has a clear practical value.

To improve search efficiency, existing methods often build a similarity graph by exploiting graph structures with heuristic rules. For example, several state-of-the-art methods [25, 36] builds a graph by iteratively adding nodes to a partially constructed graph. In this process, existing methods refine the graph by adding diversified edges (e.g., adding short-range links to create densely connected local clusters and long-range links that connect those clusters) and removing unnecessary edges (e.g., so that a node does not exceed a pre-defined upper bound out-degree) to improve the navigability of the graph. Different methods [24, 25, 36] employ slightly different heuristics to accomplish this, but overall they are driven by the same principles. Despite showing promising results, one subtle issue of this type of approach is that the heuristics do not directly optimize the index structure for online search efficiency. As a result, different heuristics can lead to superior results on some datasets but worse performance on others.

Since many online services often receive hundreds of thousands of queries per day, which are available in abundance (e.g., in query log), it seems natural to ask, "can we improve the near neighbor search efficiency by learning from existing queries?" – an important yet surprisingly under-explored aspect for the ANN search task. To answer this question, we formulate the graph pruning process as a discrete optimization problem towards maximizing search efficiency for a large set of training queries when a subset of edges are removed from the original graph. To solve this problem, we introduce a new graph optimization algorithm for nearest neighbor search (**GraSP**: **Gra**ph **S**ampling and **P**runing), that efficiently improve the search efficiency of similarity graphs by learning to prune redundant edges. GraSP enhances an existing similarity graph with a probabilistic model called Annealable Similarity Graph (ASG). GraSP then performs a novel subgraph sampling and iterative refinement approach to learn important edges based on the joint probability of individual edges that maximize the accuracy and minimize the search complexity in expectation over graphs.

Specifically, the contributions of our paper consist of (1) a preliminary analysis that reveals the challenges and opportunities from the existing graph-based ANN search. (2) A novel probabilistic model called **A**nnealable **S**imilarity **G**raph (ASG) for similarity graphs, which makes it easy to learn an *edge probability* for each edge that indicates whether an edge can be pruned without hurting search efficiency. (3) A novel subgraph sampling and iterative refinement optimization method which efficiently learns an important subset of edges of a graph to reduce ANN search latency without sacrificing accuracy. (4) A theoretical proof of the correctness of our optimization. (5) A detailed evaluation of the proposed method on real-world benchmarks. The evaluation results show that GraSP enhances existing graph-based ANN search algorithms such as HNSW to achieve up to 1.49x speedups and is up to 2.24x faster than NSG in search time without sacrificing accuracy.

2 RELATED WORK

The literature on the nearest neighbor search is vast, and hence, we restrict our attention to the most relevant works here. Earlier works on ANN indexing mostly focus on space partitioning based methods, which partition the vector space and index the resulting sub-spaces for fast retrievals, such as KD-Tree [14], R*-Tree [13], and Randomized KD-Tree [39]. However, the complexity of these methods becomes not more efficient than a brute-force search as the dimension becomes large (e.g., >15) [31]. Therefore, they perform poorly on embedding vectors that are more than a few tens of dimensions. Prior works have also devoted extensive efforts over locality-sensitive hashing [5, 7]. These methods have solid theoretical foundations and allow us to estimate the search tim@e or the probability of successful search. However, LSH and similar approaches have been designed for large sparse vectors with hundreds of thousands of dimensions, not dense continuous vectors with at most a few hundreds of dimensions. As a result, graph-based approaches outperform LSH-based methods by a large margin on large-scale datasets [29, 32, 36].

A separate line of research involves compression, which computes compressed representations of points to make computing the approximate distance quickly while saving memory (e.g., compact Hamming codes [43] and product quantization [27, 29, 42]). Compression can be combined with indexing approaches to save memory while providing fast ANN search [12, 48, 55]. Although compression-based methods achieve outstanding memory savings, they are sensitive to quantization errors and result in poor recall@1 accuracy on large datasets [20, 32].

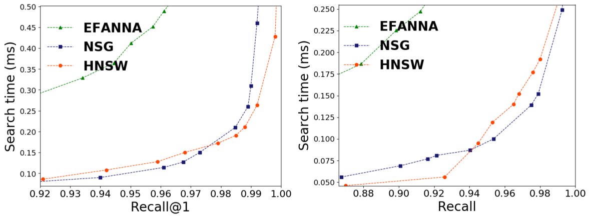
More recently, Malkov and Yashunin found that graphs that satisfy the *Small World* property are good candidates for *best-first search*. They introduce Hierarchical Navigable Small World (HNSW) [36], which iteratively builds a hierarchical k-NN graph with randomly inserted long-range links to enhance graph connectivity. For each query, it then performs a walk, which eventually converges to the nearest neighbor in logarithmic complexity. Subsequently, Fu et al. proposed NSG, which approximates *Monotonic Relative Neighbor Graph* (MRNG) [25] that also involves long-ranged links for enhancing connectivity. To the best of our knowledge, both HNSW and NSG are considered as the state-of-the-art methods for ANN search [21, 32] and have been adopted by major players in the industry [4, 34, 48].

3 OBSERVATIONS AND OPPORTUNITIES

This section presents our observation that motivated the design of the approach introduced in Section 4. We conduct the evaluations on SIFT1M [33] and Deep1M [10].

Optimizing graphs based on heuristics? First, we carry out an analysis by comparing ANN graphs that apply different heuristics in graph construction. The goal is to measure if a certain heuristic consistently works better than others. Fig. 1 shows the accuracy-vs-latency between EFANNA [23], HNSW [36], and NSG [25]. We observe that HNSW and NSG consistently perform better than EFANNA, one of the best-performing k-NN graph-based methods, for a wide range of recall regimes. We find out that this is because k-NN graphs have poor performance on clustered data (the graph has a high probability of being disconnected). In contrast, both HNSW and NSG contain heuristics that increase the graphs' connectivity, ensuring the other points are reachable from a given starting point. These results suggest that diversification of edges is indeed crucial for improving search efficiency. Interestingly, we observe that although NSG is faster than HNSW on SIFT1M in the < 0.98 recall range; HNSW achieves better search efficiency than NSG when recall > 0.98. Similarly, NSG outperforms HNSW on Deep1M in the > 0.95 recall range but is slower when recall < 0.94.

Our hypothesis is that since neither HNSW nor NSG explicitly maximizes the online search latency-vs-accuracy when constructing graphs, there might be redundant edges in a graph that can negatively affect search efficiency. Specifically, edges that do not help reduce the search path length can negatively increase the neighbor inspection cost because they create redundant computations. As a result, neither method consistently achieves superior performance on both datasets. We conjecture that by removing redundant computations, the ANN search efficiency can be significantly improved.

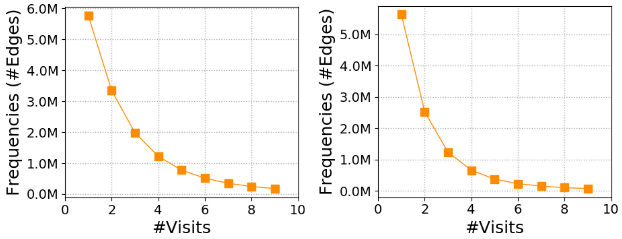


(a) SIFT1M **(b) Deep1M**

Figure 1: Comparison of EFANNA, HNSW, and NSG in accuracy-vs-latency. The results show that neither method consistently achieves a superior performance results.

Highly skewed query distribution. Our second analysis measures the frequency of visits on all edges. The goal is to evaluate the load-balancing property of ANN graphs. We randomly sample 100K vectors from the corresponding learning sets of SIFT1M and Deep1M as queries. Fig. 2 shows that the bottom layer of the HNSW graph has a highly skewed distribution. The x-axis shows the number of visits, and the y-axis indicates the frequencies of each visit count. The majority of edges are visited only once, and the frequency roughly follows an exponential decay curve, where only a small number of edges are highly visited. We further find that the skewed distribution is caused by the fact that some hub nodes (e.g., nodes with high in-degree) are much more likely to be visited than others.

Existing graph-based ANN algorithms often ignore this query distribution information, which can be exploited for improving ANN search efficiency. For example, existing graph-based methods often need to decide an upper bound of out-degree M for all nodes. Without query distribution information, it is challenging to decide how to place such an upper bound to the out-degree M: (a) a large M tends to increase the number of neighbors to examine at a local point but may decrease the search path length; (b) a small M gives the opposite tradeoff. More importantly, it seems suboptimal to place a fixed upper bound given that there are indeed "hub" nodes in the graph that require more connections. Therefore, one optimization opportunity is to let each node adaptively select its own out-degree, helping avoid redundant computation.



(a) SIFT1M **(b) Deep1M**

Figure 2: The frequency distributions of visits to edges. The results show that the distributions are highly skewed.

4 GraSP: LEARNING TO OPTIMIZE GRAPHS

In this section, we first provide the overview of our methods (Fig. 3). Then we will describe the details of its major components.

Problem Definition. Considering a set of N d-dimensional embedding vectors $X = \{x_1, ..., x_N\}$, the graph-based nearest neighbor search problem aims to build a directed graph $G = (V, E)$, where each node $v_i \in V$ corresponds to a vector x_i, and each edge $e_j \in E$ represents a relative distance measure (e.g., Euclidean distance) between two node, such that for a given set of query vectors $Y = \{y_1, ..., y_M\}$, the time to retrieve their nearest neighbors in X for a target recall is minimum.

We propose GraSP to tackle this problem, which contains three major phases.

Stage 1: Probabilistic graph construction. To begin with, we first introduce a novel probabilistic model called Annealable Similarity Graph (ASG) (Section 4.1), where we associate each edge of a graph with a learnable *edge probability* that indicates whether to keep or remove that edge. This new representation allows framing edge connectivity refinement as an optimization problem towards maximizing search efficiency for a large set of training queries. ASG can be defined on any existing similarity graphs, e.g., those constructed with existing heuristic-based methods such as HNSW [36].

Stage 2: Learning edge importance via subgraph sampling and iterative refinement. To quantitatively measure the edge importance for different queries, we model edge importance as the *robustness* of graph search efficiency to *edge removal*. Such a strategy allows us to define an objective function that reflects the loss of search efficiency, which is quantified by the distance error for a query from searching an induced subgraph (G' in $\boxed{2.a}$) and the full graph (G^* in $\boxed{2.b}$) within a search budget. To optimize the objective function, we introduce a novel subgraph sampling and iterative refinement approach ($\boxed{2.c}$, $\boxed{2.d}$ in Fig. 3). In the beginning, all edges have equal probabilities (edge weights here are uninformative). During the optimization, GraSP enables exploration and exploitation of edge probabilities and iteratively refine edge probabilities to create an ensemble of refined subgraphs.

Stage 3: Final pruning. In this step, we select a small but critical set of edges to form the final similarity graph and prune the remaining edges by masking them out. The final refined graph can then be used for answering queries.

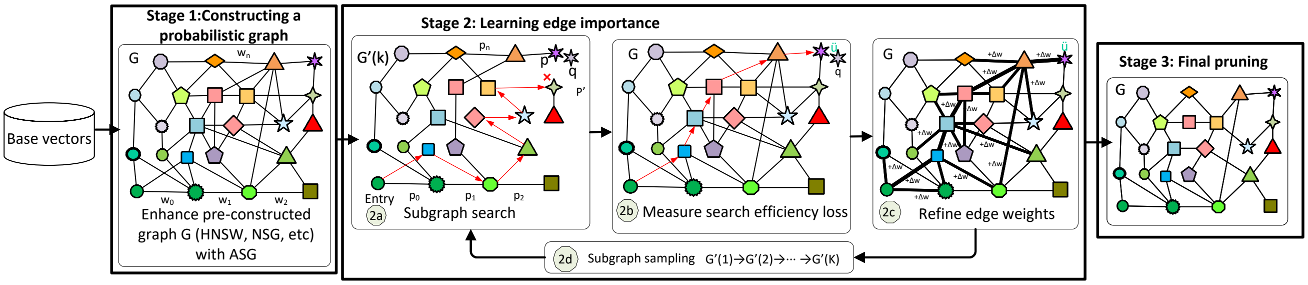

Base vectors
Stage 1:Constructing a probabilistic graph
G
w_n
w_0
w_1
w_2
Enhance pre-constructed graph G (HNSW, NSG, etc) with ASG
Stage 2: Learning edge importance
G'(k)
p_n
p
q
P'
Entry
p_0
p_1
p_2
2a Subgraph search
G
q
2b Measure search efficiency loss
G
+Δw
2c Refine edge weights
2d Subgraph sampling G'(1)→G'(2)→··· →G'(K)
Stage 3: Final pruning
G

Figure 3: Overview of GraSP, which consists of three stages: 1. Constructing a probabilistic model for a similarity graph; 2. Learning edge importance via subgraph sampling and iterative refinement; 3. Pruning edges based on learned edge importance.

4.1 Subgraph Sampling and Iterative Refinement

This section introduces the proposed GraSP approach in detail.

A probabilistic model for similarity graph. Traditionally, similarity graphs contain only structure information regarding how nodes are connected by edges. Here, we introduce a novel probabilistic model for similarity graphs — *Annealable Similarity Graph.*

Definition 4.1. *(Annealable Similarity Graph) Given a similarity graph $G(V, E)$, an Annealable Similarity Graph $G(V, E, \mathcal{W})$ is obtained by associating each edge $e \in E$ with a weight variable $w_e \in \mathcal{W}$, and a* transformation function *$p : E \longrightarrow (0, 1)$ such that $p(e) \equiv p_e$ indicates the edge probability of e. That is, an independent Bernoulli random variable R_e, where $\mathbb{P}(R_e = 1) = p_e, \mathbb{P}(R_e = 0) = 1 - p_e$ is assigned to each e, which decides whether e should exist in the graph.*

Such a probabilistic model allows sampling of subgraphs in continuous space, and with associated probabilities. Intuitively, an edge probability p_e should be: (i) monotonically increasing as w_e increases; and (ii) $\lim_{w_e \to +\infty} p_e = 1$, and $\lim_{w_e \to -\infty} p_e = 0$. More importantly, it is desirable to have w_e initialized to similar values for all edges, allowing each edge to have an equal probability of consideration when there is little information about the edge probability. As there are more about edge information, p_e should converge into a degenerated distribution that allows identifying a subset of edges that maximizes search efficiency.

To satisfy the above conditions, we introduce the following *transformation function* p for ASG:

$$p_e(T) = \frac{1}{1 + \exp\left(-\frac{w_e + \mu}{T}\right)} \tag{1}$$

where μ is a normalizing factor to keep $\sum_{e \in E} p_e(T_n) = C$ a constant and $T \in (0, \infty)$ is the *temperature*, which smooths the probabilities p_e as following. If $T \longrightarrow \infty$, the probabilities p_e converges uniformly to the same value regardless of edge e; on the other hand, if $T \longrightarrow 0$, the probabilities p_e converges to either 1 or 0, respectively. Mathematically, our definition of the transformation function implies that the joint distribution of $R_e(T)$ for all edges e should be equal for $T \longrightarrow \infty$, which corresponds to *exploration*, and sparsely supported for $T \longrightarrow 0$, which corresponds to *exploitation*. We note that the above definition is closely related to the Fermi-Dirac distribution [18], which use state energies, temperature, and total chemical potential to model a physic system.

With the probabilistic model, we still need to answer the question: whether we can measure how removing or keeping an edge would affect search efficiency, and if so, how to quantitatively obtain the loss of search efficiency?

Modeling edge importance. In this work, we consider modeling edge importance of proximity graph with respect to search efficiency as *the robustness of search efficiency against edge removal*. We do so by stochastically deleting each edge e with probability $1 - p_e$, and we denote the resulting random graph as $G'(V, E')$, where E' is the set of selected edges. For any query q, if we find the exact nearest neighbor in G but not in G' under a search budget, then we treat the *edge hops* H^q (i.e., the set of edges traversed along the search path in G rather than the full set of edges that have been inspected) of q in G to be important for preserving the search efficiency, because it is their deletion that causes the failure to find the nearest neighbor in G'.

We remark that the above way of modeling edge importance was first introduced in percolation theory [15], to study the phase transition of a physical system when one or more of its properties change abruptly after a slight change in controlling variables. It was then applied to model adversarial attacks to complex networks such as Internet [46]. To our best knowledge, we are the first to apply it for modeling edge importance in graph-based ANN search.

Search-efficiency-aware objective function. Once we get the set of edge hops for a query q (denote as H^q), we define the loss for query q as how far off the found candidate is relative to its true nearest neighbor, which is remotely similar to the teacher-student loss function in the Knowledge Distillation scheme [26]. In particular, assume G answers the query q by returning a point p, and G' answers q with a candidate p', we define the search-efficiency-aware loss over a training set Q as:

$$L(Q, G, G') = \frac{1}{|Q|} \sum_{q \in Q} \mathbb{E}\left[L(q, G, G')\right] \tag{2}$$

$$L(q, G, G') = \frac{\delta\langle p', q\rangle}{\delta\langle p, q\rangle} - 1 \tag{3}$$

where $\delta\langle\cdot,\cdot\rangle$ represents the distance between two nodes. The loss measures the delta of q's nearest neighbor in a subgraph G' versus in the full graph G.

Learning edge importance via weighted sampling. At each iteration, the algorithm will (1) increase the weight of an edge based on its contribution to the loss and (2) normalize all edge weights (i.e., the weight of an unimportant edge will be decreased). To reduce the loss, we update the weight w_h of $h \in H^q$ by increasing their weights with Δw_h, which is defined as:

$$\Delta w_h = \left(\frac{\delta\langle p', q\rangle}{\delta\langle p, q\rangle} - 1\right) \cdot \eta, \forall h \in H^q \tag{4}$$

where η represents the learning rate. The idea is that when the returned candidate p' in G' is the exact nearest neighbor as q returned by G, then the deleted edges are less important and Δw_h is 0. Otherwise, the larger $\delta\langle p', q\rangle$ in comparison to $\delta\langle p, q\rangle$, the more importance H^q indicates, and the edge weights of H^q should increase more such that H^q shall have a higher probability being sampled (i.e., being more important for preserving search efficiency).

At learning time, we would like to obtain a subgraph G' from G by sampling a set of expected $\mathbb{E}[|E'|] = \lceil \lambda \cdot |E| \rceil$ edges based on their edge weights, where λ represents a sampling ratio. One challenge would be that the number of edges of a random subgraph $G'(k)$ follows a Poisson binomial distribution: It is the sum of Bernoulli random variables $R_e, e = 1, ..., |E|$, each taking on values 0 and 1 with probabilities $1 - p_e$ and p_e, respectively. However, since $\Delta w \geq 0$ in Eq 4, how to expect the sampled subgraph $G'(k)$ to have $\mathbb{E}[|E'(k)|]$ edges given that increased weights also increase the sum of R_e? To address this challenge, we do a *binomial normalization* to adjust the edge weights at the beginning of each iteration by adding a normalizing factor $\mu(k)$ (as in Eq 1) to all edge weights so that the sum of R_e equals to $|E'(k)|$:

$$|E'(k)| \equiv \mathbb{E}\left[\sum_{e \in E} R_e\right] = \sum_{e \in E} p_e(T) = \sum_{e \in E} \frac{1}{1 + e^{-\frac{w_e + \mu(k)}{T}}} \tag{5}$$

where $\mu(k)$ is calculated through the binary search of the computed sum of probabilities (Appendix B).

Iterative refinement. To make the graph robust to test time deletion of edges from the input queries, our subgraph sampling technique also involves an iterative refinement process, where it generates a sequence of randomized subgraphs: $G'(1) \rightarrow G'(2) \rightarrow \cdots \rightarrow G'(K)$, which correspond to K optimization steps. At each step k, the edge probability is computed from the weight of the previous step $w(k-1)$. The new weight $w(k)$ is then obtained through learning to minimize the search efficiency aware loss measured on the newly sampled subgraph $G'(k)$.

The sampling ratio at iteration k is governed by a sampling policy $0 < \lambda(k) \leq 1$, which decides how many edges a sampled subgraph $G'(k)$ has. We define $\lambda(k)$ as:

$$\lambda(k) = \sigma + (\lambda(0) - \sigma)\left(1 - \frac{k}{K}\right)^c \tag{6}$$

where σ is the fraction of selected edges, $c \in \{1, 3\}$ and $\lambda(0) >= \sigma$ is the initial sampling rate (e.g., 1). There is an intuitive and straightforward motivation regarding this choice: It selects edges more randomly in the beginning and gradually becomes more selective as the optimization is close to the end. Furthermore, we update T by $T(k) = T_0 \cdot \beta^k$ along the iterations, which starts with a relatively high temperature T_0 and decays fast with a decay factor β. By generating a sequence of subgraphs varying the sampling rate, we create an ensemble of sets of subgraphs, which collectively optimize the joint probability of individual edges.

Putting it together. Algorithm 1 shows the GraSP algorithm. The optimization process ends when it meets a stopping criterion. In our current implementation, we use a simple criterion that stops after a small number of iterations (e.g., ≤ 20). There are other ways to stop without using a constant number K (e.g., by detecting how close it is to convergence) that are worth further exploration. Once the optimization finishes, we re-rank edges based on their final edge probabilities and select edges with high probabilities according to the desired keep ratio σ. In case it detects that the input distribution has significantly shifted from the training set distributions [38], it *falls back* to the original heuristic graph by unmasking the pruned edges. The computation complexity for our approach is $O(K \times (|E| \cdot log(max(\mathcal{W}) - min(\mathcal{W})) + |E| + |Q| \cdot log(|V|) + |E| \cdot log(|E|))$, and we provide a detailed analysis in Appendix C.

Algorithm 1 **GraSP**

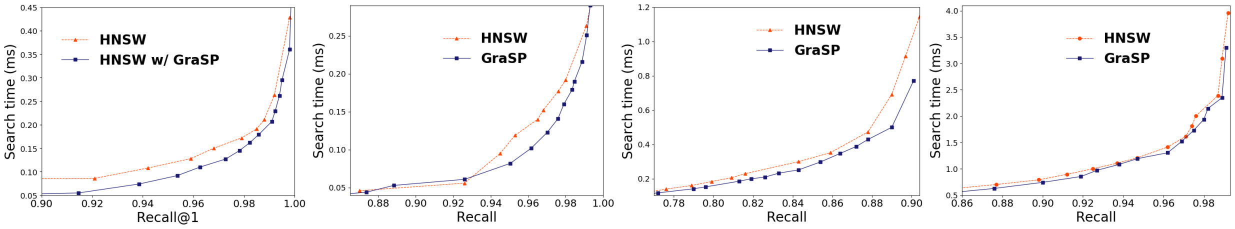
```
Input: Base set $X$, learning set $Q$.
Output: Pruned graph $G(V, E)$.
Parameters: Learning rate $\eta$, starting temperature $T_0$, decay factor $\beta$, keep ratio $\sigma$, max iteration $K$, candidate queue size $L$
Init: $T \leftarrow T_0, k \leftarrow 0$
Stage 1: Construct Annealable Similarity Graph G(V, E, $\mathcal{W}$)
Stage 2: Learning edge importance
while $k \leq K$ do
    $\lambda \leftarrow \sigma + (\lambda(0) - \sigma)\left(1 - \frac{k}{K}\right)^c$
    Normalize $w$ s.t. $|E'(k)| = \lceil \lambda(k) \cdot |E| \rceil \equiv \sum_{e \in E} \frac{1}{1+\exp\left(-\frac{w_e+\mu(k)}{T}\right)}$
    Randomly sample a subgraph $G'(k)$ based on edge probabilities
    for $q$ in $Q$ do
        $p', \_ \leftarrow search(G'(k), q, L)$
        $p, H^q \leftarrow search(G, q, L)$
        if $p \neq p'$ then
            for $h$ in $H^q$ do
                $w_h \leftarrow w_h + (\frac{\delta\langle p', q\rangle}{\delta\langle p, q\rangle} - 1) \cdot \eta$
    $T \leftarrow T_0 \cdot \beta^k$
    Shuffle $Q$
Stage 3: Pruning edges
Select $\sigma \cdot |E|$ highest ranked edges and prune the rest edges
```

5 EVALUATION

In this section, we conduct empirical evaluations by optimizing the state-of-the-arts similarity graphs using GraSP.

5.1 Methodology

Implementation details. Our implementation of GraSP is based on HNSW [1]. We choose HNSW to construct the initial graph due to its excellent performance in practice [9]. For a given dataset, we first construct an HNSW graph using the base set. Since HNSW uses 1-greedy search on upper layers (i.e., to find a good starting point at the bottom layer) and performs N-greedy search at the bottom layer, which is where the majority of search happens, we apply GraSP to optimize the bottom layer of the constructed HNSW graph. We then optimize the HNSW bottom layer using the training set by fixing the hyperparameters as $T_0 = 1$, $K = 20$, $\beta = 0.8$, $\eta = 0.1$ and tune $\sigma = [0.5, 0.9]$. We use the hold-out testing vectors provided by the benchmarks for final evaluation, and we make sure there are no overlapping between the training set and the testing set to avoid data leakage. We tune hyperparameters M and $efConstruction$ for the baseline HNSW graphs to get the best performing results, with the corresponding type of diversification heuristic turned on. We



(a) SIFT1M

(b) Deep1M

(c) GloVe

(d) GIST

Figure 4: Comparison of search time and accuracy on SIFT1M, Deep1M, GloVe, and GIST. For recall (x-axis), larger is better. For search time (y-axis), lower is better.

empirically find that performing GraSP with $\sigma = 0.7$ on an graph constructed with the average out-degree M equal or slightly larger (e.g., M=14–20) than the best performing baseline graph (e.g., M=14) could lead to better results, via grid search over M and σ. We also note that if the baseline uses a larger M (i.e., which appears to have more redundant edges) or we tune more hyperparameters of GraSP, GraSP could achieve even better results than the baseline. Additional implementation details and all parameters are reported in Appendix D.

Datasets. We evaluate our approach on four widely used benchmarks. **SIFT1M** contains 128-dimensional SIFT descriptors [27]. It consists 1,000,000 base vectors, 100,000 learning vectors, and 10,000 testing vectors. To prevent the learning from overfitting, we remove overlapped testing vectors from the original learning set, which results in 89,983 learning vectors for SIFT1M. **Deep1M** is a random 1,000,000 subset of one billion 96-dimensional machine-learned vectors [10]. We sample 100,000 vectors from the provided learn set for training. For testing, we take the original 10,000 queries. **GloVe** is a collection of 200-dimensional word embedding vectors from Twitter data [44]. We divide the original 1,193,514 vectors to get base, training, and testing sets, each containing 1,083,514, 100,000, and 10,000 vectors, respectively. **GIST** consists 960-dimensional vectors [19]. It consists 1,000,000 base vectors, 500,000 learning vectors, and 1,000 testing vectors.

Evaluation metrics. Both latency and accuracy are important metrics as they measure search efficiency. We measure query latency as the average time of per-query execution (one query at a time) in millisecond. For accuracy, we measure Recall@1, which is a challenging metric, as it also measures Precision@1 and any order inversion causes the loss of accuracy.

Platform. All the experiments were done on a 64-bit Linux Ubuntu 16.04 server with Intel Xeon CPU E5-2650 v4 @ 2.20GHz processor.

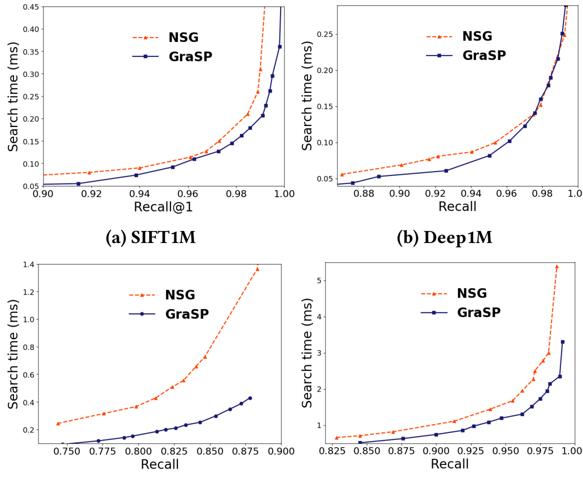
5.2 Comparison with HNSW

We first compare GraSP with the baseline HNSW method in latency-vs-recall. We use the default best-first search and vary the search queue length L to obtain search time at different recall regions. We highlight a few key observations below: For all datasets, the learning-augmented provides gains at different Recall@1 regions. For SIFT1M, there are more significant gains at the lower recall regions, where we observe 1.49X latency reduction at 90% recall. As the recall increases, the gain becomes smaller. For Deep1M, there are more significant gains in the range of 93%–99% recall, where we observe 1.34X latency reduction at 96% recall. For GloVe, the gain starts to show up at 80% recall and becomes bigger as the recall increases. For GIST, GraSP provides relatively small yet consistent gains across all recall ranges.

GraSP is able to provide better performance because it learns to identify the importance of individual edges through exploiting query distributions, which allows to better capture the specific roles of graph vertices. By pruning redundant edges, it only keeps a fraction of edges crucial for search efficiency, allowing a query to check less number of edges to find the closest vector. The improved accuracy under a given time budget is therefore an effect of the shift of the Pareto curve [2].

5.3 Comparison with NSG

NSG [25] is another state-of-the-art graph-based approach, which uses heuristics to construct graphs that approximate MRNG. This is a strong baseline, as it already performs edge control by limiting the out-degrees of all the nodes to a small value by abandoning the longer edges. We use the parameters listed on their repository for corresponding datasets due to their excellent performance.



(a) SIFT1M

(b) Deep1M

(c) GloVe

(d) GIST

Figure 5: Comparison of GraSP with NSG. The results show that GraSP outperforms NSG by achieving comparable accuracy while providing up to 2.24x faster speedups.

Fig. 5 compares NSG with GraSP. The results show that GraSP consistently achieves better accuracy-vs-latency than NSG. For example, GraSP outperforms NSG by 2.24X at 99% recall@1 point, and it outperforms NSG by 1.45X at 92% recall@1 point. GraSP outperforms NSG, because it exploits query distribution to optimize the graph indices by explicitly maximizing search efficiency. We remark that despite showing better performance than NSG, GraSP is compatible and complementary to NSG, and the GraSP should be applicable to NSG to improve its performance as well.

5.4 Comparison with Heuristic Pruning

We also compare GraSP with two pruning methods that are based on heuristics. *1. Random pruning (RP)*: we uniformly sample half of the edges from the bottom layer of HNSW to prune. *2. Reducing degree (RD)*: we directly reduce the maximum out-going degree to half of M during the graph construction phase. For a given accuracy target (e.g., 0.98 for SIFT1M, 0.94 for Deep1M, and 0.83 for GloVE), we vary L to find the minimum latency to reach the desired accuracy and report the results in Table 1.

Dataset	Configuration	Recall	Latency(ms)
SIFT1M	RP	0.966(-1.4%)	0.23
	RD	0.982(+0.2%)	0.16
	GraSP	0.982(+0.2%)	**0.13**
Deep1M	RP	0.945(+0.5%)	0.16
	RD	0.943(+0.3%)	0.14
	GraSP	0.943(+0.3%)	**0.1**
GloVe	RP	0.814(-1.6%)	1.45
	RD	0.83(0%)	0.73
	GraSP	0.833(+0.3%)	**0.55**

Table 1: Comparison between GraSP and heuristic-based pruning over SIFT1M, Deep1M, and GloVe.

We make two main observations. First, random pruning often cannot reach the target recall (e.g., on SIFT1M and GloVe), despite spending more time than the RD and GraSP. This is because random pruning may accidentally destroy graph properties that are crucial for search efficiency. Second, GraSP achieves faster speed than directly building an HNSW graph with a smaller out-degree upperbound M (e.g., M=16). This is interesting but it seems to indicate first building a large graph with redundant edges and then pruning can produce a graph with better search quality. From a graph construction perspective, setting a small upperbound M reduces the average out-degree but increases the diameter of the graph, which may accidentally increase the search path length and lead to worse performance. In contrast, our approach explicitly maximizes search efficiency by lowering the expected out-degree.

5.5 Experimental Analysis

In this section, we analyze the convergence of the optimization process and the effect of pruning ratio (i.e., 1 - keep ratio). We study how GraSP affects the graph properties and its impact to optimization time. We also study how the proposed method responds to out-of-distribution queries.

Convergence analysis. We analyze the edge probability distribution at different optimization iterations. Fig. 6 shows that the edge probability distribution starts with a uniform distribution, which corresponds to more exploration in the beginning. As the iterative refinement moves forward, T decreases by following the annealing schedule $T(k) = T_0 \cdot \beta^k$. As a result, the distribution shifts to be more biased: most of the edge probability are distributed around the two peaks, which indicates that GraSP has entered into an exploitation phase, where it tends to select edges that are important for preserving search efficiency. The probability distribution has converged around the 16–17 iterations. This is consistent with Theorem A.1, which shows that GraSP eventually converges with the joint distribution of $R_e(T)$ for all edges e being equal for $T \to \infty$ and sparsely supported for $T \to 0$.

Effect of σ and L. Fig.7a reveals that the learned graph has an interesting *phase transition* phenomenon, under different search budget ($L = 20, 50, 100$): by setting the sampling rate $\lambda(K)$ to 0.5 during the optimization, we observe that the search accuracy quickly drops after a slight change of σ to be less than 0.5. This is because GraSP learns to identify half of the edges that maximize the search efficiency. This result indicates that there is an edge-minimal graph beyond which further deleting edges would lead to significant accuracy loss. Fig.7b and Fig.7c show the distance computations (machine independent) and search time (machine dependent) under different edge selection ratio σ. The results show that (1) the pruning ratio has a large influence on search speed: smaller σ often leads to much fewer distance computations and faster search time varying L, and (2) the search time and the distance computation are linearly correlated. Finally, Fig.7d shows that, as σ decreases, the number of edges in the final graph linearly decreases, which is expected because larger sampling rates indicate that the optimization is more aggressive in terms of reducing the number of edges.

Graph properties. We are interested in how with or without GraSP affects graph properties such as the distributions of in-degree and out-degree. Fig. 8 shows that both approaches create graphs that have a binomial distribution of in-degree. However, for HNSW only, the number of neighbors (i.e., out-degree) for all vertices is distributed around a fixed value (e.g., 32). This is because HNSW forces all vertices to have a similar number of edges, where some of the intrinsic distribution of the data, such as variation in density, is inevitably lost. In contrast, the out-degree of GraSP optimized graph is smoothed out to have a truncated power-law distribution: It keeps the connections of "hubs" while limiting the neighbors for nodes in sparse areas. Interestingly, prior work investigates graph navigability with truncated power-law degree distribution for similarity search and indicates that such distribution is likely to provide efficient search [35]. In our approach, such properties naturally emerge after the learning (more results in Appendix F).

Optimization time. The learning time of our approach is 77, 40, and 118 minutes for SIFT, Deep1M, and GloVe, respectively with single thread, in addition to the index construction time of the heuristic graphs. The index construction time is 29 minutes for HNSW [1], and 3 minutes for NSG (including k-NN construction) on SIFT1M. Despite taking additional time to learn, we do want to make it clear that the primarily goal of this work is to improve online search efficiency in order to reduce the total cost of ownership: a single index that gets queried hundreds of millions of times over its life cycle, where index construction is one-time cost and is not a major bottleneck. Additionally, the optimization time can

[1] We use a large *efConstruction* (e.g. 400) than the default 200, which leads to increased construction time but provides better online search-vs-accuracy performance.

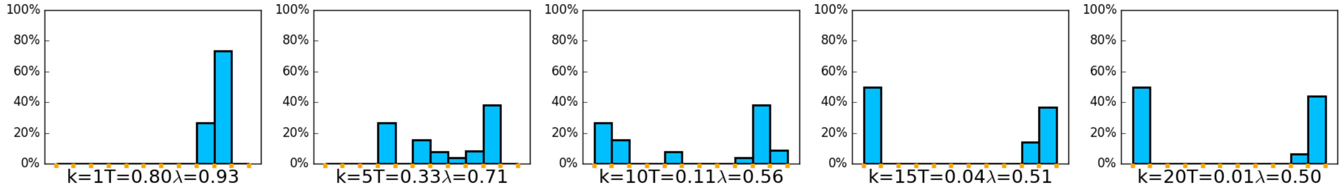
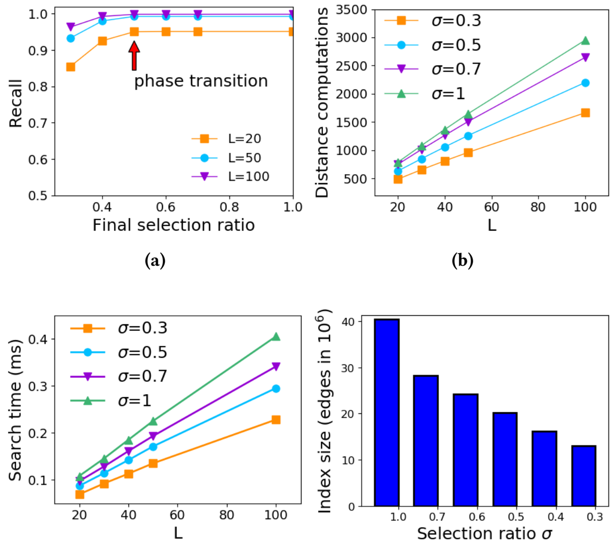


Figure 6: Distribution of edge probability at different learning steps of GraSP. The x-axis shows the probability bins, from 0 to 1 with intervals of 0.1. The y-axis represents the percentage of edges that fall into a bin.



(a)

(b)

(c)

(d)

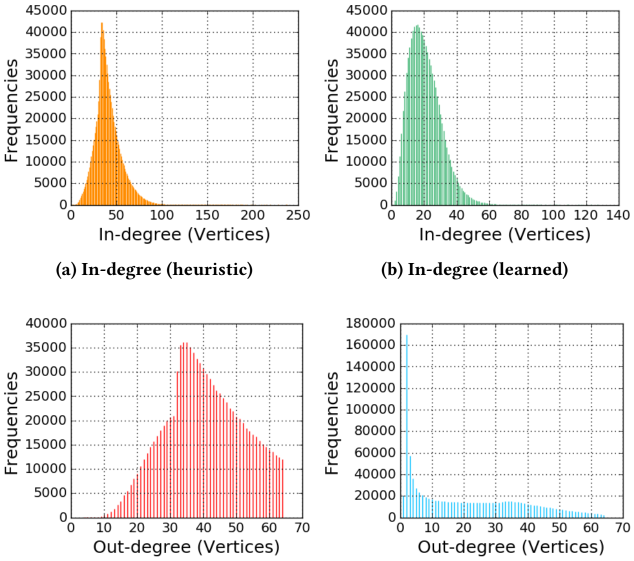
Figure 7: Impact of (a) index size, (b) accuracy, (c) distance computations, and (d) search time, under different selection ratio σ and different L.

be accelerated through parallelism, which we leave to explore in future work.

Out-of-distribution queries. To test out how the proposed method responds to out-of-distribution queries, we generated queries with normal distribution $\mathcal{N}(0, 1)$. Fig. 9 shows the comparison with HNSW. The results indicate that out-of-distribution queries affect GraSP's performance more than HNSW. This is somewhat expected, because our approach optimizes graphs based on query distribution. An out-of-distribution query serves as an adversarial example to the learning-based approach. However, in practice, given that both database vectors and query vectors are generated by the same DNN model, it is unlikely the case that these out-of-distribution queries would appear.

6 CONCLUSION AND FUTURE WORK

The similarity graph is an important data structure for building deep learning applications. It is crucial to make it answer queries with low latency and high accuracy in production. In this work, we show that it is possible to further improve the search efficiency of graph-based similarity search by exploring query distribution, opening new possibilities for ANN search optimizations. One limitation of our work is that we have not investigated larger datasets because learning can be computationally intensive as the dataset size grows. In future work, we intend to explore more efficient learning methods that will make the learning-based optimization easier to scale to large-scale datasets. Furthermore, it would be interesting to find an easier or automatic way to tune the hyperparameters in GraSP, such that GraSP will be able to automatically prune excess edges from the graph as much as it will not hurt the recall. This would offer a great advantage against existing methods, for which we need to determine a good M through tuning.



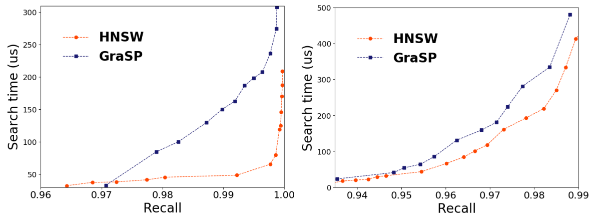
(a) In-degree (heuristic)

(b) In-degree (learned)

(c) Out-degree (heuristic)

(d) Out-degree (learned)

Figure 8: Distributions of in-/out-degree without and with GraSP optimization on SIFT1M (M=32).



(a) SIFT1M

(b) Deep1M

Figure 9: Comparison of out-of-distribution queries.

REFERENCES

[1] [n.d.]. Header-only C++/python library for fast approximate nearest neighbors. https://github.com/nmslib/hnswlib.

[2] [n.d.]. Pareto Efficiency . https://en.wikipedia.org/wiki/Pareto_efficiency. Accessed: 17-March-2021.

[3] 2019. Faiss: A library for efficient similarity search. https://engineering.fb.com/data-infrastructure/faiss-a-library-for-efficient-similarity-search/.

[4] Accessed: 05-20-2019. Faiss: A library for efficient similarity search and clustering of dense vectors. https://github.com/facebookresearch/faiss.

[5] Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM* (2008).

[6] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. 2015. Practical and Optimal LSH for Angular Distance. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015*. 1225–1233.

[7] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. 2015. Practical and Optimal LSH for Angular Distance. In *NeurIPS*. 1225–1233.

[8] Alexandr Andoni, Piotr Indyk, and Ilya P. Razenshteyn. 2018. Approximate Nearest Neighbor Search in High Dimensions. *CoRR* abs/1806.09823 (2018).

[9] Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Inf. Syst.* 87 (2020). https://doi.org/10.1016/j.is.2019.02.006

[10] Artem Babenko and Victor S. Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *CVPR 2016*. 2055–2063.

[11] Dmitry Baranchuk and Artem Babenko. 2019. Towards Similarity Graphs Constructed by Deep Reinforcement Learning. arXiv:1911.12122 [cs.LG]

[12] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. 2018. Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In *ECCV 2018*. 209–224.

[13] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD 1990*. 322–331.

[14] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (Sept. 1975), 509–517.

[15] Duncan S Callaway, Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. 2000. Network robustness and fragility: Percolation on random graphs. *Physical review letters* 85, 25 (2000), 5468.

[16] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. *SPTAG: A library for fast approximate nearest neighbor search*.

[17] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, Shilad Sen, Werner Geyer, Jill Freyne, and Pablo Castells (Eds.). ACM.

[18] Paul Adrien Maurice Dirac. 1926. On the theory of quantum mechanics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 112, 762 (1926), 661–677.

[19] Matthijs Douze, Hervé Jégou, Harsimrat Sandhawalia, Laurent Amsaleg, and Cordelia Schmid. 2009. Evaluation of GIST descriptors for web-scale image search. In *Proceedings of the 8th ACM International Conference on Image and Video Retrieval, CIVR 2009, Santorini Island, Greece, July 8-10, 2009*, Stéphane Marchand-Maillet and Yiannis Kompatsiaris (Eds.). ACM.

[20] Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. 2018. Link and Code: Fast Indexing With Graphs and Compact Regression Codes. In *CVPR 2018*.

[21] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2019. Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search. *Proc. VLDB Endow.* 13, 3 (2019), 403–420.

[22] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. 2013. Reducing Web Latency: The Virtue of Gentle Aggression. In *SIGCOMM '13*. 159–170.

[23] Cong Fu and Deng Cai. 2016. EFANNA : An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph. *CoRR* abs/1609.07228 (2016).

[24] Cong Fu, Changxu Wang, and Deng Cai. 2019. Satellite System Graph: Towards the Efficiency Up-Boundary of Graph-Based Approximate Nearest Neighbor Search. *CoRR* abs/1907.06146 (2019).

[25] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph. In *VLDB'19*.

[26] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *CoRR* abs/1503.02531 (2015).

[27] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. In *TPAMI 2011*.

[28] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* (2019).

[29] Yannis Kalantidis and Yannis S. Avrithis. 2014. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search. In *CVPR 2014*.

[30] Quoc V. Le and Tomás Mikolov. 2014. Distributed Representations of Sentences and Documents. In *ICML 2014*, Vol. 32. JMLR.org, 1188–1196.

[31] D. T. Lee and C. K. Wong. 1977. Worst-case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees. *Acta Informatica* 9, 1 (March 1977), 23–29.

[32] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Wenjie Zhang, and Xuemin Lin. 2019. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[33] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision* 60, 2 (Nov. 2004).

[34] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Inf. Syst.* 45 (2014), 61–68.

[35] Yury A. Malkov. 2015. Growing homophilic networks are natural optimal navigable small worlds. *CoRR* abs/1507.06529 (2015).

[36] Yury A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.

[37] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR 2013*, Yoshua Bengio and Yann LeCun (Eds.).

[38] Jose G. Moreno-Torres, Troy Raeder, Rocío Alaíz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern Recognit.* (2012), 521–530.

[39] Marius Muja and David G. Lowe. 2014. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *TPAMI 2014* 36, 11 (2014), 2227–2240.

[40] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian Allen Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. 2019. Semantic Product Search. In *KDD 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 2876–2885.

[41] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian Allen Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. 2019. Semantic Product Search. In *KDD 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 2876–2885.

[42] Mohammad Norouzi and David J. Fleet. 2013. Cartesian K-Means. In *CVPR 2013*.

[43] Mohammad Norouzi, David J. Fleet, and Ruslan Salakhutdinov. 2012. Hamming Distance Metric Learning. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. 1070–1078.

[44] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *EMNLP*. 1532–1543.

[45] Xiang Ren, Yujing Wang, Xiao Yu, Jun Yan, Zheng Chen, and Jiawei Han. 2014. Heterogeneous Graph-based Intent Learning with Queries, Web Pages and Wikipedia Concepts. In *WSDM '14* (New York, New York, USA). 23–32.

[46] Christian M. Schneider, André A. Moreira, José S. Andrade Jr., Shlomo Havlin, and Hans J. Herrmann. 2011. Mitigation of Malicious Attacks on Networks. *CoRR* abs/1103.1741 (2011). arXiv:1103.1741

[47] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[48] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Rand-NSG: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In *NeurIPS*. 13748–13758.

[49] Danny Sullivan. 2018. FAQ: All about the Google RankBrain algorithm. https://searchengineland.com/faq-all-about-the-new-google-rankbrain-algorithm-234440.

[50] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba. In *KDD 2018*. 839–848.

[51] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *SIGIR 2017*. 55–64.

[52] Peter N. Yianilos. 1993. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *SODA '93*. 311–321.

[53] Hamed Zamani, Mostafa Dehghani, W. Bruce Croft, Erik G. Learned-Miller, and Jaap Kamps. 2018. From Neural Re-Ranking to Neural Ranking: Learning a Sparse Representation for Inverted Indexing. In *CIKM 2018*. 497–506.

[54] Jialiang Zhang, Soroosh Khoram, and Jing Li. 2018. Efficient Large-Scale Approximate Nearest Neighbor Search on OpenCL FPGA. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4924–4932.

[55] Minjia Zhang and Yuxiong He. 2019. GRIP: Multi-Store Capacity-Optimized High-Performance Nearest Neighbor Search for Vector Search Engine. In *CIKM 2019*. 1673–1682.

[56] Yanhao Zhang, Pan Pan, Yun Zheng, Kang Zhao, Yingya Zhang, Xiaofeng Ren, and Rong Jin. 2021. Visual Search at Alibaba. *CoRR* abs/2102.04674 (2021). arXiv:2102.04674

A CONVERGENCE AND CORRECTNESS

In this part, we provide a theoretically derived proof of the convergence and correctness of our algorithm in Theorem A.1 below.

THEOREM A.1. *Let $G_0(V, E_0)$ be the original graph to prune, and let Q be the query set. Suppose there exists a subset of edges $E^* \subseteq E_0$ such that the average recall of retrieving the nearest neighbor in V for all queries $q \in Q$ using edges in E^* is 1. Let $G(k)$ be the random graph at iteration k running Algorithm 1, and let $\{R_e(k)\}$ be the family of Bernoulli random variables defined on the edges of $G(k)$, with $\mathbb{P}(R_e(k) = 1) = p_e(k)$. Assume that $p_e(k_e) < \frac{1}{2}$ for $e \in E_0 \backslash E^*$ at some step k_e. Also assume that $T(k) \geq T(k+1)$ for all k. If $\sigma = 1 - |E^*|/|E_0|$, then:*

(1) $\sum_{e \in E^*} p_e(k+1) \geq \sum_{e \in E^*} p_e(k)$ for all sufficiently large k. In particular, $\lim_{k\to\infty} \sum_{e \in E^*} p_e(k)$ exists.

(2) If $\lim_{k\to\infty} T(k) = 0$, then $\lim_{k\to\infty} p_e(k) = 1$ for all $e \in E^*$, and $\lim_{k\to\infty} p_e(k) = 0$ for all $e \in E_0 \backslash E^*$.

(3) With the condition in (2), suppose $r(k)$ denotes the average expected recall of retrieving the ground truth nearest neighbor of a query $q \in Q$ in a subgraph of $G(k)$ randomly sampled from the joint distribution $\{R_e(k)\}$. Then $\lim_{k\to\infty} r(k) = 1$.

Theorem **??**. We start with showing (1). Note that the probability sum constraints

$$\sum_{e \in E^*} p_e(k) + \sum_{e \in E_0 \backslash E^*} p_e(k) = (1-\sigma)|E_0| = |E^*|$$

is satisfied for all k. It then suffices to show that

$$\sum_{e \in E \backslash E^*} p_e(k+1) \leq \sum_{e \in E \backslash E^*} p_e(k)$$

for sufficiently large k. Observe that $w_e(k)$ remains a constant ($= w_e(0)$) for all k and all $e \in E_0 \backslash E^*$, and that $\mu(k)$ monotonically decreases as k increases. $w_e(k)$ remains a constant for all k and all edges in E^* because the recall through all edges in E^* is 1, so the weights associated to these edges are not penalized by the algorithm. Therefore they remain constant. $\mu(k)$ decreases because the constraint in Equation 5 monotonically increases as either w_e or $\mu(k)$ increases. During the annealing process, $\mu(k)$ does not increase since each w_e does not. We claim that $p_e(k+1) \leq p_e(k)$ for all $k \geq k_e$. Together with the probability sum constraints this shows that $\sum_{e \in E^*} p_e(k)$ is monotonically increasing for $k \geq \max_e k_e$. Since $\sum_{e \in E^*} p_e(k)$ is bounded from above, $\lim_{k\to\infty} \sum_{e \in E^*} p_e(k) = \sup_k \sum_{e \in E^*} p_e(k)$ exists.

For (2), since $\lim_k T_k = 0$, observe that $\lim_{k\to\infty} p_e(k) \to 0$ for all $e \in E_0 \backslash E^*$. Thus the probability sum constraint enforces that $\lim_k \sum_{e \in E^*} p_e(k) = |E^*|$. Recall that $\liminf_k x_k + \limsup_k y_k \geq \liminf_k (x_k + y_k)$ for any real sequences x_k and y_k. Now consider any $e \in E^*$, we have

$$\begin{aligned}
&\liminf_{k\to\infty} p_e(k) \\
\geq\ & \lim_{k\to\infty} \sum_{e \in E^*} p_e(k) - \limsup_{k\to\infty} \sum_{e' \in E^* \backslash \{e'\}} p_{e'}(k) \\
\geq\ & |E^*| - \sum_{e' \in E^* \backslash \{e'\}} \limsup_{k\to\infty} p_{e'}(k) \\
\geq\ & |E^*| - \sum_{e' \in E^* \backslash \{e'\}} 1 \\
=\ & |E^*| - \big(|E^*| - 1\big) = 1.
\end{aligned}$$

which implies that $\lim_k p_e(k) = 1$.

To prove (3), we first define the notation $\chi(q, E)$ to be the indicator function of whether the ground truth nearest neighbor of a query q can be retrieved in G using a subset of edges $E \subseteq E_0$. More precisely, $\chi(q, E) = 1$ if the nearest neighbor of q can be retrieved using edges in E; and $\chi(q, E) = 0$ otherwise. Note that the average recall at step k can be written as

$$r_k = \frac{1}{|Q|} \sum_{q \in Q} \sum_{E \subseteq E_0} \chi(q, E) \mathbb{P}(E \text{ is chosen from } G(k)),$$

where the second summation is amongst all subsets of E_0. Since each edge being chosen is independent of other edges, we can expand the probability and it follows that

$$r_k = \frac{1}{|Q|} \sum_{q \in Q} \sum_{E \subseteq E_0} \left(\chi(q, E) \prod_{e \in E} p_e(k) \prod_{e \notin E} (1 - p_e(k)) \right).$$

Since we assumed that $\chi(q, E^*) = 1$ for all $q \in Q$, it follows that $\chi(q, E) = 1$ for all $E \supseteq E^*$. Thus

$$r_k \geq \frac{1}{|Q|} \sum_{q \in Q} \sum_{E^* \subseteq E \subseteq E_0} \left(\prod_{e \in E} p_e(k) \prod_{e \notin E} (1 - p_e(k)) \right).$$

Note that

$$\begin{aligned}
& \sum_{E^* \subseteq E \subseteq E_0} \left(\prod_{e \in E} p_e(k) \prod_{e \notin E} (1 - p_e(k)) \right) \\
=\ & \sum_{E^* \subseteq E \subseteq E_0} \left(\prod_{e \in E^*} p_e(k) \prod_{e \in E \backslash E^*} p_e(k) \prod_{e \notin E} (1 - p_e(k)) \right) \\
=\ & \prod_{e \in E^*} p_e(k) \sum_{E^* \subseteq E \subseteq E_0} \left(\prod_{e \in E \backslash E^*} p_e(k) \prod_{e \notin E} (1 - p_e(k)) \right) \\
=\ & \prod_{e \in E^*} p_e(k) \sum_{E^* \subseteq E \subseteq E_0} \mathbb{P}(E \text{ is chosen from } E_0 \backslash E^*) \\
=\ & \prod_{e \in E^*} p_e(k).
\end{aligned}$$

Therefore

$$\begin{aligned}
\liminf_{k\to\infty} r_k \quad & \geq \quad \liminf_k \frac{1}{|Q|} \sum_{q \in Q} \prod_{e \in E^*} p_e(k) \\
& \geq \quad \frac{1}{|Q|} \sum_{q \in Q} \liminf_{k\to\infty} \prod_{e \in E^*} p_e(k) \\
& \geq \quad \frac{1}{|Q|} \sum_{q \in Q} \prod_{e \in E^*} \liminf_{k\to\infty} p_e(k) \\
& = \quad \frac{1}{|Q|} \sum_{q \in Q} 1 = 1.
\end{aligned}$$

This shows that $\lim_{k\to\infty} r_k = 1$ and completes the proof. □

B BINOMIAL WEIGHT NORMALIZATION

Algorithm 2 describes the details of the binomial normalization and graph sampling based on normalized weights. The binary search is equivalent to find $\mu(k)$ to have the sum of join distribution equal to the target number of edges in the sampled subgraph, which has a time complexity of $O(|E| \cdot \log(\max(w(k)) - \min(w(k)))$.

	EFANNA					HNSW		NSG				GraSP	
	K	L	iter	S	R	M	efC	nn	R	L	C	M	efS (learning)
SIFT1M	200	200	10	10	100	14	500	200	50	40	500	14	400
Deep1M	200	200	10	10	100	14	500	200	50	40	500	20	400
GloVe	-	-	-	-	-	16	500	400	70	60	500	20	400
GIST	-	-	-	-	-	24	500	400	70	60	500	24	400

Table 2: Hyperparameters for running SIFT1M, Deep1M, GloVe, and GIST.

Algorithm 2 **Binomial_Weight_Normalization()**

```
Input: $G^*(V, E)$, sampling ratio $\lambda$, $\mathcal{W} = \{w_e | e \in E\}$, temperature T
Output: $G^{\succ}$(V, E')
E' ⟵ {}
target ⟵ $\lambda \cdot |E|$
max_w ⟵ $max(\mathcal{W})$
min_w ⟵ $min(\mathcal{W})$
avg_w ⟵ $T \cdot log(\frac{\lambda}{1-\lambda})$
search_range_min ⟵ avg_w - max_w
search_range_max ⟵ avg_w - min_w
$\mu$ ⟵ binary_search(arr=$\mathcal{W}$, left=search_range_min, right=search_range_max, num=target)
for e in $E$ do
    $w_e \longleftarrow w_e + \mu$                        ▷ Normalize weights with offset $\mu$
    $p_e = \frac{1}{1+exp^{(-\frac{w_e}{T})}}$                ▷ Recompute keep probability
```

C COMPUTATION COMPLEXITY

GraSP (Algorithm 1) goes through K optimization steps. For each step, the cost consists of three parts:

- The cost of binomial normalization, which has a complexity of $O(|E| \cdot log(max(\mathcal{W}) - min(\mathcal{W})))$;
- The cost of sampling to get a subgraph, which takes $O(|E|)$ complexity;
- The cost for updating the edge weights, which is $O(|Q| \cdot log(|V|))$, because queries take logarithmic time complexity on proximity graphs.
- The cost for the final ranking of the edge weights, which is $O(|E| \cdot log(|E|))$.

Consequently, GraSP has the computation complexity of $O(K \times (|E| \cdot log(max(\mathcal{W}) - min(\mathcal{W})) + |E| + |Q| \cdot log(|V|) + |E| \cdot log(|E|))$. The value K is often small (e.g., < 20) in practice.

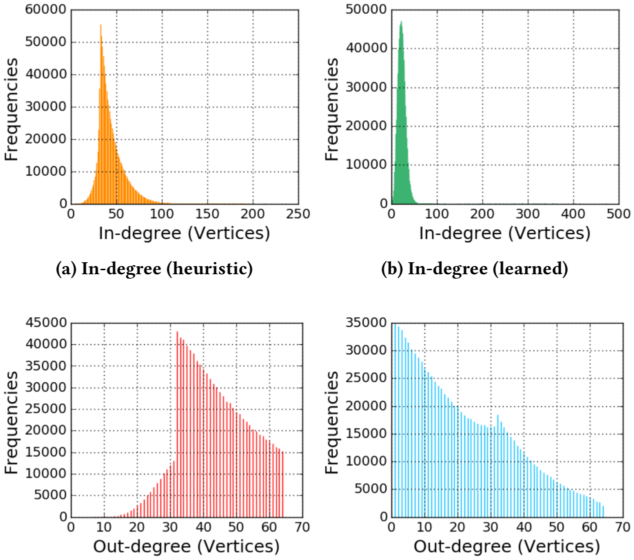
D IMPLEMENTATION

We implement GraSP in C++. Subgraph sampling is implemented by using a binary mask map, which is of the same size as the number of edges, to determines edges that are kept in the sampled subgraph. The edges that are masked are not visited when searching the sampled subgraphs. However, their weight can still be updated (e.g., during the binomial normalization). For our baseline, we also try to test a concurrent method that uses reinforcement learning to construct graphs [11]. However, the open-sourced code reports a proxy metric NDC (num. of distance calculations) instead of actual latency. When measuring the actual execution time, it appears to be orders of magnitude slower than our approach. By looking into their implementation, the code is implemented in Python and is not optimized for online searching. In contrast, we implement GraSP using C++ and take high performance into account. For this reason, we focus on evaluation of high performance graph-based ANN methods.

E HYPERPARAMETERS

Table 2 reports the hyperparameters for each dataset/configuration.

F MORE RESULTS ON GRAPH PROPERTY ANALYSIS



(a) In-degree (heuristic)

(b) In-degree (learned)

(c) Out-degree (heuristic)

(d) Out-degree (learned)

Figure 10: Distributions of in-/out-degree from heuristic-based and learning-augmented graphs over Deep1M.

This part includes more results on graph property analysis. In particular, Fig. 10 show the frequency distributions of in-degree and out-degree from heuristic-based (HNSW) and learned (GraSP) proximity graphs. Overall, we observe similar results as SIFT: the in-degree remains to have a binomial distribution, and the out-degree has a truncated power-law degree distribution for the GraSP graph. The heuristic-based graph has an uneven distribution with a discontinuity from out-degree 32 and 33. This is because the HNSW graph has a nested hierarchy, where upper layers are recursively sampled from the bottom layer. While nodes at the bottom layer have maximally M = 64 outgoing edges for each node, nodes at the upper layers have only 32 ($\frac{M}{2}$) outgoing edges per node. Therefore, the discontinuity at 32 is caused by those upper layers, which have a different edge distribution.