



Title	IVF++: A Flexible and Efficient Algorithm for Approximate Nearest Neighbor Search under Attribute Constraint
Author(s)	Uemura, Reon; Amagata, Daichi
Citation	IEEE International Conference on Big Data. 2026, p. 7444-7451
Version Type	AM
URL	https://hdl.handle.net/11094/105173
rights	© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

IVF++: A Flexible and Efficient Algorithm for Approximate Nearest Neighbor Search under Attribute Constraint

Reon Uemura

*Graduate School of Information Science and Technology
The University of Osaka
Osaka, Japan
uemura.reon@ist.osaka-u.ac.jp*

Daichi Amagata

*Graduate School of Information Science and Technology
The University of Osaka
Osaka, Japan
amagata.daichi@ist.osaka-u.ac.jp*

Abstract—Many objects are nowadays represented as high-dimensional vectors, and approximate nearest neighbor search (ANNS) on high-dimensional vectors has been receiving attention. Applications, such as recommender systems and retrieval-augmented generation, benefit from this problem. This paper considers its variant, the problem of ANNS under attribute constraint, and assumes that each object is associated with a vector and multiple attributes. Many applications, e.g., e-commerce, have this setting. For example, selling items have images and texts, which are represented as vectors, and are associated with prices, reviews, and dates. Given a query vector and a value for each specified attribute, this problem searches for an approximate nearest object that matches the attribute value constraints. Although this is the standard search setting, existing works (i) assume that each object has only a single attribute or (ii) suffer from high computational costs. The main issue with the existing techniques is that they access objects that do not match the attribute value constraints, incurring unnecessary computational cost. To remove the above limitations and overcome this challenge, we propose IVF++, a new approach that efficiently solves this problem. The building time of our proposed data structure can be only linear to n , where n is the dataset size, and its space complexity can also be linear to n . Furthermore, our ANNS algorithm does not access objects that do not match the attribute value constraints, and it runs in time much faster than $O(n)$ in practice while providing high recall. Our experimental results demonstrate that IVF++ outperforms existing state-of-the-art algorithms.

Index Terms—approximate nearest neighbor search, high-dimensional vector, algorithm

I. INTRODUCTION

Many machine learning models have been developed to represent objects (e.g., users, items, images, and documents, to name a few) as high-dimensional vectors. Due to this trend, the importance of vector data management systems, such as Milvus [1], Vearch [2], Vepsa [3], Weaviate [4], Pinecone [5], and ADBV [6], has been increasing. Approximate nearest neighbor search (ANNS) is one of the main functions in such systems, so many works developed efficient and accurate algorithms for the ANNS problem, including locality-sensitive hashing [7]–[10], quantization [11]–[14], and proximity graphs [15]–[18]. (Because of the curse of dimensionality, the fastest

algorithm for exact NNS in high-dimensional spaces is a linear scan, but this is slow [19].) It is well known that many industrial companies (e.g., Microsoft [20], [21], Apple [22], and Alibaba [6]) employ ANNS algorithms in their search engines.

Recently, to allow flexible search, the problem of ANNS under attribute constraint has been receiving much attention [23]–[29]. In this setting, each object is associated with a vector and attributes, and a given ANNS query focuses only on objects matching the query attribute constraint. Many real-world scenarios prefer this ANNS variant. For example, paper search engines filter results by publication years and conferences. Similarly, e-commerce sites, such as Amazon.com, allow result filtering by price, categories, and rating. This attribute filtering is a standard implementation in these services, suggesting the importance of the problem of ANNS under multi-attribute constraint. This paper addresses this problem.

A. Issues in Existing Techniques

The main requirements of ANNS (with attribute constraints) are efficiency and accuracy. Generally, efficiency and accuracy are in a trade-off relationship. To obtain a good trade-off, existing works [24], [26]–[28], [30], [31] developed some techniques. However, they assume that each object is associated with only a single attribute, and their algorithms are specific to this single attribute case. This assumption significantly limits applications and is far from the practical situations mentioned above (i.e., paper search and e-commerce examples).

Two basic approaches, called “pre-filter” [1], [6] and “post-filter” [32], are general to the number of attributes. Pre-filter first retrieves all objects matching a given attribute constraint and then linearly scans the set of these objects. Although this approach yields the exact answer, it is efficient only when the set size is small. Post-filter first builds an ANNS index offline. Given a query, it runs an ANNS algorithm associated with the index and then removes the objects that do not satisfy the attribute constraint from the search result. If the resulting set size is less than k , post-filter runs additional ANNS by enlarging the search space, e.g., by setting $k' > k$. This

is repeated until we have k objects satisfying the attribute constraints. Clearly, post-filter is also not efficient, because it needs to run ANNS many times and/or set a (very) large k' . To alleviate this inefficiency, “inline processing” [27] traverses the index and computes distance only when accessed objects match the attribute constraints. However, it still accesses many unnecessary objects and suffers from high computational costs.

CAPS [23] tackled the problem of ANNS under multi-attribute constraint and is faster than pre-filter and post-filter. Conceptually, CAPS employs a hybrid approach of pre-filter and inline processing. Given a set of objects, CAPS builds a decision tree based on the frequency of attribute values. Specifically, when CAPS splits a node of the decision tree, its branch condition is determined by the most frequently appearing attribute value. Each node of the decision tree hence contains objects that satisfy the branch conditions. Given a query, CAPS traverses the decision tree based on the query attribute values and arrives at a leaf node. Then, CAPS runs a linear scan of the object set maintained in this node while computing the distance between the query and the object satisfying the query attribute values. CAPS can avoid “some” objects that do not match the query attributes with the decision tree, but the corresponding node still maintains objects that do not satisfy the attribute constraints. Therefore, CAPS also suffers from inefficiency.

B. Contribution

As seen above, the existing works suffer from the single attribute limitation or high computational costs. We overcome these issues and propose a novel algorithm, IVF++, that exploits inverted file structure in two ways and allows an arbitrary number of attributes. Different from CAPS and the other algorithms employing post-filter and inline processing, IVF++ essentially does not access the objects that do not match attribute constraints. In addition, different from pre-filter, we avoid an exhaustive search, i.e., IVF++ does not access the objects that satisfy the attribute constraints but are not similar to the query vector. Thanks to these properties, IVF++ yields highly accurate search results quickly.

To summarize, this paper makes the following contributions:

- We propose IVF++ for the problem of ANNS under multi-attribute constraint. IVF++ consumes only $O(n)$ space and efficiently yields highly accurate search results.
- We conduct experiments on real-world datasets, and the results demonstrate that IVF++ is significantly fast and outperforms the existing state-of-the-art algorithms. For example, IVF++ deals with thousands of queries per second at 99% recall@10 on datasets with a million vectors.

C. Organization

The rest of this paper is organized as follows. Section II introduces preliminary information, including a formal problem definition. Section III presents our solution IVF++. Section IV reports our experimental results, and we review related works in Section V. Finally, Section VI concludes this paper.

II. PRELIMINARY

Let O be a set of n objects, and each object $o \in O$ is associated with a d -dimensional vector $\in \mathbb{R}^d$ and m attributes (e.g., category and color when $m = 2$), where $d = O(1)$ is large. This paper assumes that O is static and resident in RAM [15]–[17], [23]–[26], [28], [33]. Note that $o.v$ is the vector of o , and $o.A_i$ denotes the i -th attribute’s value of o . We assume that each attribute value can be represented as a non-negative integer [23]¹. Let $dist(o, o')$ be the Euclidean distance² between o and o' ($o.v$ and $o'.v$ more precisely, but we use that notation for simplicity). We first define the *exact* version of our problem below.

DEFINITION 1 (NEAREST NEIGHBOR SEARCH WITH ATTRIBUTE CONSTRAINT). *Given O , a query vector q , a set of attribute values $\mathcal{A} = \{q.A_1, \dots, q.A_j\}$, and a result size k , let*

$$O(\mathcal{A}) = \{o \mid o \in O, o.A_i = q.A_i, \forall q.A_i \in \mathcal{A}\}. \quad (1)$$

This problem searches for S^ such that $|S^*| = k$ and $\forall o \in S^*$ and $\forall o' \in O(\mathcal{A}) \setminus S^*$, $dist(o, q) \leq dist(o', q)$. (Ties are broken arbitrarily.)*

Problem definition. We consider an approximation version of the above problem. That is, the problem of approximate nearest neighbor search (ANNS) is to retrieve $S \subseteq O(\mathcal{A})$ such that $|S| = k$ but S may not be equal to S^* . We use recall to measure the accuracy of S , and $recall@k = \frac{|S \cap S^*|}{k}$.

III. IVF++

Main idea. To obtain an approximate but accurate result quickly, we should access only objects that are close to a given query vector q and satisfy the attribute constraints. Existing algorithms suffer from inefficiency since they do not quickly access such objects.

We avoid this issue by proposing a new data structure that allows an arbitrary number of attributes. The building block of our data structure is an inverted file. By exploiting the idea of the inverted file, we partition the dataset O first based on vector similarity and then based on attributes. Thanks to this idea, given a query q , we can first obtain a subset of O that is close to q and then can access *only* objects in this subset that satisfy the attribute constraints. In Section III-A, we present this data structure and how to build this in detail. After that, in Section III-B, we show our ANNS algorithm that exploits this data structure.

A. Data Structure

In IVF++, O is maintained by an inverted file $\mathcal{I}$ defined below.

DEFINITION 2 (INVERTED FILE OF O). *Given K representative objects $c_1, \dots, c_K$, an inverted file $\mathcal{I}$ of O is a set of pairs of a key and a postings list. Each key is a representative object*

¹It is straightforward to convert categorical attributes into an integer domain. Also, negative values can be shifted to a non-negative domain, and continuous values can be scaled to an integer domain (by allowing approximation).

²IVF++ allows other distance functions.

c , and its corresponding postings list is a set of objects whose nearest representative object is c .

From this definition, it is clear that O is partitioned into K disjoint subsets $O_1, \dots, O_K$ such that $O_i \cap O_j = \emptyset$ for $i, j \in [1, K] (i \neq j)$.

We further partition each $O_i (i \in [1, K])$ based on attributes. Recall that each attribute value is a non-negative integer, so the domain of the i -th attribute can be represented by $[0, \max_{o \in O} o.A_i]$. Then we see that $o.A_i$ is represented as a one-hot vector. Its j -th dimension is one if $o.A_i$ corresponds to the j -th value of the domain, and the other dimensions are zero. As each $o \in O$ has m attributes, o has m one-hot vectors. By concatenating these m one-hot vectors, o can have a single bitstring (or a binary vector). IVF++ uses this bitstring to partition O_i and maintains each object in O_i in an inverted file manner.

DEFINITION 3 (INVERTED FILE OF O_i). *The inverted file of $O_i \subset O$ is a set of pairs of a key and a postings list. Each key is a bitstring B obtained from m attribute values, and its corresponding postings list is a set of objects whose bitstrings are B .*

Our data structure is now clear: O is indexed by an inverted file $\mathcal{I}$, and each subset of O is also indexed by an inverted file but in a different manner from $\mathcal{I}$. Although our data structure may be simple, it (i) yields an efficient query processing algorithm as mentioned at the beginning of Section III and (ii) scales well to large datasets because, as introduced later, its space complexity is only $\mathcal{O}(n)$ (with a careful setting).

Offline algorithm. We show our offline algorithm in Algorithm 1. Figure 1 illustrates an overview of our offline step and data structure. Note that our data structure is general to any queries, so this pre-processing is done only once.

We first run a K -means clustering algorithm on O to partition O into $O_1, \dots, O_K$. We obtain K representative objects $c_1, \dots, c_K$ (i.e., cluster centers) and their corresponding clusters. These are maintained by $\mathcal{I}$. Next, we partition each O_i based on attributes. For each $o \in O_i$, we compute its bitstring B . We implement the inverted file of O_i by using a hash table H_i , and we use $H_i[B]$ to denote the postings list of B . We store o in $H_i[B]$. After all objects are stored in their corresponding postings lists, we scan each postings list. If $|H_i[B]|$ is larger than a constant threshold α , we build an ANNS index structure on $H_i[B]$.

Time complexity. By setting $K = \mathcal{O}(1)$ and the number of iterations $= \mathcal{O}(1)$, the K -means clustering takes $\mathcal{O}(n)$ time. The inverted files of $O_1, O_2, \dots$, and O_K are built in $\mathcal{O}(n)$ time because $m = \mathcal{O}(1)$ and the inverted file of O_i is built in $\mathcal{O}(|O_i|)$. If $|H_i[B]| \geq \alpha$, we make an ANNS index on $H_i[B]$, and let $\mathcal{T}(n)$ be the time to make the ANNS index on n vectors. Our pre-processing needs $\mathcal{O}(n) + K \sum \mathcal{T}(|H_i[B]|)$ in the worst case. If $\mathcal{T}(n) = \mathcal{O}(n)$ [34], [35], our offline time is $\mathcal{O}(n)$.

Space complexity. Definitions 2 and 3 suggest that the space complexity of our data structure depends on the hash table of

Algorithm 1: PRE-PROCESSING

Input: O and K (number of clusters)

/ Building a cluster-based inverted file structure of O */*

1 $O_1, O_2, \dots, O_K \leftarrow K\text{-MEANS CLUSTERING}(O, K)$

/ Building an attribute-based inverted file structure for each cluster */*

2 **foreach** $i \in [1, K]$ **do**

3 $H_i \leftarrow$ a hash table of O_i , where key is a binary vector

4 **foreach** $o \in O_i$ **do**

5 $B \leftarrow \emptyset$

6 **foreach** $j \in [1, m]$ **do**

7 Concatenate B and the one-hot vector of $o.A_j$

8 $H_i[B] \leftarrow H_i[B] \cup \{o\}$

9 **foreach** $H_i[B] \in H_i$ such that $|H_i[B]| \geq \alpha$ **do**

10 Build an ANNS index on $H_i[B]$

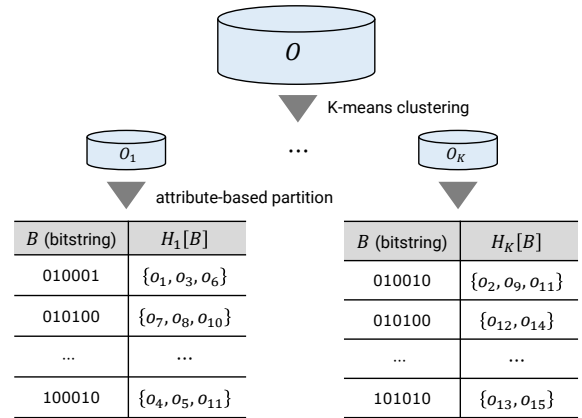


Fig. 1. Overview of pre-processing and our data structure. Assume $m = 2$ and the two attributes' domains are both $[0, 2]$. Each domain can be represented as a 3-dimensional one-hot vector, so the bitstring (concatenated vector) is a 6-dimensional binary vector.

each $O_i \in O$. Clearly, H_i does not have duplicated objects, so its space complexity is $\mathcal{O}(|O_i|)$ if no ANN index is built. Let $\mathcal{S}(n)$ be the space complexity of the ANN index of n vectors. The worst space complexity of our data structure is $K \sum \mathcal{S}(|H_i[B]|)$. Again, if $\mathcal{S}(n) = \mathcal{O}(n)$, our data structure consumes only $\mathcal{O}(n)$ space.

REMARK 1 (WHY INVERTED FILE). *It is well known that the proximity graph-based approach is empirically the current state-of-the-art w.r.t. time-recall trade-off for the standard ANNS problem [19], [36]. However, under the multi-attribute constraint, the performance of this approach significantly degrades. To deal with this constraint, the proximity graph-based approach needs to use post-filter or build a proximity graph for every combination of attribute values. As introduced*

Algorithm 2: ANNS

Input: $\mathcal{I}$, q , $\{q.A_1, \dots, q.A_m\}$, k , and $n_{probe}(\leq K)$

```
1  $S \leftarrow \emptyset$ 
2 Compute  $dist(q, c_i)$  for each  $i \in [1, K]$ , where  $c_i$  is
  the cluster center of  $O_i$ 
3 Sort  $c_1, \dots, c_K$  in ascending order of distance to  $q$ 
4  $B \leftarrow \emptyset$ 
5 foreach  $i \in [1, m]$  do
6   Concatenate  $B$  and the one-hot vector of  $q.A_i$ 
7 foreach  $i \in [1, n_{probe}]$  do
8   Let  $c_i$  be the  $i$ -th cluster center in the above sort
    order
9   if  $|H_i[B]| < \alpha$  then
10    Update  $S$  by a linear scan of  $H_i[B]$ 
11  else
12    Update  $S$  by the ANNS algorithm associated
    with the ANNS index of  $H_i[B]$ 
13 return  $S$ 
```

in Section I, the former case suffers from a larger search space, resulting in a slow response time [23]. The latter case is infeasible due to its significant space requirement and pre-processing time. IVF++ does not have these limitations; its pre-processing and query processing (as presented later) are both efficient with a reasonable space requirement. We therefore employ the inverted file-based structure.

B. Query Processing

We next introduce our ANNS algorithm that exploits the data structure proposed in Section III-A. Given a query vector q , we first compute the distance between q and each cluster center in $\mathcal{I}$. We sort $\{c_1, \dots, c_K\}$ in ascending order of the distance to q to determine which subset is accessed. Then, we compute the bitstring B of the query from $q.A_1, q.A_2, \dots$, and $q.A_m$. IVF++ accesses only the first $n_{probe}(\leq K)$ subsets in the distance order, because the remaining subsets tend to have vectors that are *not* close to q , which can be seen from the K -means cluster structure. (Note that n_{probe} is a parameter that controls the trade-off between time and accuracy.) Let O_i be the i -th subset in the distance order, and we access only $H_i[B]$ (i.e., only objects satisfying the attribute constraints). If $|H_i[B]| < \alpha$, we run a linear scan of $H_i[B]$ to compute an approximate k nearest neighbor search result S . Otherwise, we run the ANNS algorithm associated with the ANNS index to obtain an intermediate S . We do this from the first subset to the n_{probe} -th subset. Algorithm 2 summarizes the above operations.

Time complexity. We analyze the time complexity of Algorithm 2. Lines 2–3 take $\mathcal{O}(1)$ time since $K = \mathcal{O}(1)$ and $m = \mathcal{O}(1)$. Lines 8–12 need at most $\mathcal{O}(|H_i[B]|)$ time. Since $n_{probe} \leq K = \mathcal{O}(1)$ time, Algorithm 2 needs $\mathcal{O}(|H_i[B]|) \ll \mathcal{O}(n)$ time.

IV. EXPERIMENT

This section reports our experimental results. All experiments were conducted on a Ubuntu 22.04 LTS machine with Intel Core i9-9980XE@3.0GHz CPU and 128GB RAM.

A. Setting

Dataset. We used real vector datasets, Deep [37] and GloVe [38], which are commonly used in ANNS works [19], [24], [25], [27], [36]. Deep consists of 10 million 96-dimensional deep feature vectors. GloVe consists of 1.2 million 100-dimensional word embedding vectors.

As with existing studies [1], [6], [23], [24], [27], [28], for each $o \in O$ and for each $i \in [1, m]$, $o.A_i$ is set to a random value. By default, $m = 2$. Note that A_i ($i \in [1, m]$) follows a uniform or Gaussian distribution. Therefore, when $m = 2$, there are four combinations: (1) A_1 and A_2 follow a uniform distribution, (2) A_1 follows a uniform distribution, whereas A_2 does a Gaussian one, (3) A_1 follows a Gaussian distribution, whereas A_2 does a uniform one, and (4) A_1 and A_2 follow a Gaussian distribution. We tested these four combinations.

We picked random vectors from these datasets and used them as query vectors. (These query vectors were removed from O .) Query attribute values follow a uniform distribution.

Evaluated algorithms. We evaluated the following algorithms.

- IVF-inline [27]: the baseline algorithm introduced in Section I. We used FAISS IVF [39] as its ANNS index.
- CAPS [23]: the state-of-the-art algorithm for our problem. We used the original source code.
- IVF++: our algorithm presented in Section III. We set $K = 20$ and $\alpha = 10,000$.

The above algorithms were implemented in C++ and compiled by g++ 11.4.0 with -O3 optimization. We did not use pre-filter and post-filter (with proximity graph) because they are slower than CAPS [23] (recall that inline processing is an optimized version of post-filter). Our experiments focused on IVF-based algorithms, and considering graph indices is future work.

The index structures of the above algorithms were built offline by using multi-threading (36 threads with hyper-threading). ANNS was done in a single-thread mode.

Evaluation criteria. We measured the average running time and recall of the queries used while varying the time-accuracy control parameter of each algorithm. We show recall-QPS (Queries Per Second) curves of the evaluated algorithms³.

B. Result

Pre-processing time. Table I exhibits the pre-processing time of each algorithm when A_1 and A_2 follow Gaussian distribution. The other attribute distribution settings show the same tendencies as Table I, so we omit them because of the space limitation.

³Pre-processing time and memory usage were also measured. On Deep, the pre-processing time of IVF++ was about 5 minutes, whereas its memory usage was 44 [MB]. It can be seen that IVF++ can build its data structure within a reasonable time even on large datasets, and the data structure is easy to fit into modern RAM.

TABLE I
PRE-PROCESSING TIME [SEC]

Dataset	Deep	GloVe	SIFT
IVF-inline	306.56	40.02	28.07
CAPS	387.10	47.59	34.35
IVF++	317.68	40.94	29.77

TABLE II
MEMORY CONSUMPTION [MB]

Dataset	Deep	GloVe	SIFT
IVF-inline	50.23	7.18	5.10
CAPS	626.76	74.59	62.50
IVF++	44.35	8.35	8.93

IVF-inline and IVF++ show similar pre-processing times because they share the dataset partition based on K -means clustering (whereas IVF-inline does not do further partitioning). As IVF++ builds other inverted files, its pre-processing time is slightly longer than that of IVF-inline, but the difference is slight. CAPS needs a longer pre-processing time than the others. This result implies that its decision tree needs a longer time to build.

QPS vs. recall@10. From here, we focus on query processing (online) time and accuracy of each algorithm. Figures 2 and 3 illustrate QPS–recall curves of the evaluated algorithms on Deep and GloVe. In these experiments, we set $k = 10$.

The first observation is that IVF++ outperforms the competitors, i.e., it has a better time–accuracy trade-off than CAPS and IVF-inline. For example, when recall@10 ≈ 0.95 , IVF++ is about 5 (840), 6 (765), and 4 (811) times faster than CAPS (IVF-inline) on Deep, GloVe, and SIFT, respectively. This result is derived from the data structure of IVF++. IVF-inline and CAPS need to access objects that do not satisfy the attribute constraints, whereas IVF++ does not. This advantage provides our state-of-the-art performance. Among these algorithms, IVF++ is the only one that yields more than 1000 QPS at recall@10 ≈ 0.9 .

The second observation is that the performance tendency of IVF++ is not affected by the distributions of the attribute values. Recall that the time complexity of IVF++ is $\mathcal{O}(|H_i[B]|)$, which suggests that the time of IVF++ depends on the size of a set of objects having B in the first n_{probe} subsets. Therefore, it is trivial that, regardless of the distribution, a larger (less) n_{probe} yields a slower but more accurate (faster but less accurate) result.

Space consumption of data structure. Table II shows the memory consumption of each algorithm with the same setting as in the pre-processing time experiment. The result is similar to that in Table I. CAPS consumes much more memory than the others. This observation suggests that our data structure introduced in Definitions 2 and 3 is much better than the state-of-the-art w.r.t. space requirement.

Impact of k . We next investigate the impact of k . We mea-

sured QPS when recall@ $k \approx 0.9$. In this experiment, each attribute value followed a Gaussian distribution. Figure 5 shows the result.

It can be seen that as k increases, the QPS of each algorithm decreases. This is a reasonable result because, as k increases, the search space to achieve recall@ $k \approx 0.9$ becomes larger. However, the degradation of QPS is only slight, and IVF++ keeps outperforming the competitors and a sufficiently large QPS.

Impact of m . Algorithms for the problem of ANNS under multi-attribute constraint should be robust against m (the number of attributes). To investigate this robustness, we measured the QPS of each algorithm when recall@10 ≈ 0.9 and each attribute value follows a Gaussian distribution. Figure 6 depicts the result.

The main observation is that the performance of IVF++ (CAPS) increases (decreases) as m increases. This observation clarifies the drawback of CAPS, and it is clear that the decision tree cannot yield high selectivity when m becomes larger. IVF++’s tendency is desirable for our problem because the tendency means robustness against m . As m increases, the number of objects that match the attribute constraints tends to decrease. Therefore, $|H_i[B]|$ becomes smaller, and it is now easy to understand why we have this observation from the time complexity of IVF++.

V. RELATED WORK

A. ANNS

Many studies devised efficient and accurate ANNS algorithms, and ANNS is a building block of many important operations, such as recommender systems [40], EMST [41], multi-metric search [42], joins [43], clustering [44], [45], outlier detection [34], and distance estimation [46]. Many studies devised efficient and accurate ANNS algorithms. Locality-sensitive hashing is a theoretical approach, but, in practice, slower than the other approaches [19]. Quantization is space-efficient but cannot provide high recall [13]. Proximity graphs are the current state-of-the-art approach [36]. Roughly, existing proximity graphs belong to the KGraph-based type or the navigable small-world graph-based one. Vamana [20] and HNSW [17] are famous proximity graphs and belong to the former and latter types, respectively. They have been extended for the attribute case.

B. ANNS under Attribute Constraint

As mentioned in Section I, this problem has also been well studied. Most existing works employ proximity graphs [24]–[28]. These works assume a single attribute and consider how to optimize post-filter by carefully designing edge addition strategies. However, using these strategies in multi-attribute cases is not trivial, because there are a significant number of attribute value combinations. Our data structure does not suffer from this observation. CAPS [23] and the standard inverted file, which is employed in vector data management systems (e.g., [1], [6]), also do not have the above limitation. However,

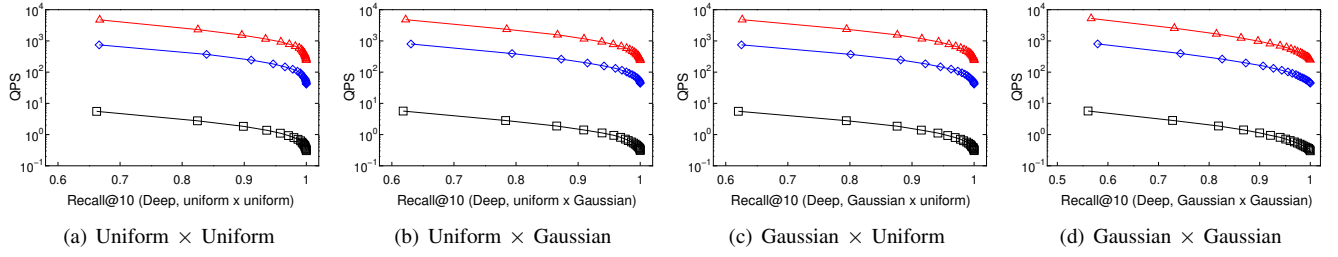


Fig. 2. QPS vs. recall@10 on Deep. “ $\square$ ” shows IVF-inline, “ $\diamond$ ” shows CAPS, and “ $\triangle$ ” shows IVF++.

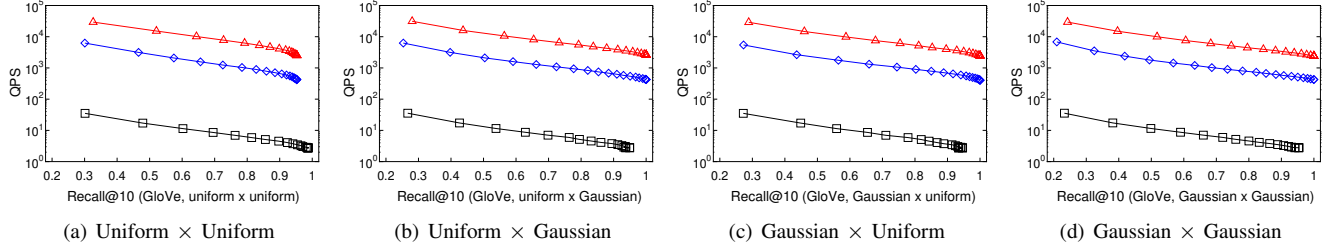


Fig. 3. QPS vs. recall@10 on GloVe. “ $\square$ ” shows IVF-inline, “ $\diamond$ ” shows CAPS, and “ $\triangle$ ” shows IVF++.

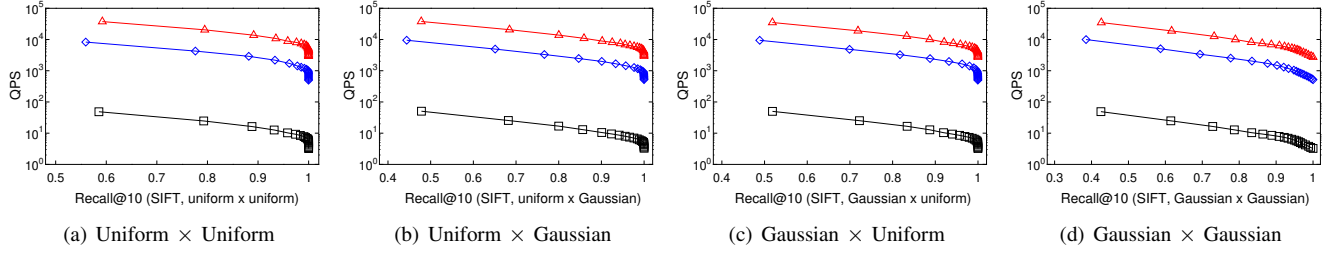


Fig. 4. QPS vs. recall@10 on SIFT. “ $\square$ ” shows IVF-inline, “ $\diamond$ ” shows CAPS, and “ $\triangle$ ” shows IVF++.

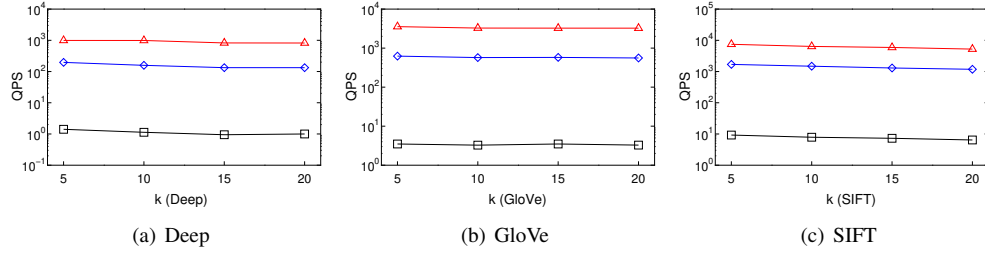


Fig. 5. Impact of k when recall $\approx$ 0.9. “ $\square$ ” shows IVF-inline, “ $\diamond$ ” shows CAPS, and “ $\triangle$ ” shows IVF++.

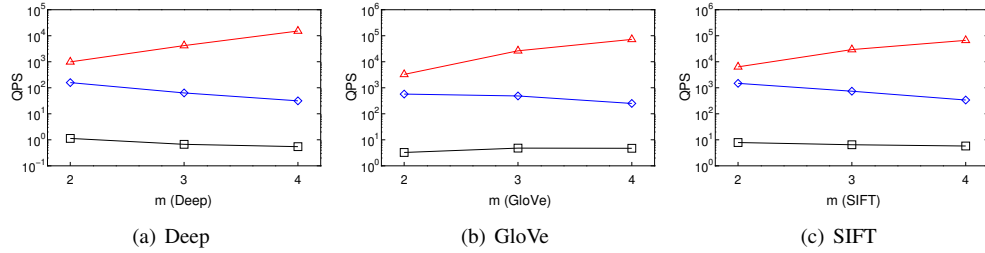


Fig. 6. Impact of m when recall@10 $\approx$ 0.9. “ $\square$ ” shows IVF-inline, “ $\diamond$ ” shows CAPS, and “ $\triangle$ ” shows IVF++.

they still incur inefficient computation and are outperformed by IVF++.

C. Range-filtered ANNS

Recently, ANNS with a range-filter, which focuses on objects with attribute values falling in a given range, has also been addressed in several studies [47], [48]. These works also assume that each object has a single attribute, and their techniques are optimized in this setting. We put extending IVF++ for range-filter as future work.

VI. CONCLUSION

This paper addressed the problem of approximate nearest neighbor search under multi-attribute constraint. Although this problem has many real-world applications, existing techniques cannot solve this problem efficiently. We overcame this issue and proposed a novel algorithm IVF++. Our experimental results demonstrate that IVF++ significantly outperforms state-of-the-art algorithms.

ACKNOWLEDGMENTS

This work was partially supported by AIP Acceleration Research JPMJCR23U2, JST.

REFERENCES

- [1] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu *et al.*, “Milvus: A purpose-built vector data management system,” in *SIGMOD*, 2021, pp. 2614–2627.
- [2] <https://github.com/vearch/vearch>.
- [3] <https://github.com/vespa-engine>.
- [4] <https://weaviate.io/>.
- [5] <https://www.pinecone.io/>.
- [6] C. Wei, B. Wu, S. Wang, R. Lou, C. Zhan, F. Li, and Y. Cai, “Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data,” *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3152–3165, 2020.
- [7] B. Zheng, Z. Xi, L. Weng, N. Q. V. Hung, H. Liu, and C. S. Jensen, “Pm-lsh: A fast and accurate lsh framework for high-dimensional approximate nn search,” *Proceedings of the VLDB Endowment*, vol. 13, no. 5, pp. 643–655, 2020.
- [8] J. Wei, B. Peng, X. Lee, and T. Palpanas, “DET-LSH: A locality-sensitive hashing scheme with dynamic encoding tree for approximate nearest neighbor search,” *Proceedings of the VLDB Endowment*, vol. 17, no. 9, pp. 2241–2254, 2024.
- [9] Y. Tian, X. Zhao, and X. Zhou, “Db-lsh: Locality-sensitive hashing with query-based dynamic bucketing,” in *ICDE*, 2022, pp. 2250–2262.
- [10] —, “Db-lsh 2.0: Locality-sensitive hashing with query-based dynamic bucketing,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 3, pp. 1000–1015, 2024.
- [11] T. Ge, K. He, Q. Ke, and J. Sun, “Optimized product quantization,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 744–755, 2013.
- [12] J. Gao and C. Long, “Rabbitq: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search,” *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–27, 2024.
- [13] R. Wang and D. Deng, “Deltapq: Lossless product quantization code compression for high dimensional similarity search,” *Proceedings of the VLDB Endowment*, vol. 13, no. 13, pp. 3603–3616, 2020.
- [14] Y. Fu, C. Chen, X. Chen, W.-F. Wong, and B. He, “Optimizing the number of clusters for billion-scale quantization-based nearest neighbor search,” *IEEE Transactions on Knowledge and Data Engineering*, 2024.
- [15] Y. Peng, B. Choi, T. N. Chan, J. Yang, and J. Xu, “Efficient approximate nearest neighbor search in multi-dimensional databases,” *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–27, 2023.
- [16] C. Fu, C. Wang, and D. Cai, “High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 4139–4150, 2021.
- [17] Y. A. Malkov and D. A. Yashunin, “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2018.
- [18] Y. Arai, D. Amagata, S. Fujita, and T. Hara, “Lgtm: A fast and accurate knn search algorithm in high-dimensional spaces,” in *DEXA*, 2021, pp. 220–231.
- [19] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, “Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1475–1488, 2019.
- [20] S. J. Subramanya, Devvrit, R. Kadekodi, R. Krishnaswamy, and H. V. Simhadri, “Diskann: Fast accurate billion-point nearest neighbor search on a single node,” in *NeurIPS*, 2019, pp. 13 766–13 776.
- [21] Q. Chen, B. Zhao, H. Wang, M. Li, C. Liu, Z. Li, M. Yang, and J. Wang, “Spann: Highly-efficient billion-scale approximate nearest neighborhood search,” in *NeurIPS*, 2021, pp. 5199–5212.
- [22] J. Mohoney, A. Pacaci, S. R. Chowdhury, A. Mousavi, I. F. Ilyas, U. F. Minhas, J. Pound, and T. Rekatsinas, “High-throughput vector similarity search in knowledge graphs,” *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1–25, 2023.
- [23] G. Gupta, J. Yi, B. Coleman, C. Luo, V. Lakshman, and A. Shrivastava, “Caps: A practical partition index for filtered similarity search,” *arXiv preprint arXiv:2308.15014*, 2023.
- [24] L. Patel, P. Kraft, C. Guestrin, and M. Zaharia, “Acorn: Performant and predicate-agnostic search over vector embeddings and structured data,” *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–27, 2024.
- [25] M. Wang, L. Lv, X. Xu, Y. Wang, Q. Yue, and J. Ni, “An efficient and robust framework for approximate nearest neighbor search with attribute constraint,” in *NeurIPS*, 2023, pp. 15 738–15 751.
- [26] W. Zhao, S. Tan, and P. Li, “Constrained approximate similarity search on proximity graph,” *arXiv preprint arXiv:2210.14958*, 2022.
- [27] S. Gollapudi, N. Karia, V. Sivashankar, R. Krishnaswamy, N. Begwani, S. Raz, Y. Lin, Y. Zhang, N. Mahapatro, P. Srinivasan *et al.*, “Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters,” in *Web Conference*, 2023, pp. 3406–3416.
- [28] W. Wu, J. He, Y. Qiao, G. Fu, L. Liu, and J. Yu, “Hqann: Efficient and robust similarity search for hybrid queries with structured and unstructured constraints,” in *CIKM*, 2022, pp. 4580–4584.
- [29] W. Yang, T. Li, G. Fang, and H. Wei, “Pase: Postgresql ultra-high-dimensional approximate nearest neighbor search extension,” in *SIGMOD*, 2020, pp. 2241–2253.
- [30] Y. Xu, J. Gao, Y. Gou, C. Long, and C. S. Jensen, “irangegraph: Improvising range-dedicated graphs for range-filtering nearest neighbor search,” vol. 2, no. 6, pp. 1–26, 2024.
- [31] A. Liang, P. Zhang, B. Yao, Z. Chen, Y. Song, and G. Cheng, “Unify: Unified index for range filtered approximate nearest neighbors search,” *Proceedings of the VLDB*, vol. 18, no. 4, pp. 1118–1130, 2025.
- [32] S. Yu, J. Engels, Y. Huang, and J. Shun, “Pecann: Parallel efficient clustering with graph-based approximate nearest neighbor search,” *arXiv preprint arXiv:2312.03940*, 2023.
- [33] W. Dong, C. Moses, and K. Li, “Efficient k-nearest neighbor graph construction for generic similarity measures,” in *World Wide Web*, 2011, pp. 577–586.
- [34] D. Amagata, M. Onizuka, and T. Hara, “Fast and exact outlier detection in metric spaces: A proximity graph-based approach,” in *SIGMOD*, 2021, pp. 36–48.
- [35] —, “Fast, exact, and parallel-friendly outlier detection algorithms with proximity graph in metric spaces,” *The VLDB Journal*, vol. 31, no. 4, pp. 797–821, 2022.
- [36] M. Wang, X. Xu, Q. Yue, and Y. Wang, “A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search,” *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 1964–1978, 2021.
- [37] A. Babenko and V. Lempitsky, “Efficient indexing of billion-scale datasets of deep descriptors,” in *CVPR*, 2016, pp. 2055–2063.
- [38] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *EMNLP*, 2014, pp. 1532–1543.

- [39] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, “The faiss library,” *arXiv preprint arXiv:2401.08281*, 2024.
- [40] Z. Li, D. Amagata, Y. Zhang, T. Hara, S. Haruta, K. Yonekawa, and M. Kurokawa, “Mutual information-based preference disentangling and transferring for non-overlapped multi-target cross-domain recommendations,” in *SIGIR*, 2024, pp. 2124–2133.
- [41] K. Kido, D. Amagata, and T. Hara, “Fast approximation algorithm for euclidean minimum spanning tree building in high dimensions,” in *PAKDD*, 2025, pp. 432–443.
- [42] R. Uemura, D. Amagata, and T. Hara, “An efficient framework for approximate nearest neighbor search on high-dimensional multi-metric data,” in *SISAP*, 2024.
- [43] H. Nakama, D. Amagata, and T. Hara, “Approximate top-k inner product join with a proximity graph,” in *IEEE Big Data*, 2021, pp. 4468–4471.
- [44] D. Amagata, “Scalable and accurate density-peaks clustering on fully dynamic data,” in *IEEE Big Data*, 2022, pp. 445–454.
- [45] D. Amagata and T. Hara, “Fast density-peaks clustering: Multicore-based parallelization approach,” in *SIGMOD*, 2021, pp. 49–61.
- [46] D. Amagata, Y. Arai, S. Fujita, and T. Hara, “Learned k-nn distance estimation,” in *SIGSPATIAL*, 2022, pp. 1–4.
- [47] J. Engels, B. Landrum, S. Yu, L. Dhulipala, and J. Shun, “Approximate nearest neighbor search with window filters,” in *ICML*, 2024.
- [48] C. Zuo, M. Qiao, W. Zhou, F. Li, and D. Deng, “Serf: Segment graph for range-filtering approximate nearest neighbor search,” *Proceedings of the ACM on Management of Data*, vol. 2, no. 1, pp. 1–26, 2024.