

Constrained Approximate Similarity Search on Proximity Graph

Weijie Zhao, Shulong Tan, Ping Li

Cognitive Computing Lab
Baidu Research
10900 NE 8th St. Bellevue, WA 98004, USA
{zhaoweijie12, tanshulong2011, pingli98}@gmail.com

Abstract

Search engines and recommendation systems are built to efficiently display relevant information from those massive amounts of candidates. Typically a three-stage mechanism is employed in those systems: (i) a small collection of items are first retrieved by (e.g.,) approximate near neighbor search algorithms; (ii) then a collection of constraints are applied on the retrieved items; (iii) a fine-grained ranking neural network is employed to determine the final recommendation. We observe a major defect of the original three-stage pipeline: Although we only target to retrieve k vectors in the final recommendation, we have to preset a sufficiently large s ($s > k$) for each query, and “hope” the number of survived vectors after the filtering is not smaller than k . That is, at least k vectors in the s similar candidates satisfy the query constraints.

In this paper, we investigate this **constrained similarity search** problem and attempt to merge the similarity search stage and the filtering stage into one single search operation. We introduce **AIRSHIP**, a system that integrates a user-defined function filtering into the similarity search framework. The proposed system does not need to build extra indices nor require prior knowledge of the query constraints. We propose three optimization strategies: (1) starting point selection, (2) multi-direction search, and (3) biased priority queue selection. Our first starting point selection optimization is to locate good starting points for the graph searching algorithm. We would like to start the graph search inside a cluster of satisfied vectors. In this scenario, a small number of satisfied vectors can be retrieved along the path to the query vector. Then, we propose a multi-direction search optimization that enables multi-direction searching. Compared with the single priority queue used in the original graph search, we have two priority queues to keep search candidates: one stores the satisfied vectors and the other maintains remaining unsatisfied vectors. When both priority queues are not empty, we choose each queue with a weight ratio, *alter_ratio*. On top of that, we propose a biased priority queue selection to adaptively select the queue for searching: when the top candidate from the satisfied vector queue is better than the one from the other queue, we override the *alter_ratio* restriction and select the satisfied vector queue. All three optimizations improve the similarity search performance. The proposed algorithm has a hyper-parameter *alter_ratio* that is data/query dependent. We present an estimation method to adaptively choose the hyper-parameter. Experimental evaluations on both synthetic and real data confirm the effectiveness of the proposed **AIRSHIP** algorithm.

We focus on constrained graph-based approximate near neighbor (ANN) search in this study, in part because graph-based ANN is known to achieve excellent performance. We believe it is also possible to develop constrained hashing-based ANN or constrained quantization-based ANN.

1 Introduction

The task of searching for similar items is the standard routine in numerous industrial applications as well as research problems in computer science and other fields. Finding the exact near (or nearest) neighbors is often expensive and typically an approximate solution would be sufficient. The research on developing efficient algorithms for approximate near neighbor (ANN) search dated back at least to the 1970's (Friedman et al., 1975, 1977). In recent years, ANN algorithms have become the crucial component in modern recommender systems (Davidson et al., 2010; Dror et al., 2011; Fan et al., 2019; Kanakia et al., 2019), Among many application domains. In this paper, we study ANN algorithms for an important application scenario, that is, the retrieved items must satisfy certain pre-specified constraints. More specifically, we focus on graph-based ANN algorithms under constraints. We should emphasize that, although this research problem was motivated from the production needs at Baidu, we only report experiments using public datasets.

1.1 Algorithms for Approximate Near Neighbor (ANN) Search

Broadly speaking, most approximate similarity search methods fall in the following 5 categories:

- Hash-based methods. Traditionally, hashing-based methods (Broder et al., 1997; Indyk and Motwani, 1998; Charikar, 2002; Datar et al., 2004; Li and Church, 2005; Lv et al., 2007; Paulevé et al., 2010; Shrivastava and Li, 2012, 2014; Wang and Dasgupta, 2016; Li, 2019; Li et al., 2021) partition the base vectors into a constant number of buckets via a specific hashing function. The chosen hashing function guarantees a collision probability is positively correlated to the specific similarity of two vectors, i.e., similar vectors have a high probability to be assigned to the same hash bucket. Relatively recently, learning to hash methods (Kulis and Darrell, 2009; Grauman and Fergus, 2013; Lai et al., 2015; Wang et al., 2016a; Dong et al., 2020) become popular. Typically they learn the hashing on a sample of data so that the learned hash function can better fit the data distribution compared with the classic data-independent hashing algorithms.
- Tree-based methods (Friedman et al., 1975, 1977; Jagadish et al., 2005; Cayton, 2008; Ram and Gray, 2012; Curtin et al., 2013; Ram and Sinha, 2019) partition the high-dimensional data space into smaller regions via a decision tree/space partitioning tree. The partitioned spaces are organized within a tree: A leaf node in the tree corresponds to the finest granularity of the partitioned space, and edges between tree nodes represent the hierarchical relationship of partitioned spaces. Hamilton et al. (2020) recently propose a tree-based method that builds an inverted index for each label/attributes. It is infeasible to build an index for each label combination without prior knowledge of the query constraints. Our proposed solution performs the original index and does not need to build any additional indices. Moreover, we do not need to know the query constraints beforehand.
- Quantization-based algorithms (Jégou et al., 2011; Ge et al., 2013; Wang et al., 2016b; Wu et al., 2017; Xu et al., 2018; Echiabi et al., 2021) learn a code-book from base vectors and quantize the original high-dimensional vectors into a low-dimensional quantized values. The quantized value is a low-dimensional representation of the original vectors. The distances between quantized vectors approximate the distances between original vectors.

- Graph-based algorithms achieve a lot of attention recently due to its superior performance to other similarity search techniques on many real-world datasets and industrial applications (Hajebi et al., 2011; Wu et al., 2014; Malkov et al., 2014; Malkov and Yashunin, 2020; Fu et al., 2019; Tan et al., 2019; Zhou et al., 2019; Tan et al., 2021a; Xu et al., 2022). Graph-based algorithms build a proximity graph as an index and perform a graph searching to answer the query. We investigate the constrained similarity search on the graph-based algorithm. Recently, Zhao et al. (2020) develop the first GPU-based graph-based ANN algorithm.
- Neural ranking. To resolve issues related to proximity search in various neural models, in recent years, the subject of “neural ranking” has attracted increasingly more attentions (Zhu et al., 2018, 2019; Tan et al., 2020; Zhuo et al., 2020; Gao et al., 2020; Tan et al., 2021b; Yu et al., 2022; Zhao et al., 2022). Effective algorithms for neural ranking can be very desirable in practice. For example, while the standard “two-tower” model is popular and fairly effective, practitioners might hope to replace the simple “cosine” loss function with a most sophisticated deep neural nets for better retrieval accuracy. However, once we have replaced the cosine by a neural net, we can no longer use the standard ANN techniques to achieve fast ranking.

Among the above methods, hashing-based methods are typically the simplest but the performance highly depends on the data and the chosen hashing functions. Graph-based ANN methods typically outperforms other methods, often considerably. In this study, we focus on graph-based ANN methods.

1.2 Approximate Near Neighbor Search Under Constraints

In this paper, we consider a practical scenario commonly encountered in industrial applications, as illustrated in Figure 1. That is, the returned search results have to satisfy certain constraints. While engineers and researchers from industry would probably agree this is a ubiquitous task in general, how to effectively solve the problem depends on the pattern of the constraints.

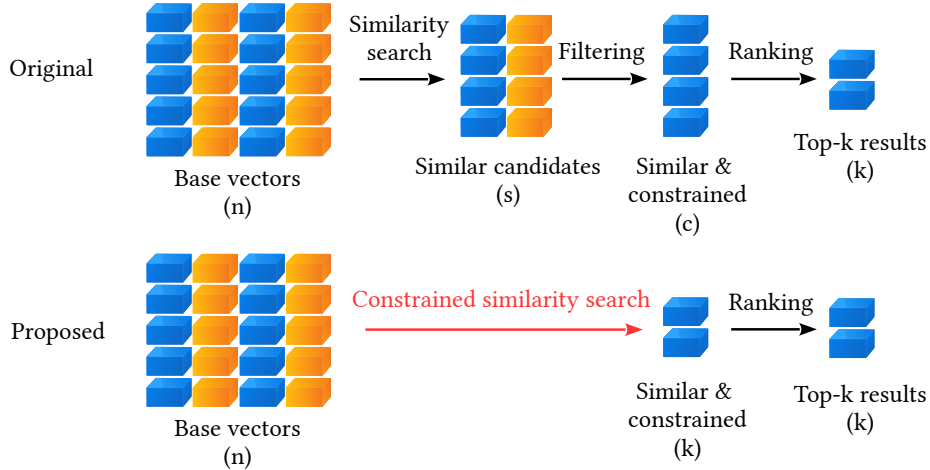


Figure 1: The three-stage pipeline v.s our proposed constrained similarity search workflow.

In Figure 1 (upper), we describe the usual strategy. That is, we consider there are initially n vectors. For a particular query, the ANN search algorithm returns s vectors (from the n vectors). After applying the constraints on the s vectors, the number is reduced to c . Finally, we apply the re-ranking algorithm (which can be either simply based on the similarity or based on the prediction results from a neural net or other learning model) to obtain the final top- k results. Obviously, with this strategy, practitioners would like to make sure that $c > k$, but this is not always easy to achieve

unless we increase s to be much larger than k . For example, when the target is to retrieve the top-1000 vectors, practitioners may have to first retrieve the top-5000 or even top-10000 vectors and then apply the constraints on the retrieved vectors. In this study, we hope to develop better schemes to improve the efficiency.

Different from the pipeline in the upper part of Figure 1, there is another obvious strategy by simply applying the constraints on the original n vectors before conducting approximate near neighbor search. If engineers know (or guess) beforehand that directly filtering on the n vectors would result in only a small number of vectors, then this would be a good strategy, provided that the cost for applying the constraints on all n vectors is not too high (for example, when indices have been built for those constraints). In this paper, we do not consider this scenario.

As illustrated in the bottom part of Figure 1, in this study, our goal is to directly retrieve the top- k vectors which satisfy the constraints. We focus on the graph-based ANN algorithm in this paper and we believe that one can also develop, with efforts, retrieval algorithms of similar nature based on other ANN methods such as hashing-based or quantization-based ANN algorithms. We leave them for future research.

1.3 Constrained Similarity Search

More formally, we merge the similarity search stage and the filtering stage into one single search operation. The merging has two principal advantages:

1. The constrained similarity search targets to return top- k similar candidates that satisfy the constraints. We do not need to estimate s to ensure the search performance.
2. The similarity search algorithm takes the constraints into the consideration—additional indices and optimizations can be potentially applied to the search.

Problem Statement. We have a collection of n d -dimensional base vectors $V = \{v_1, v_2, \dots, v_n\}$. Each vector has m attributes: the attributes of the i^{th} vector are $\{a_{i1}, a_{i2}, \dots, a_{im}\}$. In addition, we are given Q queries. The j^{th} query consists of a query vector q_j and a user-defined function f_j as the constraint. For each query vector q_j , we are required to return k similar candidates $(r_{j1}, r_{j2}, \dots, r_{jk})$ from base vectors and each candidate $r_{j.}$ satisfies the constraint, i.e., $f_j(r_{j.}) \rightarrow \mathbf{true}$.

Challenges & approaches. We identify two main challenges for the constrained similarity search. The first challenge is that we are required to efficiently identify the “interesting region” that contains vectors that meet the given constraints. There is no sub-linear algorithm to tackle this challenge without any assumptions to the query constraints and vector distributions: Before knowing queries, it is impractical to build indices for all possible constraints in pre-processing. On the other hand, after queries arrive, building indices for all different constraints is more time-consuming than a linear scan. Therefore, we have one assumption in this paper: for each constraint, there are at least $p\%$ of base vectors are satisfied vectors.

Secondly, the distributions of the vectors that satisfy the constraints are unknown. We need to retrieve similar vectors from them. However, similarity search techniques require us to construct an index beforehand. In this paper, we propose an adaptive searching algorithm on proximity graph that explores the clusterness of the vector distribution—vectors which meet the constraints are very likely to cluster in real-world datasets because the vector representations are trained based on those attributes. We leverage a heuristic algorithm to determine hyper-parameters in the proposed searching algorithm.

Why not inverted index? We have explained this earlier but we would like to re-iterate the problem since it is a common question. Basically, for applications in which it is efficient to build indices for the constraints and the fraction of data vectors after filtering is small, then we might want to go with this straightforward approach. In real applications, however, it is often inefficient to build indices for all the (potentially combinatorial number of) constraints. Creating this many indices is not only time-consuming but also space-intensive, because we would need to duplicate a vector each time it is captured by a query constraint. Note that the satisfied vectors for different query constraints may overlap. Hence, in this paper we seek more efficient solutions for scenarios in which the inverted index approach is not applicable.

Contributions. Our major contributions are summarized as follows:

- We introduce AIRSHIP (an acronym for Attribute-constrained Similarity search on proximity graph) that integrates a user-defined function filtering into the similarity search framework. AIRSHIP requires no extra indices nor prior knowledge of the query constraints.
- We propose three optimizations for the attribute-constrained searching problem: starting point selection, multi-direction search, and biased priority queue selection.
- We present an estimation method to choose the hyper-parameter of the searching algorithm.
- We experimentally evaluate the proposed algorithm on both synthetic and real datasets. The empirical results confirm the effectiveness of our optimizations.

2 AIRSHIP: Attribute-Constrained Similarity Search

In this section, we introduce our solutions to the attribute-restricted similarity search problem. We begin with a baseline adaption of graph-based similarity search methods. Then, we discuss the drawbacks of the baseline solution and propose our optimizations step by step. Each optimization is built on top of the previous one.

2.1 Vanilla Graph-Based Similarity Search

Algorithm 1 illustrates the workflow of the vanilla graph-based similarity search. From Lines 1-3, a random starting point S is selected and base data structures are initialized. In each iteration, the vector that is closest to the query point is extracted from the priority queue (Line 5). The algorithm is terminated when we have obtained K candidates and the current searched vector is worse than the worst vector in $topk$ (Lines 6-8). The only difference to the original graph-based similarity search algorithm locates in Lines 9-14: We only update the top- K candidates when the current searched vector satisfies the query constraint. After that, Lines 15-21 iterate over all unvisited neighbors of now_idx , add them to the priority queue, and mark them as visited.

Note that this vanilla algorithm is inefficient when the query vector does not locate near the satisfied vectors. Figure 2(a) presents an example: The searching algorithm goes from the starting point toward the query vector, however, no satisfied vectors can be found in that path. The searching algorithm keeps expanding the searching radius and cannot touch any satisfied vector until it enumerates a lot of very similar but unsatisfied vectors.

Algorithm 1 Vanilla Graph-Based Similarity Search

Input: A Proximity graph $G(V, E)$; a query vector q ; a user-defined query constraint f ; a number of output candidates K .

Output: Top- K vectors for the query q with the constraint f .

1. Initialize a min-heap priority queue pq ; a max-heap priority queue $topk$.
 2. find a random starting point S
 3. $d \leftarrow dist(S, q)$; $pq \leftarrow pq \cup \{(d, S)\}$; $visited \leftarrow \{S\}$
 4. **while** $pq \neq \emptyset$ **do**
 5. $(now_dist, now_idx) \leftarrow pq.pop_min()$
 6. **if** $|topk| = K$ **and** $now_dist > topk.peek_max()$ **then**
 7. **break**
 8. **end if**
 9. **if** $f(now_idx) = \text{true}$ **then**
 10. $topk \leftarrow topk \cup \{now_dist, now_idx\}$
 11. **if** $|topk| > K$ **then**
 12. $topk.pop_max()$
 13. **end if**
 14. **end if**
 15. **for each** $next_idx \in \text{neighbors of } now_idx \text{ in } G$ **do**
 16. **if** $next_idx \notin visited$ **then**
 17. $d \leftarrow dist(next_idx, q)$
 18. $pq \leftarrow pq \cup \{(d, next_idx)\}$
 19. $visited \leftarrow visited \cup \{next_idx\}$
 20. **end if**
 21. **end for**
 22. **end while**
 23. **return** $topk$
-

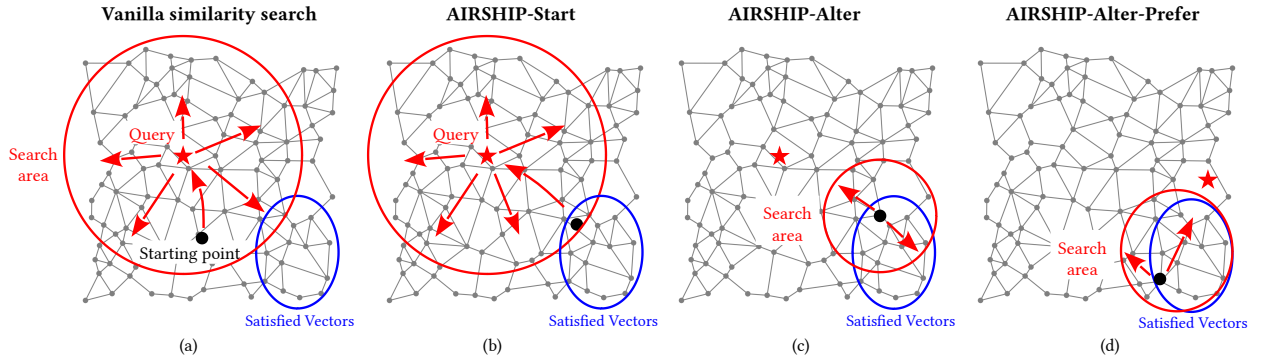


Figure 2: Examples for the vanilla similarity search, AIRSHIP-Alter, and AIRSHIP-Alter-Prefer.

2.2 Starting Points Selection

Our first optimization in the proposed AIRSHIP system is to locate good starting points for the graph searching algorithm. We denote this method as **AIRSHIP-Start**. We would like to start the graph search inside an “interesting region”—a cluster of satisfied vectors. In this scenario (Figure 2(b)), a small number of satisfied vectors can be retrieved along the path to the query vector.

Linear Scan. The most straightforward solution to find satisfied vectors is linear scan that goes over all vectors and checks whether it meets the constraints. Consider the time complexity of evaluating the constraint (a user-defined function) is $O(f)$. The linear scan takes $O(nf)$ time, where n is the number of base vectors. Note that a brute force nearest neighbor search requires $O(nd)$ to compute distances between the query vector and all query vectors. Common constraints (e.g., checking if the attribute of a vector belongs to a range or a collection of categories) takes constant time— $O(1)$. In this case, the time complexity of linear scan is $O(n)$ —it is d times faster than the brute force nearest neighbor search especially when we are tackling with high dimensional data (i.e. $d > 100$). We urge that this linear scan is necessary for selective query constraints—there are only a small/constant number of satisfied vectors. Because we do not know the query constraints in the pre-processing stage, it is infeasible to build an index for each possible constraint. In addition, even if we have prior knowledge of the constraints, it is expensive to construct and store all those indices. The indices roughly require $O(nc)$ time and space, where c is the number of distinct constraints— c can be as large as the number of the queries. In this extreme scenario, after retrieved the small constant number of satisfied vectors, we do not need any advanced techniques other than a brute force distance computation to rank these vectors. Thus, as mentioned above in Section 1, we have the following assumption:

Assumption 1. *For each constraint, at least $p\%$ of base vectors are satisfied vectors. Formally,*

$$|\{v_i | f_j(v_i) \rightarrow \mathbf{true}, v_i \in V\}| \geq \frac{p}{100} \cdot n \quad \forall f_j$$

This assumption holds for real-world recommendation system applications, e.g., the query constraints filter out the retrieved vectors from a collection of categories. When Assumption 1 does not hold, a linear scan can be employed to retrieve the satisfied vectors and brute force distance computation is applied to rank them.

Sampling. Although the linear scan that checks the constraints is faster than a brute force similarity search, it still requires us an inevitable linear time to perform the similarity search. In Here we introduce our second assumption on the data distribution:

Assumption 2. *Satisfied vectors are clustered in a constant number (r) of regions—they are not randomly distributed.*

This assumption commonly holds in the real-world datasets when the constraints are applied to the data labels—vectors with the same label are clustered in the high-dimensional (embedded) space. On the other hand, when the assumption does not hold, consider the case that the satisfied vectors of a given query constraint are randomly distributed, i.e., a vector satisfies the constraint with a fixed probability $prob$. Given this kind of constraints, when we target to find top- k candidates, it is equivalent to retrieve top- $(\frac{k}{prob})$ results in expectation with any similarity search techniques—we can view the fixed satisfied probability $prob$ as the probability to keep a retrieved item in the top candidates. Therefore, no optimization can be applied to the randomly distributed satisfied vectors.

With Assumption 2, we do not have to retrieve all satisfied vectors as the linear scan method, because satisfied vectors are clustered as regions and we can perform similarity search within these regions. Therefore, we are only required to locate a sample of satisfied vectors as searching starting points for the proximity graph searching algorithm. We obtain a sample of s base vectors when we construct the graph index in the pre-processing stage. Due to Assumption 1, we have $p \cdot s$ satisfied vectors in the sample in expectation. The starting point selection in the sample takes $O(s)$ time for constant-time constraints.

Initialization with multiple starting points. After we obtain a collection of satisfied vectors from the sampling method, a straightforward question is: which vector should we use as the starting point? We put all of them into the priority queue. The priority queue extracts the vector that is closest to the query point at the beginning of the search. Instead of simply using the single closest satisfied vector as the starting point, our approach enables other vectors can still be extracted during the search—when the previously extracted vectors connect to the candidates that are far away from the query point.

2.3 Multi-Direction Search

Although AIRSHIP-start alleviates the drawback of the vanilla solution, it is still inefficient when we want to retrieve a larger number of similar vectors (a larger K). Figure 2 shows that only the satisfied vectors along the path to the query vector are retrieved at the beginning of the search. When we try to obtain more satisfied vectors, we still have to first navigate to the query point and then enumerate numerous similar but unsatisfied vectors.

AIRSHIP-Alter. Following this intuition, we propose an optimization AIRSHIP-Alter that enables multi-direction searching. Comparing with the single priority queue used in the original graph search, we have two priority queues to keep search candidates (Algorithm 2). pq_{sat} stores the satisfied vectors and pq_{other} maintains remaining unsatisfied vectors. Line 9 selects a priority queue by Algorithm 3. When both priority queues are not empty, we choose each queue with a weight ratio—*alter_ratio* (Algorithm 3 Line 6). The *alter_ratio* ($0 < alter_ratio \leq 1$) is used to determine the proportion of search candidates from the pq_{sat} . For example, both queues are selected alternatively after the other when *alter_ratio* = 0.5. The greater the *alter_ratio* is, the more biased we are to select the priority queue with satisfied vectors (pq_{sat}). Back to Algorithm 2, Line 10-13 maintain the number of times we select pq_{sat} (cnt_{sat}) and count the total number of processed iterations (cnt_{total}). Once the priority queue is determined, we extract the search candidate from the queue (Line 18-23) as the original searching algorithm. Another major difference from the original searching algorithm is Line 27-31: We insert the neighbors of the current searching vector into two separate priority queues according to their constraint satisfactory—satisfied vectors are assigned to pq_{sat} and the remaining ones are pushed to pq_{other} .

Searching with the candidates from pq_{sat} “exploits” the clusterness of satisfied vectors, while using the candidates from pq_{other} “explores” other regions that close to the query vector. This two-priority-queue design enables us to bias satisfied vectors and have a trade-off between the “exploitation” and “exploration”. Figure 2(c) presents an example when *alter_ratio* = 0.5, we alternatively extract the vectors from two priority queues and a good amount of satisfied vectors are retrieved.

Algorithm 2 AIRSHIP-Alter

Input: A Proximity graph $G(V, E)$; a collection of starting points S ; a query vector q ; a user-defined query constraint f ; a number of output candidates K .

Output: Top- K vectors for the query q with the constraint f .

1. Initialize two min-heap priority queues pq_{sat} and pq_{other} ; a max-heap priority queue $topk$.
2. $visited \leftarrow \emptyset$
3. **for each** $S_i \in S$ **do**
4. $d \leftarrow dist(S_i, q)$
5. $pq_{sat} \leftarrow pq_{sat} \cup \{(d, S_i)\}$
6. $visited \leftarrow visited \cup \{S_i\}$
7. **end for**
8. **while** $pq_{sat} \neq \emptyset$ **and** $pq_{other} \neq \emptyset$ **do**
9. $pq \leftarrow select_priority_queue(pq_{sat}, pq_{other}, cnt_{sat}, cnt_{total}, alter_ratio)$
10. **if** $pq = pq_{sat}$ **then**
11. $cnt_{sat} \leftarrow cnt_{sat} + 1$
12. **end if**
13. $cnt_{total} \leftarrow cnt_{total} + 1$
14. $(now_dist, now_idx) \leftarrow pq.pop_min()$
15. **if** $|topk| = K$ **and** $now_dist > topk.peek_max()$ **then**
16. **break**
17. **end if**
18. **if** $pq = pq_{sat}$ **then**
19. $topk \leftarrow topk \cup \{now_dist, now_idx\}$
20. **if** $|topk| > K$ **then**
21. $topk.pop_max()$
22. **end if**
23. **end if**
24. **for each** $next_idx \in \text{neighbors of } now_idx \text{ in } G$ **do**
25. **if** $next_idx \notin visited$ **then**
26. $d \leftarrow dist(next_idx, q)$
27. **if** $f(next_idx) = \text{true}$ **then**
28. $pq_{sat} \leftarrow pq_{sat} \cup \{(d, next_idx)\}$
29. **else**
30. $pq_{other} \leftarrow pq_{other} \cup \{(d, next_idx)\}$
31. **end if**
32. $visited \leftarrow visited \cup \{next_idx\}$
33. **end if**
34. **end for**
35. **end while**
36. **return** $topk$

2.4 alter_ratio Estimation

As a hyper-parameter, *alter_ratio* can be enumerated and tuned for applications to achieve the best performance. It is also common to have no prior knowledge of the query constraints, i.e., no pre-tuning can be applied. In this section, we derive a low-overhead heuristic algorithm to estimate *alter_ratio*. Intuitively, *alter_ratio* should be positively correlated with the clusteriness of satisfied vectors: The more clustered satisfied vectors, the larger *alter_ratio* should be. In the extreme case when all satisfied vectors are distributed in the same region and no unsatisfied vectors are within this region, *alter_ratio* should be 1 to obtain the optimal performance, as we do not need to explore unsatisfied vectors. Here, we adapt the concept of k-nearest-neighbor statistics (Hastie et al., 2009) as the estimation of *alter_ratio*. With a given constraint f and a collection of satisfied vectors from the samples (SSV), our estimation of *alter_ratio* is:

$$\frac{1}{|SSV|} \sum_{v \in SSV} \frac{|\{v' | f(v') = \mathbf{true} \forall v' \in \text{top-}k_neighbor(v)\}|}{k} \quad (1)$$

where $\text{top-}k_neighbor(v)$ is the nearest k neighbors of vertex v . Note that we do not need any extra computation to obtain these neighbors in the query time. Since the proximity graph approximates a k-nearest-neighbor graph by its definition, we can view the immediate neighbors of v in the proximity graph as its nearest k neighbors. We sort the edges in the proximity graph according to their distance to the vertex. Therefore, no distance computation is required in the query time to obtain the estimation of *alter_ratio*. For each sampled satisfied vectors, we only need to go over its first k edges and check whether its neighbors are satisfied vectors. *alter_ratio* is estimated as the average number of satisfied neighbors among all sampled satisfied vectors.

2.5 Biased Priority Queue Selection

As depicted in Figure 2(d), when the path from the starting point to the query vector are inside the region of satisfied vectors, the exploration of pq_{other} are doing “useless” efforts, as it is not exploring the direction toward the query vector. These ineffective operations are brought by parameter *alter_ratio* that forces us to explore unsatisfied vectors. We propose AIRSHIP-Alter-Prefer to overcome this disadvantage: when the top candidate from the satisfied vector queue is better than the one from the other queue, we override the *alter_ratio* restriction and select the satisfied vector queue. This optimization is realized by adding another `if` statement in the priority queue selection (Algorithm 3):

Algorithm 3 select_priority_queue

Input: Two priority queues pq_{sat} and pq_{other} ; cnt_{sat} ; cnt_{total} ; *alter_ratio*.

Output: A priority queue for the current searching iteration.

1. **if** ($pq_{other} = \emptyset$) **then**
2. $pq \leftarrow pq_{sat}$
3. **else if** $pq_{sat} = \emptyset$ **then**
4. $pq \leftarrow pq_{other}$
5. **else if** $\frac{cnt_{sat}}{cnt_{total}} \leq \text{alter_ratio}$ **then**
6. $pq \leftarrow pq_{sat}$
7. **else**
8. $pq \leftarrow pq_{other}$
9. **end if**
10. **return** pq

if the top candidate from pq_{sat} is better than the one from pq_{other} , we select pq_{sat} without considering $alter_ratio$. This optimization is aggressive: it benefits the searching when the satisfied vectors are highly clustered, while it also reduces the explorations of unsatisfied vectors—what may hurt the performance when top similar satisfied vectors do not reside near the clusters of sampled starting points. We investigate this optimization in Section 3.2.

3 Experimental Evaluation

The objective of the experimental evaluation is to investigate the overall performance and the impact of optimizations of the proposed system AIRSHIP. Specifically, the experiments are designed to answer the following questions:

- How does the proposed system compare with other state-of-the-art similarity search algorithms?
- What is the effect of the proposed optimizations, namely AIRSHIP-Start, AIRSHIP-Alter, and AIRSHIP-Alter-Prefer?
- How is our estimation to $alter_ratio$ on different data distributions? Does searching performance with the estimated ratio match the best enumerated/hand-picked constant?
- How does the data distribution affect the performance of the proposed system?

Implementation. We implement the vanilla graph-based similarity search and AIRSHIP as a C++11 prototype. The code is compiled with g++-7.5.0 enabling the “O3” optimization. The code contains special function calls to harness detailed profiling data.

System. We execute the experiments on a single node server. The server has one Intel(R) Xeon(R) Platinum 8276 CPU @ 2.20GHz (64 bit)—28 cores 56 threads—and 1 TB of memory. Ubuntu 20.04 LTS 64-bit is the operating system.

Baselines. We have two baseline algorithms in the experiments: **Vanilla** and **product quantization (PQ)** (Jégou et al., 2011). Vanilla, which uses the same code base of Zhao et al. (2020), is an adaption of HNSW (Malkov and Yashunin, 2020) to our constrained similarity search problem. We add the query constraint filter before pushing top-k results into the result heap. All our proposed AIRSHIP optimizations are built on top of the vanilla solution. PQ is one of the most popular quantization-based similarity search algorithms. It quantizes the high-dimensional vector to a limited number of integers to accelerate the distance computation. For the PQ comparison, we go over all vectors and check whether they meet the query constraint. Then, top-k similar vectors are chosen by the distances on the quantized vectors instead of the original data vector.

Methodology. We pre-build the index for each algorithm as a pre-processing—the index construction time is not included in the experiments. All experiments are executed with one single thread to have a fair comparison. Since all algorithms support inter-query parallel, we can assume their performances are linearly scalable to the number of threads. We use Euclidean distance as the similarity measure—0 distance represents the highest similarity. The performance of an algorithm is evaluated by two factors: searching time (*Query Per Second*) and retrieval quality (*recall*).

Searching Time. We measure wall-clock time of each algorithm and present the number of completed Query Per Second (throughput) as the searching time measurement. We employ Query Per Second as the metric instead of the execution time of a query batch, because Query Per Second can be compared without normalizing query batch to the same size. All experiments are performed 3 times and we report the average value as the result.

Retrieval Quality. Recall is a commonly-used retrieval quality measurement for similarity search algorithms. Suppose the candidate point set returned by an algorithm is A , and the correct K nearest neighbors of the query is B , recall is defined as: $Recall(A) = \frac{|A \cap B|}{|B|}$. A higher recall corresponds to a better approximation to the correct nearest neighbor result.

Data. Two datasets are used in the experimental evaluation, one un-labeled data with synthetic labels and one real data with real labels. SIFT1m¹ has 1 million 128-dimensional vectors and does not have labels for each data vector. We cluster the vectors using k-means (Jain, 2010) ($k = 10$) into 10 clusters and assign the corresponding cluster id as the label for each vector. The randomness of the label is also studied. Consider $R\%$ randomness, when we assign the label for each vector, we have a probability $R\%$ to set a random label in $\{1, 2, \dots, 10\}$ instead of its corresponding cluster id. We have 0%, 50%, and 100% random labels on SIFT in the following experiments. The dataset with real labels we used is MNIST with 784 dimensions². We use the first 1 million vectors in this paper to have the same base vector size as the SIFT dataset.

Queries. The queries in the original similarity search datasets do not have constraints. We synthesize two categories of constraints. (a) **equal**: the returned similar vectors should have the same label as the query vector; (b) **unequal-X%**: for each query, we randomly choose $X\%$ labels that are unequal to the label of the query vector. The labels of returned similar vectors should be one of these randomly chosen labels. We extract 10,000 vectors from the testing data as the query vectors. The label of the SIFT1m query vector is generated in the same fashion as the base vectors.

3.1 Performance Comparison

Figure 3 depicts the comparison of PQ, vanilla graph-based similarity search, AIRSHIP-Start, and AIRSHIP—AIRSHIP is the algorithm with all proposed optimizations.

Equal label queries. The first row of Figure 3 illustrate the comparison when our query constraints are: the returned similar vectors should have the same label as the query vector. Since the data with the same label is clustered—similar vectors have the same label to the query vector, this kind of constraint has no substantial difference to the original similar search without considering the constraints. Therefore, all 3 graph-based methods have comparable performance. PQ iterates all vectors to locate the satisfied vectors. Since it is linear to the number of base vectors, inferior performance is observed.

Unequal label queries. When the query constraint targets to locate unequal labels—which is the setting of the constrained similarity search, substantial gaps are shown in the second to the fourth row of Figure 3. Choosing a good starting point slightly improve the vanilla solution in **unequal-10%**. For other constraints, vanilla and AIRSHIP-Start have almost identical performance. There is a substantial gap between AIRSHIP and other methods. Note that the y-axis is in the logarithmic scale. For **unequal-10%**, AIRSHIP achieves a consistent 10-100x speedup for all top-1, 10, and 100 tasks. The gap is reduced with more relaxed constraints. In **unequal-20%**, the gap becomes smaller

¹<http://corpus-texmex.irisa.fr/>

²<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#mnist8m>

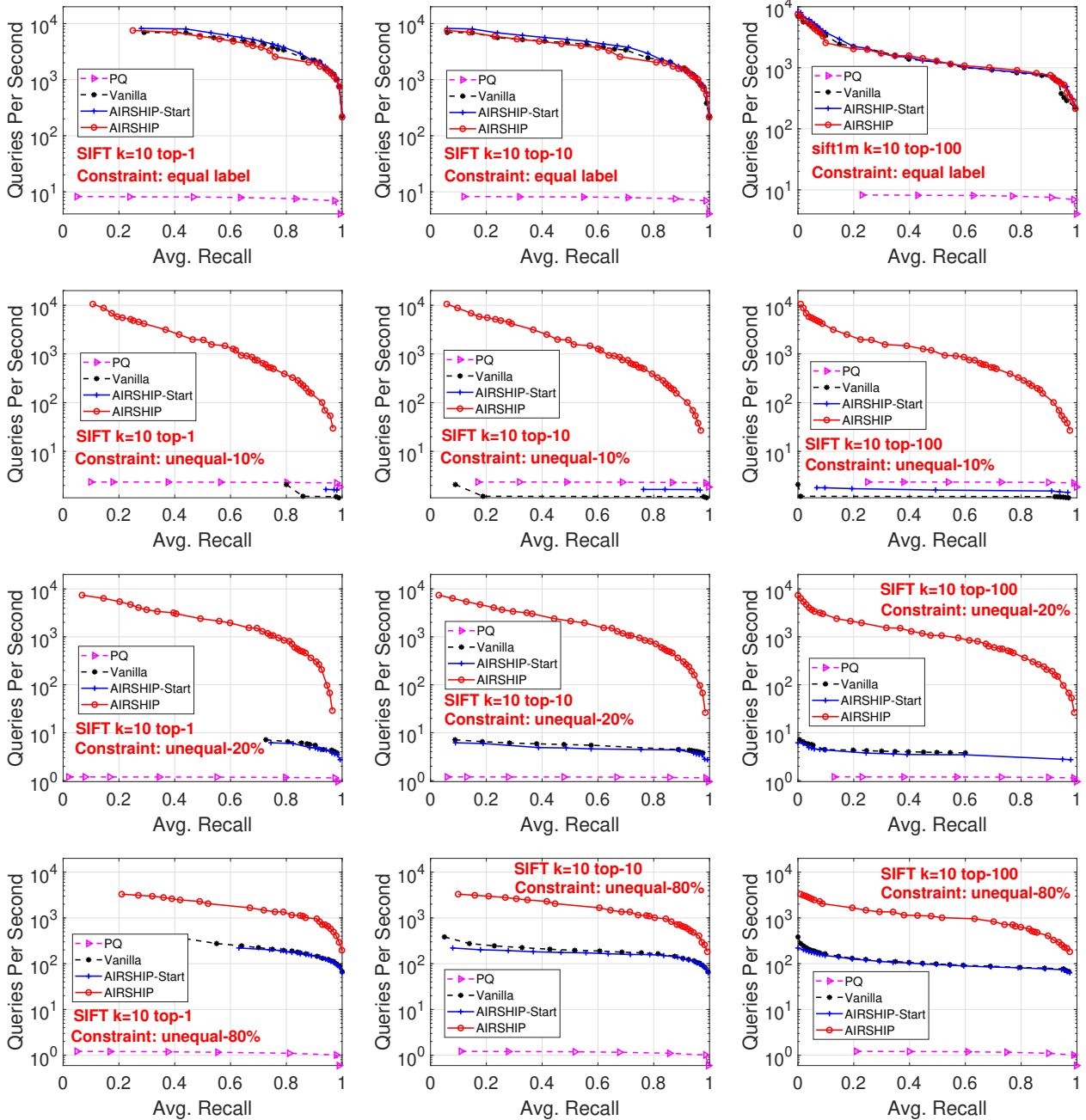


Figure 3: Results on SIFT. The number of cluster $k = 10$. Different constraints are applied.

because there are more satisfied vectors than `unequal-10%`—the vanilla solution and AIRSHIP-Start have a greater chance to find the satisfied vectors. `unequal-80%` is very similar to the unconstrained general similarity search problem. Thus, vanilla and AIRSHIP-Start shows better performance.

AIRSHIP has a consistent execution time across all 4 query constraints. This confirms that our proposed solution is stable for various query constraints without building extra indices.

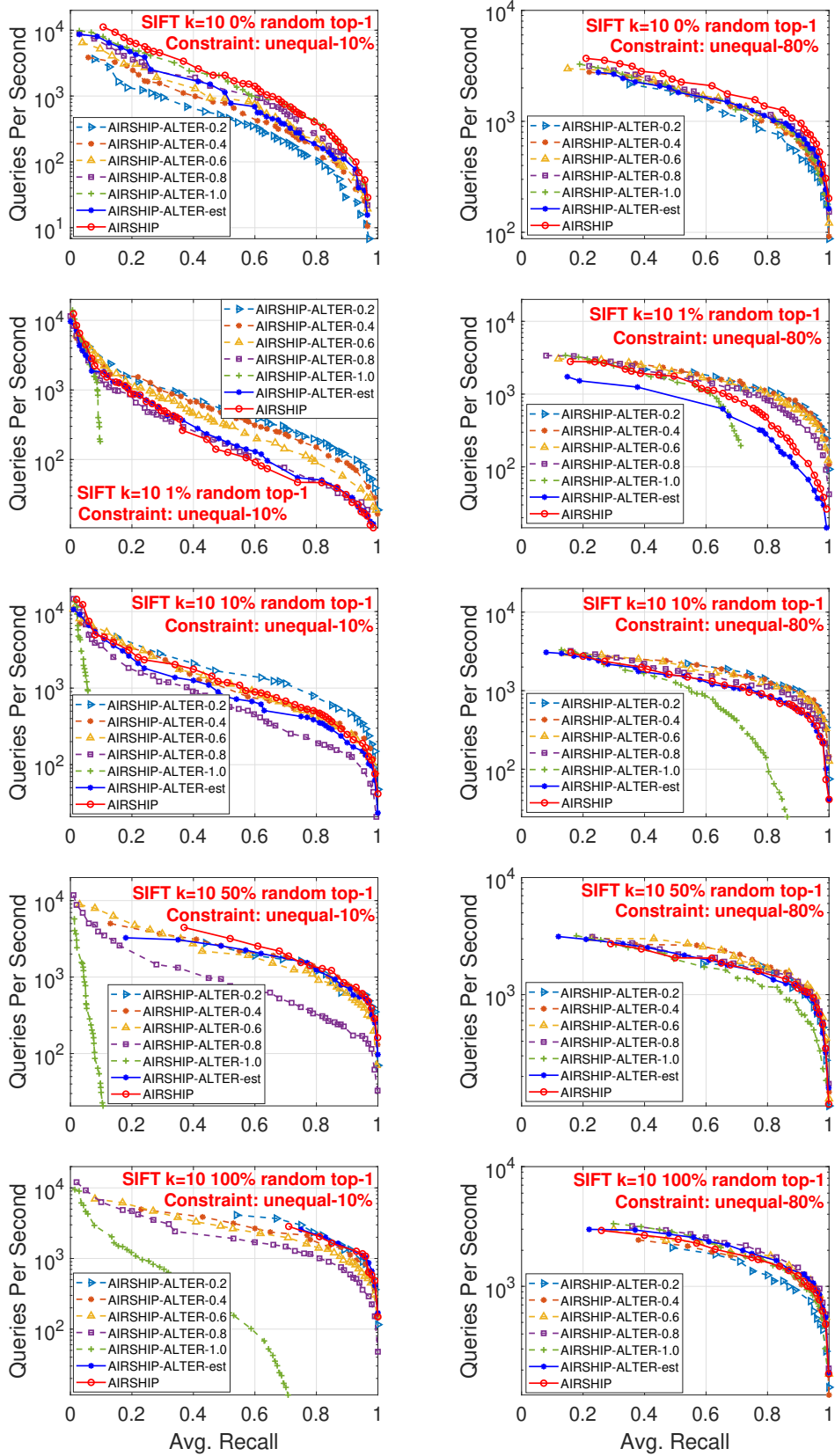


Figure 4: Results for alpha ratio estimation on SIFT over different amounts of random labels.

3.2 Optimization Effectiveness

***alter_ratio* estimation.** We evaluate the estimation of our *alter_ratio* in Figure 4 by comparing the searching performance using the estimated ratio (AIRSHIP-ALTER-est) with methods using constant *alter_ratio*. We enumerate the ratio in 0.2, 0.4, 0.6, 0.8, and 1.0. The comparison is conducted on 3 different label randomness data (0%, 1%, 10%, 50%, and 100%) and 2 query constraints (*unequal-10%* and *unequal-80%*). For clustered data (0%*random*), higher *alter_ratio* yields better performance. When the randomness of the label goes upper, i.e., 50%*and*100%, A low *alter_ratio* gains the best throughput (Queries Per Second). Our estimated *alter_ratio* (AIRSHIP-ALTER-est) shows a comparable performance to the best constant *alter_ratio*. Therefore, we can conclude that our estimation method effectively determines the ratio without having prior knowledge of the query constraint.

Biased priority queue selection. We introduced an aggressive priority queue selection algorithm in Section 2.5: When the top candidate from the satisfied vector queue is better than the one from the other queue, we override the *alter_ratio* restriction and select the satisfied vector queue. We denote this optimization as AIRSHIP-Alter-Prefer or AIRSHIP for short since it consists of all the proposed optimizations. Figure 4 shows that AIRSHIP obtains the best performance in most randomness and query constraint combinations. For 50% and 100% label randomness cases, it was slightly slower than AIRSHIP-ALTER-est since the biased queue selection is designed for the clustered satisfied vectors.

3.3 Number of Satisfied Vector Clusters

In previous experiments, our dataset has $k = 10$ distinct labels that correspond to k clusters of satisfied vectors. It can be viewed as the dataset contains k classes. In Figure 5, we report a comparison when we vary k for 10, 100, and 1,000 distinct labels. The top-1 results illustrate more diverged curves than the top-100 ones because top-1 is more sensitive—we have to find the exact

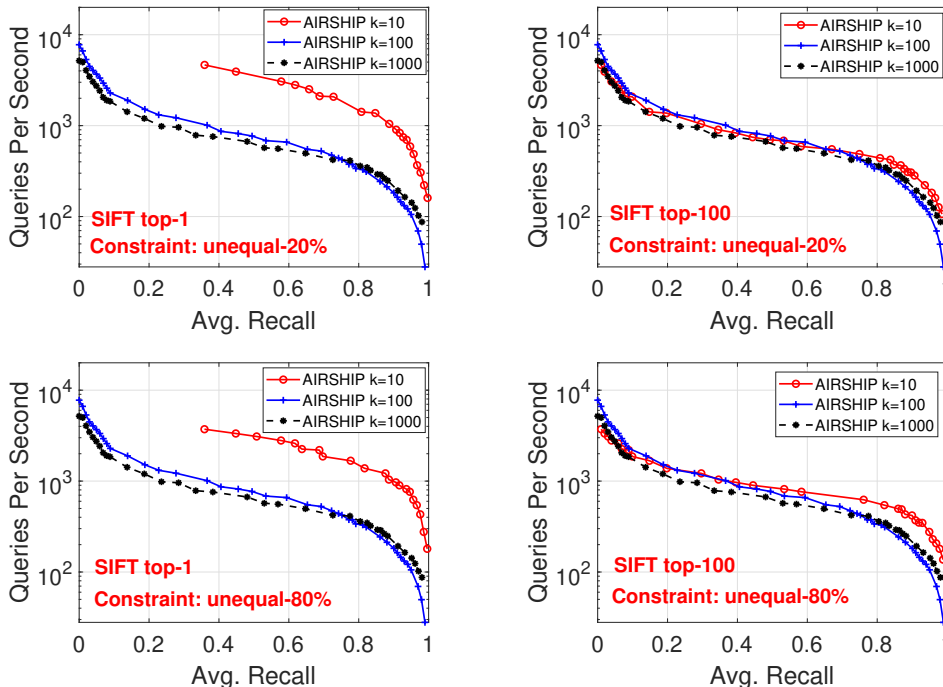


Figure 5: Results for varying the number of clusters, k .

most similar satisfied vector. AIRSHIP has a better performance when the number of labels (classes) is small for top-1 situation, because there are fewer satisfied vector clusters to explore. The difference disappears when we consider top-100 results, since the expected numbers of vectors to be explored are the same for all three cases—the top-100 candidates are distributed across the same proportion (20% or 80%) of satisfied clusters.

3.4 MNIST

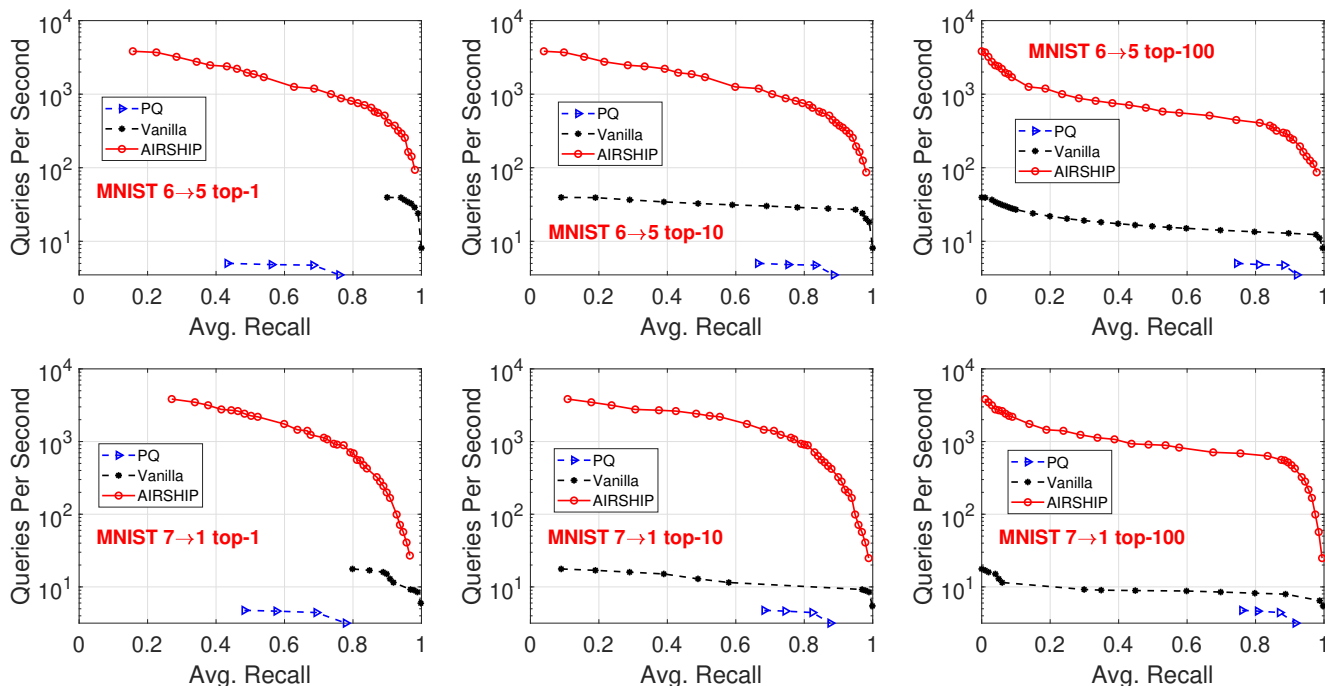


Figure 6: Results on MNIST. (1) searching “5” by “6” ; (2) searching “1” by “7”.

MNIST is a hand-written digits datasets. Each vector in MNIST has its label (10 classes from 0 to 9). We use MNIST as an experiment to check the performance of AIRSHIP on real-world data distribution. Figure 6 shows a performance comparison with two queries. (a) $6 \rightarrow 5$: the query point is from class 6 and the returned similar vectors must be class 5; (b) $7 \rightarrow 1$: given a query vector from class 7 and find the most similar vector from class 1. PQ suffers from computing the distances between all satisfied vectors (10%). AIRSHIP achieves around 100X speedup over the vanilla graph search. The speedup is consistent across top-1, 10, and 100 results. Figure 7 visualizes one query image and its top-3 retrieved candidates for the $6 \rightarrow 5$ constraint. This kind of constrained query can be employed not only in the recommendation system but also in the security context and negative sampling finding tasks. Although this task seems to be simple by partitioning the

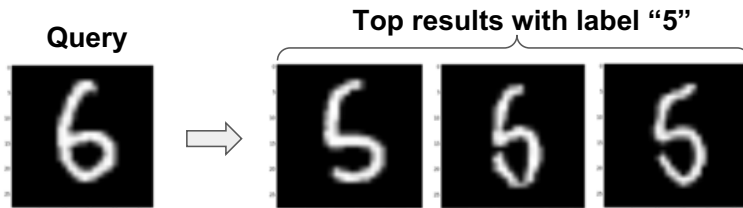


Figure 7: An Example of searching “5” by “6” on MNIST.

base vectors according to their labels. However, there can be around 2^{10} different queries when we target to obtain a vector within a collection of labels—it is time-consuming and space-inefficient to construct this large number of indices. Our proposed algorithm AIRSHIP does not need to build extra indices—we just perform the similarity search on the original proximity graph without worrying about the unknown query constraints.

3.5 Discussion

Based on the results reported above, we can answer the questions driving our experimental evaluation. AIRSHIP substantially improves over other similarity search algorithms, i.e., HNSW, PQ. The speedup can be 10-100X depending on the data distributions. The starting point selection optimization (AIRSHIP-Start) slightly improves over the vanilla graph search algorithm. The multi-direction priority queue search (AIRSHIP-Alter) is the major source of the speedup. The biased priority queue selection is beneficial when the satisfied labels are highly clustered. Our estimation to *alter_ratio* on different data distributions (0%, 1%, 10%, 50%, and 100% randomness) shows a comparable performance compared with enumerated constants. The performance of AIRSHIP is consistent across all combinations of data randomness and query constraints.

4 Conclusion

We investigate the **constrained similarity search** problem, and introduce **AIRSHIP**, a system that integrates a user-defined function filtering into the similarity search framework. The proposed system does not need to build extra indices nor require prior knowledge of the query constraints. We propose three optimizations for the constrained searching problem: starting point selection, multi-direction search, and biased priority queue selection. All three optimizations improve the similarity search performance. The proposed algorithm has a hyper-parameter *alter_ratio* that is data/query dependent. We present an estimation method to adaptively choose this hyper-parameter. Experimentally evaluations of the proposed algorithm on both synthetic and real-world data confirm the effectiveness of our optimizations: AIRSHIP is 10-100X faster than baseline solutions.

The constrained similarity search problem arises from many practical applications in search and advertising when the authors worked with product teams. Obvious solutions in some cases did work well but there are many practical scenarios which required more effective solutions. Evaluating the constraints can be very expensive unless they are properly indexed. When engineers first apply ANN on the original data and apply filtering on the remaining data vectors, they often have to substantially increase the number vectors returned for ANN (e.g., returning 10000 vectors while the target is to find top-1000 vectors which also satisfy the constraints), leading to substantial performance loss. Our proposed solution has been deployed for production by the ads team, The empirical evaluations presented in this paper, however, are solely based on public datasets.

We choose to develop solutions for the constrained similarity search problem, based on graph-based ANN. This is because graph-based ANN is in the production system owing to its excellent performance. Among graph-based ANN, hashing-based ANN, and quantization-based ANN algorithms, typically hashing-based ANN is the simplest in terms of indexing building and the performance is acceptable if a proper hashing function is chosen (Shrivastava and Li, 2014). Nevertheless, if the goal is to optimize for high efficiency (even at the cost of building extra indexing for graphs), then graph-based ANN should be the choice. We expect that researchers and engineers, with non-trivial effects, can also develop solutions for constrained similarity search based on other ANN techniques.

References

- Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Comput. Networks*, 29(8-13):1157–1166, 1997.
- Lawrence Cayton. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the Twenty-Fifth International Conference on Machine learning (ICML)*, pages 112–119, Helsinki, Finland, 2008.
- Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing (STOC)*, pages 380–388, Montreal, Canada, 2002.
- Ryan R Curtin, Parikshit Ram, and Alexander G Gray. Fast exact max-kernel search. In *Proceedings of the 13th SIAM International Conference on Data Mining (SDM)*, pages 1–9, Austin, TX, 2013.
- Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry (SCG)*, pages 253–262, Brooklyn, NY, 2004.
- James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Proceedings of the 2010 ACM Conference on Recommender Systems (RecSys)*, pages 293–296, Barcelona, Spain, 2010.
- Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *International Conference on Learning Representations (ICLR)*, 2020.
- Gideon Dror, Yehuda Koren, Yoelle Maarek, and Idan Szpektor. I want to answer; who has a question?: Yahoo! answers recommender system. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1109–1117, San Diego, CA, 2011.
- Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. New trends in high-d vector similarity search: AI-driven, progressive, and distributed. *Proc. VLDB Endow.*, page 3198–3201, jul 2021.
- Miao Fan, Jiacheng Guo, Shuai Zhu, Shuo Miao, Mingming Sun, and Ping Li. MOBIUS: towards the next generation of query-ad matching in baidu’s sponsored search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 2509–2517, Anchorage, AK, 2019.
- Jerome H. Friedman, F. Baskett, and L. Shustek. An algorithm for finding nearest neighbors. *IEEE Transactions on Computers*, 24:1000–1006, 1975.
- Jerome H. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.
- Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.*, 12(5):461–474, 2019.
- Weihao Gao, Xiangjun Fan, Chong Wang, Jiankai Sun, Kai Jia, Wenzhi Xiao, Ruofan Ding, Xingyan Bin, Hui Yang, and Xiaobing Liu. Deep retrieval: Learning a retrievable structure for large-scale recommendations. *arXiv preprint arXiv:2007.07203*, 2020.

- Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2946–2953, Portland, OR, 2013.
- Kristen Grauman and Rob Fergus. Learning binary hash codes for large-scale image search. In *Machine Learning for Computer Vision*, pages 49–87. 2013.
- Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1312–1317, Barcelona, Catalonia, Spain, 2011.
- Mark Hamilton, Stephanie Fu, Mindren Lu, Johnny Bui, Darius Bopp, Zhenbang Chen, Felix Tran, Margaret Wang, Marina Rogers, Lei Zhang, Chris Hoder, and William T. Freeman. MosAIC: Finding artistic connections across culture with conditional image retrieval. In *Proceedings of NeurIPS 2020 Competition and Demonstration Track*, pages 133–155, 2020.
- Trevor J. Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2 edition, 2009.
- Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 604–613, Dallas, TX, 1998.
- Hosagrahar V Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS)*, 30(2):364–397, 2005.
- Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognit. Lett.*, 31(8):651–666, 2010.
- Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011.
- Anshul Kanakia, Zhihong Shen, Darrin Eide, and Kuansan Wang. A scalable hybrid research paper recommender system for microsoft academic. In *Proceedings of the World Wide Web Conference (WWW)*, pages 2893–2899, San Francisco, CA, 2019.
- Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1042–1050, Vancouver, Canada, 2009.
- Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3270–3278, Boston, MA, 2015.
- Ping Li. Sign-full random projections. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI)*, pages 4205–4212, Honolulu, HI, 2019.
- Ping Li and Kenneth Ward Church. Using sketches to estimate associations. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 708–715, Vancouver, Canada, 2005.

- Ping Li, Xiaoyun Li, Gennady Samorodnitsky, and Weijie Zhao. Consistent sampling through extremal process. In *Proceedings of the Web Conference (WWW)*, pages 1317–1327, Virtual Event / Ljubljana, Slovenia, April 19–23, 2021, 2021.
- Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, page 950–961, Vienna, Austria, 2007.
- Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.
- Yury A. Malkov and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4): 824–836, 2020.
- Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, 2010.
- Parikshit Ram and Alexander G Gray. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 931–939, Beijing, China, 2012.
- Parikshit Ram and Kaushik Sinha. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 1378–1388, Anchorage, AK, 2019.
- Anshumali Shrivastava and Ping Li. Fast near neighbor search in high-dimensional binary data. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD), Part I*, pages 474–489, Bristol, UK, 2012.
- Anshumali Shrivastava and Ping Li. In defense of minhash over simhash. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 886–894, Reykjavik, Iceland, 2014.
- Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. On efficient retrieval of top similarity vectors. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5235–5245, Hong Kong, China, 2019.
- Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. Fast item ranking under neural network based measures. In *Proceedings of the Thirteenth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 591–599, Houston, TX, 2020.
- Shulong Tan, Zhaozhuo Xu, Weijie Zhao, Hongliang Fei, Zhixin Zhou, and Ping Li. Norm adjusted proximity graph for fast inner product retrieval. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1552–1560, Virtual Event, Singapore, 2021a.
- Shulong Tan, Weijie Zhao, and Ping Li. Fast neural ranking on bipartite graph indices. *Proc. VLDB Endow.*, 15(4):794–803, 2021b.

- Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - A survey. *Proc. IEEE*, 104(1):34–57, 2016a.
- Xiaojuan Wang, Ting Zhang, Guo-Jun Qi, Jinhui Tang, and Jingdong Wang. Supervised quantization for similarity search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2018–2026, Las Vegas, NV, 2016b.
- Xinan Wang and Sanjoy Dasgupta. An algorithm for l_1 nearest neighbor search via monotonic embedding. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS)*, page 991–999, Barcelona, Spain, 2016.
- Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N Holtmann-Rice, David Simcha, and Felix Yu. Multiscale quantization for fast similarity search. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5745–5755, Long Beach, CA, 2017.
- Yubao Wu, Ruoming Jin, and Xiang Zhang. Fast and unified local search for random walk based k -nearest-neighbor query in large graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, page 1139–1150, Snowbird, UT, 2014.
- Donna Xu, Ivor W. Tsang, and Ying Zhang. Online product quantization. *IEEE Trans. Knowl. Data Eng.*, 30(11):2185–2198, 2018.
- Zhaozhuo Xu, Weijie Zhao, Shulong Tan, Zhixin Zhou, and Ping Li. Proximity graph maintenance for fast online nearest neighbor search. *arXiv preprint arXiv:2206.10839*, 2022.
- Tan Yu, Jie Liu, Yi Yang, Yi Li, Hongliang Fei, and Ping Li. EGM: enhanced graph-based model for large-scale video advertisement search. In Aidong Zhang and Huzefa Rangwala, editors, *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 4443–4451, Washington, DC, 2022.
- Weijie Zhao, Shulong Tan, and Ping Li. SONG: approximate nearest neighbor search on GPU. In *Proceedings of the 36th IEEE International Conference on Data Engineering (ICDE)*, pages 1033–1044, Dallas, TX, 2020.
- Weijie Zhao, Shulong Tan, and Ping Li. GUITAR: Gradient pruning toward fast neural ranking. *preprint*, 2022.
- Zhixin Zhou, Shulong Tan, Zhaozhuo Xu, and Ping Li. Möbius transformation for fast inner product search on graph. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8216–8227, Vancouver, Canada, 2019.
- Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 1079–1088, London, UK, 2018.
- Han Zhu, Daqing Chang, Ziru Xu, Pengye Zhang, Xiang Li, Jie He, Han Li, Jian Xu, and Kun Gai. Joint optimization of tree-based index and deep model for recommender systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3973–3982, Vancouver, Canada, 2019.
- Jingwei Zhuo, Ziru Xu, Wei Dai, Han Zhu, Han Li, Jian Xu, and Kun Gai. Learning optimal tree models under beam search. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 11650–11659, Virtual Event, 2020.