



Title	An Efficient Framework for Approximate Nearest Neighbor Search on High-Dimensional Multi-metric Data
Author(s)	Uemura, Reon; Amagata, Daichi; Hara, Takahiro
Citation	Lecture Notes in Computer Science. 2024, 15268, p. 3-17
Version Type	AM
URL	https://hdl.handle.net/11094/105172
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

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

The University of Osaka

An Efficient Framework for Approximate Nearest Neighbor Search on High-dimensional Multi-metric Data

Reon Uemura¹, Daichi Amagata¹^[0000-0001-8571-4931], and Takahiro Hara¹^[0000-0003-4807-3156]

Osaka University, Osaka, Japan

{uemura.reon, amagata.daichi, hara}@ist.osaka-u.ac.jp

Abstract. Many objects are often multi-modal data consisting of images, videos, documents, etc, where each modal exists in a different metric space. Similarity searches on multi-modal data are widely employed in information retrieval and machine learning. In these applications, each modal is usually represented as dense high-dimensional data (e.g., an embedding vector). This paper addresses the problem of nearest neighbor search (NNS) on high-dimensional multi-metric data. Although some techniques for NNS on multi-metric data exist, they do not consider high-dimensional data and query-dependent weights in multi-metric spaces. A straightforward yet fast algorithm for “exact” NNS on high-dimensional multi-metric data is linear scan, but obtaining exact results is slow on large datasets. We therefore propose an efficient framework for approximate NNS on high-dimensional multi-metric data. For each metric space, we build the same type of proximity graph that allows any search-start node. An approximate NN is found by traversing the proximity graphs while carefully selecting start nodes to avoid redundant node accesses. We conduct experiments on real-world high-dimensional multi-metric data, and the results demonstrate that our framework outperforms state-of-the-art algorithms.

Keywords: multi-metric data · approximate k NN search · high-dimensions.

1 Introduction

Given a query object q and a set O of objects, the k nearest neighbor search (k -NNS) problem finds k most similar objects $\in O$ to q based on a given distance function. This problem has many applications, such as computer vision [21], recommender systems [4], geographic information systems [3], and machine learning [22], to name a few.

Motivation. The number of objects has been still increasing, and, at the same time, objects are often represented as *multi-modal* data consisting of images, videos, documents, etc., where each modal exists in a different metric space [10–12, 16, 19, 23, 27, 35]. The importance of these objects has been accelerated because of the proliferation of multi-modal searches. For example, VLM (Vision

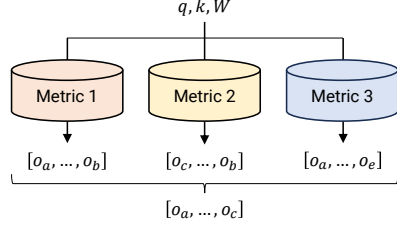


Fig. 1. Example of the separate approach. For each metric space, a k -NNS is done, and the final result is obtained by merging the result in each metric space.

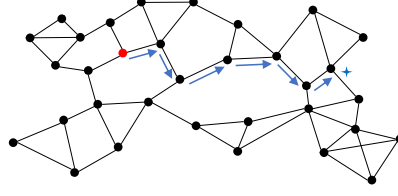


Fig. 2. Example of a proximity graph and greedy search. The red point is a start node, arrows represent graph traversal, and the blue cross is a query.

Language Model) [32] accurately deals with text-to-image processing, and Sora¹ effectively handles text-to-video processing. The high accuracy of these models is supported by training sets consisting of huge numbers of multi-modal data. Notice that the similarity between multi-modal data cannot be measured by a single metric, and it needs to focus on multi-metric spaces. That is, similarity searches on multi-modal data need to aggregate multiple metrics.

Motivated by the above observation, this paper addresses the k -NNS problem in multi-metric spaces. Given O , q , k , and $W = \langle w_1, \dots, w_m \rangle$ (a set of weights for the metric spaces), this problem finds the k objects $\in O$ that have the minimum aggregate distance to q . (A formal definition appears in Section 2.) Azure AI Search² is a concrete implementation of this problem.

Challenge. There are two approaches to solving our problem. One is the integrated approach which integrates the multi-metric spaces into a single index [11, 12, 16, 34]. The other is the separate approach, which runs a k -NNS for each metric [10, 19, 27, 35]. The integrated approach does not take W into account. For example, even if a given query ignores some metric spaces, this approach cannot ignore them. The separate approach, which is shown in Fig. 1, does not have this issue, so this paper considers this approach. Next, we need to consider how to accelerate k -NNS in each metric space.

Notice that each element of multi-modal data is represented as a dense high-dimensional vector due to recent embedding techniques. Therefore, each object $\in O$ has multiple high-dimensional vectors. In this setting, it is well-known that the fastest “exact” algorithm for our problem is a linear scan of O [24]. However, this approach is slow when $|O|$ is large. To address this issue, existing works proposed *approximation* algorithms based on locality-sensitive hashing [28], quantization [31], and proximity graphs [25]. Among these approaches, many empirical studies show that proximity graphs provide the best speed-accuracy trade-off.

The above discussion suggests that a promising approach is to employ a proximity graph for each metric space. In a proximity graph, each node is an object $\in O$, and there are edges between similar objects (nodes). A k -NNS is

¹ <https://openai.com/sora>

² <https://azure.microsoft.com/en-us/products/ai-services/ai-search>

done by a greedy search, which, from a (random) start node, iteratively traverses nodes closer to a given query, as shown in Fig. 2. Note that a straightforward implementation of this approach (i.e., the separate approach with proximity graphs) has two drawbacks. One is about the *start node*. As seen in Fig. 2, if a start node is far from a given query, we need to traverse many nodes (i.e., incur many distance computations). The other is about *full separation between metric spaces*. If we run a k -NNS for each metric space *independently*, we may access the same nodes at most m times, where m is the number of metrics, because the proximity graph in each metric space consists of the same nodes (objects).

Contribution. The above drawbacks incur redundant distance computations, so we overcome these by proposing an efficient framework for our problem. For each metric space, we introduce *representative nodes* that are picked so that they exist uniformly in the space, aiming at finding close start nodes for arbitrary queries. Furthermore, they are used to determine the access order of metric spaces. Our framework avoids the full separation between metric spaces and exploits an intermediate k -NNS result to reduce the search space when we move to the next metric space.

To summarize, our contributions are as follows. (i) We propose a new framework for the k -NNS problem in high-dimensional multi-metric spaces. (ii) We conduct experiments on real-world datasets, and the results demonstrate that our framework outperforms state-of-the-art baselines.

2 Preliminary

Let O be a set of n multi-modal data. Each object $o_i \in O$ has m modals and is represented as $o_i = \{o_i^1, \dots, o_i^m\}$, where o_i^j is o_i in the j -th metric space with high dimensionality. Let O^i be $\{o_1^i, \dots, o_n^i\}$. In our setting, the i -th metric space is represented by $(O^i, dist^i)$, where $dist^i$ is the distance function in this metric space. This distance function satisfies symmetry, non-negativity, identity, and triangle inequality.

2.1 Problem Definition

Let $\mathcal{D}$ and W be $\{dist^1, \dots, dist^m\}$ and $\{w_1, \dots, w_m\}$ ($\sum_m w_i = 1$), respectively. We first define multi-metric distance.

Definition 1 (Multi-metric distance). *The multi-metric distance between o_x and o_y is*

$$dist^W(o_x, o_y) = \sum_{dist^i \in \mathcal{D}} w_i \cdot \bar{dist}^i(o_x, o_y),$$

where $\bar{dist}^i(o_x, o_y) \in [0, 1]$ is the normalized $dist^i(o_x, o_y)$.

Now we are ready to define our problem.

Definition 2 (Multi-metric k nearest neighbor search). *Given O , a query object $q = \{q_i^1, \dots, q_i^m\}$, a result size k , and a weighting set W , a multi-metric k nearest neighbor search retrieves $R \subseteq O$ such that $|R| = k$ and $\forall o_x \in R$ and $\forall o_y \in O - R$, $\text{dist}^W(o_x, q) \leq \text{dist}^W(o_y, q)$. (Ties are broken arbitrarily.)*

As introduced in Section 1, it is practically hard to efficiently obtain the exact result when n is large and each modal has high dimensionality. This paper therefore considers an approximation approach, i.e., approximate k -NNS.

2.2 Baseline Algorithm

Before introducing a state-of-the-art baseline algorithm, we introduce notions for proximity graphs. Let G_i be a proximity graph in the i -th metric space, i.e., a proximity graph of O^i . A node set of G_i is hence O^i , and an edge set E^i has node (object) pairs. Essentially, there is an edge between o_x^i and o_y^i if they are similar w.r.t. dist^i . We use $E^i(o_x)$ to denote a set of edges $\in E^i$ held by o_x . Also, let $\mathcal{G} = \{G_1, \dots, G_m\}$.

Algorithm description. We here introduce a baseline algorithm that employs the separate approach and a proximity graph in each metric space. (This proximity graph is built offline.) This algorithm is implemented in the state-of-the-art vector management system Milvus [29] and Azure AI Search³. Its idea is fairly simple. For each metric space, it runs an approximate k -NNS with the greedy algorithm [26] and accumulates the result in this space. At last, it picks k objects with the minimum multi-metric distance in the accumulated result. Algorithm 1 details this algorithm.

2.3 Related Work

NNS in a metric space. As the problem of NNS can be defined based on an arbitrary distance function, many studies proposed efficient NNS algorithms that can be employed in any metric space [13, 14]. Recent experimental papers compare existing indexing techniques [15, 33]. Note that these works consider only low-dimensional data, so they are inappropriate for our problem.

NNS in multi-metric spaces. Existing works devised some data structures for k -NNS in multi-metric spaces to deal with multi-modal data. In [16], M2-tree was proposed, and this data structure combines all metrics. This integrated approach cannot consider W and is not robust against new metric insertions and existing metric deletions. To deal with any weights, [11, 12, 19] proposed data structures based on M-tree [17] and employ the separate approach. DESIRE [35] employs pivot-based distance indexing with B^+ -trees, different from M-tree. Again, these works essentially assume low dimensions.

As introduced in Section 2.2, Milvus [29] is a state-of-the-art vector management system and can deal with objects consisting of multiple vectors. Its

³ Milvus and Azure AI Search are not limited to proximity graphs for indices, but the essential approach is the same.

Algorithm 1: BASELINE

Input: $O, \mathcal{G}, q, k, W$, and $\epsilon \geq 1$

```

1  $R \leftarrow \emptyset$ 
2 foreach  $i \in [1, m]$  such that  $w_i > 0$  do
3    $o_s \leftarrow$  a random start node
4    $C, C' \leftarrow \langle o_s, \text{dist}^W(o_s, q) \rangle$ 
5    $F \leftarrow \{o_s\}$ 
6   while  $C' \neq \emptyset$  do
7      $o_x \leftarrow \text{pop } C'$ 
8     foreach  $o_y \in E_i(o_x)$  such that  $o_y \notin F$  do
9        $F \leftarrow F \cup \{o_y\}$ 
10       $\tau \leftarrow$  the  $\epsilon \cdot k$ -th distance in  $C$ 
11      if  $\text{dist}^W(o_y, q) < \tau$  then
12         $C \leftarrow C \cup \{\langle o_y, \text{dist}^W(o_y, q) \rangle\}$ 
13        Sort the tuples in  $C$  in ascending order of the second element
14        if  $|C| > \epsilon \cdot k_s$  then
15          Erase the last tuple in  $C$ 
16         $C' \leftarrow C' \cup \{\langle o_y, \text{dist}^W(o_y, q) \rangle\}$ 
17        Sort the tuples in  $C'$  in ascending order of the second element
18    $R \leftarrow R \cup C$ 
19   Sort the tuples in  $R$  in ascending order of the second element
20 return the first  $k$  objects in  $R$ 

```

framework for multi-metric k -NNS is to run a k -NNS for each vector space, which is the same as the separate approach. Milvus can employ any indexing technique to accelerate k -NNS in a given space, and Algorithm 1 is an example when Milvus uses a proximity graph. We use this algorithm as a competitor in Section 4. Another state-of-the-art algorithm for our problem is HJG [34]. Essentially, HJG is a HNSW [25] that considers all edges generated from all metric spaces. This algorithm is also used as a competitor in our experiments.

The above separate approach typically runs a k -NNS on each metric space, like Algorithm 1. Let T be the search time of this approach, and T is obtained by

$$T = \sum_{i=1}^m T_i + km, \quad (1)$$

where T_i is the k -NNS time in the i -th metric space. As m cannot be reduced, we need to reduce T_i to reduce T . The above works do not consider how to minimize T_i by using the intermediate results obtained in the other metric spaces. Our framework incorporates this idea.

Approximate NNS in high-dimensional spaces. Recent experimental studies [9, 24] demonstrate that proximity graphs yield the best trade-off between speed and recall. Therefore, many works developed proximity graphs [30]. Roughly, state-of-the-art proximity graphs are based on either NSW [26] or NSG [20].

NSW is an approximation of the Delaunay graph, whereas NSG is an approximation of a monotonic search graph (i.e., a graph with paths that monotonically reduce distances to a target object for any two objects).

Our framework allows arbitrary proximity graphs that can (i) use any node as a start node and (ii) be employed in any metric spaces. The first requirement is important to implement our idea for reducing search spaces. The second requirement is derived from our problem definition.

3 Proposed Framework

3.1 Main Ideas

The main bottleneck of k -NN is distance computation, so we need to minimize the number of distance computations to reduce the latency of multi-metric k -NNS. Simply employing a proximity graph for each metric space (like Algorithm 1) cannot achieve this well. Recall Fig. 2. The running time of proximity graph-based k -NNS depends on a given start node. If the start node is near a given query object q , its running time will be quite fast. On the other hand, if the start node is far from q , its performance degrades. It is hence important to select a start node to optimize the performance of (multi-metric) k -NNS. Nevertheless, this approach has not been considered in metric spaces.

Assume that we access the metric spaces in order of $O^1, O^2, \dots, O^m$, for ease of explanation. When we start to run an approximate k -NNS in O^i ($i \in [2, m]$), we can use the intermediate k -NNS result obtained in the metric spaces $O^1, \dots, O^{i-1}$. More specifically, we use the intermediate nearest neighbor node of q w.r.t. $dist^i$ as the start node. This idea brings the following merit:

There are correlations between multi-modal data. For example, similar images have similar captions (documents). This observation suggests that the intermediate nearest neighbor node of q in O^i is similar to q w.r.t. $dist^W$, so we need to traverse only nodes existing around q (and most of them may have been traversed in $O^1, \dots, O^{i-1}$).

Clearly, the above merit removes one of the drawbacks of Algorithm 1 (or of running k -NNS in each metric independently). The remaining concern is how to find a start node close to a given query q for O^1 because we have no intermediate k -NNS result for the first accessed metric space.

We overcome this concern by introducing *representative nodes* for the proximity graph in each metric space. Representative nodes are candidates for the start node. Although query objects are unknown in advance, start nodes should be close to a given query. To satisfy this, *representative nodes should be distributed uniformly in the metric space*, as shown in Fig. 3. We pick representative nodes based on this idea.

3.2 Overview

Our framework has two phases: the offline phase (indexing phase) and the k -NNS phase (query phase). The offline phase builds a proximity graph and selects

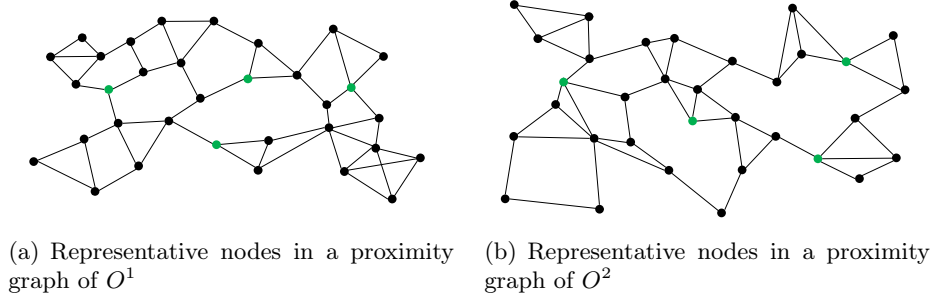


Fig. 3. Representative nodes (green nodes) in each metric space ($m = 2$)

representative nodes for each metric space. In a k -NNS phase, we first determine a start node and the access order of the metric spaces by using the data structure built in the offline phase, and then run the greedy algorithm [26] for each metric space.

3.3 Offline Phase

This phase is done only once if O does not change. (Dynamic data is not the scope of this paper.)

Building proximity graphs. First, we build a proximity graph G_i for each O^i ($i \in [1, m]$). As stated in Section 2.3, our framework can employ any proximity graph that can employ an arbitrary node as the start node and distance function. Therefore, the performance of our framework can be enhanced by a future state-of-the-art proximity graph that satisfies the above requirements.

Selecting representative nodes. To set an effective start node at the first accessed metric space, we select $r = \mathcal{O}(1)$ representative nodes in each metric space. To make them distributed uniformly in the metric space, we employ K -means++ algorithm [8] ($K = r$ in our setting) because it (i) yields useful cluster centers that tend to exist uniformly in a given data space, (ii) is robust against outliers, and (iii) runs in a reasonable time, i.e., $\mathcal{O}(Kn)$ time⁴.

Next, for each representative node o_i^j , we compute the maximum distance to its neighbor node in the proximity graph of O^j , i.e.,

$$mdist_i^j = \max_{o_l \in E^j(o_i)} \frac{dist^j(o_i, o_l) - mean^j}{std^j},$$

where $mean^j$ and std^j are the average distance and standard deviation of the distance in the j -th metric space⁵. That is, $mdist_i$ is z-normalized to enable

⁴ Similar results would be obtained by similar greedy algorithms [1, 2].

⁵ If the proximity graph has a so-called long-range edge, it is straightforward to ignore it.

Algorithm 2: DETERMINE-START-NODE

Input: q , W , and $\mathcal{P}$
Output: o_s (a start node) and I (a list of the access order of metric spaces)

- 1 $S, I \leftarrow \emptyset$
- 2 **foreach** $j \in [1, m]$ **do**
- 3 $S \leftarrow S \cup \{\langle j, o_i^j, w_j \cdot mdist_i^j \rangle\}$, where $o_i^j = \arg \min_{P^j} dist^j(o_i, q)$
- 4 Sort the triplets in S in ascending order of the last element
- 5 **foreach** $\langle \langle j, o_i^j, w_j \cdot mdist_i^j \rangle \in S$ **do**
- 6 $I \leftarrow I \cup \{j\}$ $\triangleright$ determine access order of metric spaces
- 7 $o_s \leftarrow S[0].o_i$ $\triangleright$ determine the start node

a fair comparison between the m metric spaces. (Without this, $dist^i$ may have a different scale to $dist^j$.) Our framework maintains $\mathcal{P} = \{P^1, \dots, P^m\}$, where $P^j = \{\langle o_i^j, mdist_i^j \rangle, \dots\}$ and $|P^j| = r$.

Time complexity. Let $\mathcal{T}(n)$ be the time to build a proximity graph. This phase needs $\mathcal{T}(n) \times m$ time to build m proximity graphs and $\mathcal{O}(rn) = \mathcal{O}(n) \times m$ time to obtain $\mathcal{P}$. Therefore, this phase needs $\mathcal{O}(\mathcal{T}(n) + nm)$ time. For example, when a proximity graph is KGraph [18], $\mathcal{T}(n) = \mathcal{O}(n)$ for a fixed degree [5, 6]. This offline processing, hence, can be done in time linear to n .

3.4 k -NNS Phase

Given q , k , and W , we first determine the access order of metric spaces and a start node. Algorithm 2 details this procedure. For each metric space and for each representative node o_i , we compute $dist^j(o_i, q)$ to obtain the node (object) closest to q among the representative nodes w.r.t. $dist^j$. We thus have at most m such nodes. We rank them based on $w_j \times mdist_i^j$ (smaller is better). The rank order corresponds to the access order of metric spaces. In addition, we use the first-ranked representative node as the start node.

After that, we use the greedy algorithm [26] on the proximity graph of each metric space. For the first metric space, we run it from the start node obtained above. When the search converges, we move to the proximity graph of the next metric space based on the determined access order. We set the intermediate nearest node to q as the start node and again run the greedy algorithm. Note that if an object o has been accessed in the former metric space, we do not traverse the node again. (That is, traversed nodes/objects are shared among the m metric spaces.) This procedure is also done for the subsequent metric spaces. Algorithm 3 details this phase.

Time complexity. Let θ be the average degree of the proximity graphs. Also, let h^i be the traversal hops in the i -th metric space. The time complexity of Algorithm 2 is $\mathcal{O}(m)$ since $r = \mathcal{O}(1)$. We here assume that the distance computation for each metric is done in $\mathcal{O}(1)$ time, so $dist^W$ is done in $\mathcal{O}(m)$ time. Then, the time complexity of Algorithm 3 is $\mathcal{O}(m) + \mathcal{O}(\sum_m \theta \cdot h^i \cdot m) = \mathcal{O}(\sum_m \theta \cdot h^i \cdot m)$.

Algorithm 3: Our Framework

Input: $O, \mathcal{G}, q, k, W$, and ϵ

```

1  $(o_s, I) \leftarrow \text{DETERMINE-START-NODE}$ 
2  $F \leftarrow \{o_s\}$ 
3  $C \leftarrow \emptyset$ 
4 foreach  $i \in I$  such that  $w_i > 0$  do
5    $C' \leftarrow \emptyset$ 
6   if  $C = \emptyset$  then
7      $C, C' \leftarrow \{\langle o_s, \text{dist}^W(o_s, q) \rangle\}$ 
8   else
9      $o_s \leftarrow \arg \min_{o \in C} \text{dist}^i(o^i, q^i)$ 
10     $C' \leftarrow C \cup \{\langle o_s, \text{dist}^W(o_s, q) \rangle\}$ 
11  while  $C' \neq \emptyset$  do
12     $o_x \leftarrow \text{pop } C'$ 
13    foreach  $o_y \in E^i(o_x)$  s.t.  $o_y \notin F$  do
14       $F \leftarrow F \cup \{o_y\}$ 
15       $\tau \leftarrow$  the  $\epsilon \cdot k$ -th distance in  $C$ 
16      if  $\text{dist}^W(o_y, q) < \tau$  then
17         $C \leftarrow C \cup \{\langle o_y, \text{dist}^W(o_y, q) \rangle\}, C' \leftarrow C' \cup \{\langle o_y, \text{dist}^W(o_y, q) \rangle\}$ 
18        Sort the tuples in  $C$  and  $C'$  in ascending order of the second
        element
19        if  $|C| > \epsilon \cdot k$  then
20          Erase the last tuple in  $C$ 
21 return the first  $k$  objects in  $C$ 

```

Notice that θ and m are regarded as fixed parameters because they are system- or data-dependent parameters. Since our framework aims at reducing h^i , Algorithm 3 practically yields a small h^i . This is confirmed in the next section.

4 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.

4.1 Setting

Dataset. We used two real-world multi-metric datasets: Corel Image Features⁶ and NUS-WIDE⁷. Table 1 shows their statistics. Corel Image Features consists of Color Histogram (CH), Layout Histogram (LH), Color Moments (CM),

⁶ <https://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html>

⁷ <https://lms.comp.nus.edu.sg/wp-content/uploads/2019/research/nuswide/NUSWIDE.html>

Table 1. Statistics of dataset

Dataset	#objects	Dimensionality	#modals	Distance function
Corel Image Features	66,616	9–32	4	L_1 norm and L_2 norm
NUS-WIDE	269,648	64–228	5	L_1 norm and L_2 norm

and COOC Texture (CT). We used L_1 norm for CH and LH, whereas we used L_2 norm for the other modals. NUS-WIDE consists of CH, Color Correlogram (CORR), Edge Direction Histogram (EDH), Wavelet Texture (WT), and block-wise Color Moments extracted over 5x5 fixed grid partitions (CM55). We used L_1 norm for CH, CORR and EDH, whereas we used L_2 norm for the other modals. For each dataset, we picked random 1,000 objects and used them as queries. For each query, each $w_i \in W$ was randomly determined.

Evaluated algorithms. We evaluated our framework, denoted by **Ours**⁸, Algorithm 1, denoted by **Baseline**, HJG⁹ [34], and **Linear-scan**, which simply scans O and computes $dist^W(o, q)$ for each $o \in O$ and returns the exact k -NNS result. These algorithms were single-threaded, implemented in C++, and compiled by g++ 11.4.0 with -O3 optimization. We enabled SIMD instruction for distance computations. As for **Ours**, we set $r = 1,000$.

Proximity graph. For each metric space, we built an undirected approximate K -NN graph [7], where $K = 20$, because this proximity graph satisfies the requirements of our framework¹⁰. We used NNDESCENT [18] to build a directed approximate K -NN graph and then made it undirected.

Evaluation criteria. We measured the average running time vs. average recall when issuing 1,000 queries. Let R' be a set of k objects returned by our framework or Algorithm 1. Its recall is obtained by

$$Recall(R') = \frac{|R \cap R'|}{k}.$$

4.2 Result

As for **Ours** and **Baseline**, we obtained time-recall curves by varying ϵ (e.g., see Algorithm 3), which are shown in Figs. 4 and 5. Recall that **Linear-scan** returns the exact result, and its search time is shown in Table 2. Note that HJG is slower than **Linear-scan** on Corel Image Features although HJG cannot provide the exact result, which is also shown in Table 2. In addition, HJG did not run on NUS-WIDE because of the segmentation fault error (so Table 2 shows “-” in the corresponding cells.) Therefore, we hereinafter omit the results of HJG.

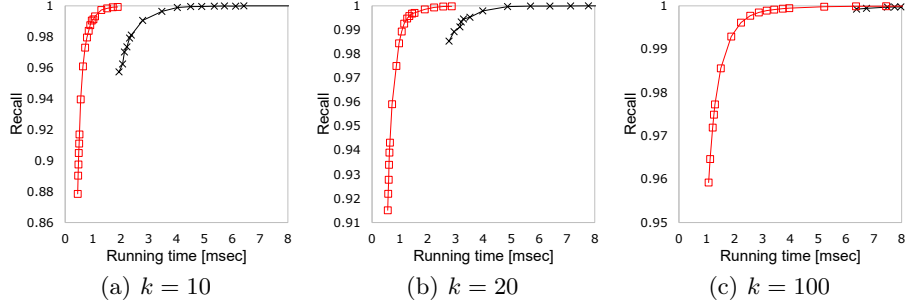
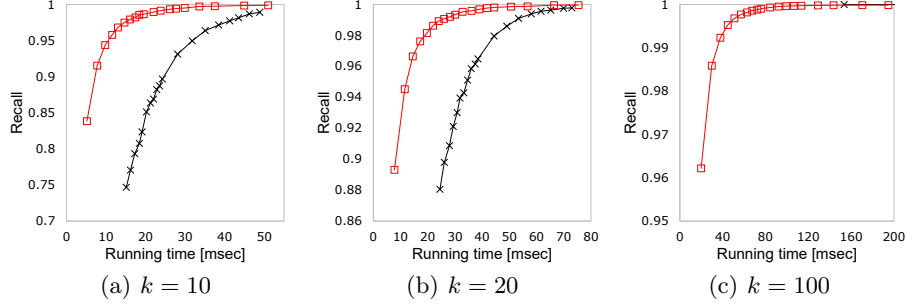
⁸ <https://github.com/leonuemura/Multimetric-ANNS-Framework>

⁹ <https://github.com/ZJU-DAILY/HJG>

¹⁰ Famous proximity graphs HNSW [25], NSG [20], and Vamana [22] fix the start node, so they cannot be employed.

Table 2. Search time and recall of Linear-scan and HJG ($k = 10$)

Dataset	Running time [msec]		Recall	
	Linear-scan	HJG	Linear-scan	HJG
Corel Image Features	32.28	42.83	1	0.97
NUS-WIDE	227.69	-	1	-

**Fig. 4.** Result on Corel Image Features. “ $\times$ ” shows Baseline and “ $\square$ ” shows Ours.**Fig. 5.** Result on NUS-WIDE. “ $\times$ ” shows Baseline and “ $\square$ ” shows Ours.

Result on Corel Image Features. Fig. 4 shows the time-recall curves on Corel Image Features with $k = 10$, $k = 20$, and $k = 100$. We see that Ours yields a better time-recall trade-off, i.e., Ours outperforms Baseline. To better understand this result, we investigated the decomposed times of Ours and Baseline under the same recall (0.97). Table 3 shows the decomposed times. As Baseline runs a k -NNS for each metric space independently, it took a similar running time for each metric space. On the other hand, Ours exploits the intermediate k -NN result to determine a start node, so it took *only a few dozen microseconds* for each $\mathcal{M}_i$ ($i \in [2, m]$), where $\mathcal{M}_i$ represents the i -th accessed metric space. (Note that Ours and Baseline *do not* share the same access order of metric spaces, i.e., $\mathcal{M}_i$ of Ours and $\mathcal{M}_i$ of Baseline can be different.) Ours thus is faster than Baseline

Table 3. Average running time [msec] under the same recall (Corel Image Features and $k = 10$). $\mathcal{M}_i$ represents the i -the accessed metric space.

	Algorithm 2	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	Total	Recall
Baseline	–	0.54	0.62	0.44	0.28	2.10	0.97
Ours	0.22	0.43	0.02	0.01	0.01	0.70	0.97
Ours by a random start node	–	0.96	0.02	0.01	0.01	1.01	0.97

Table 4. Average running time [msec] under the same recall (NUS-WIDE and $k = 10$). $\mathcal{M}_i$ represents the i -the accessed metric space.

	Algorithm 2	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$	Total	Recall
Baseline	–	3.16	4.44	4.74	3.17	7.54	24.33	0.90
Ours	0.55	6.03	0.19	0.12	0.08	0.08	7.09	0.90
Ours by a random start node	–	9.85	0.21	0.16	0.09	0.08	10.38	0.90

under the same recall on Corel Image Features. This result confirms the efficacy of our first main idea described in Section 3.1.

In addition, comparing the results in Fig. 4 and Table 2, **Ours** returns almost perfect k -NN result much faster than **Linear-scan**. For example, when $k = 10$, **Ours** returns 0.96 recall in 0.64 [msec], which is 50 times faster than **Linear-scan**.

Result on NUS-WIDE. Fig. 4 shows the time-recall curves on NUS-WIDE with $k = 10$, $k = 20$, and $k = 100$. The results are essentially similar to those on Corel Image Features. **Ours** always outperforms **Baseline**. Furthermore, **Ours** obtains a reasonably high recall in 10 microseconds, which is much faster than **Linear-scan**.

Efficacy of our start node selection. Last, we investigate the efficacy of representative nodes. To show this, we compare **Ours** with its variant that used a random node as the start node. Tables 3 and 4 show the results. We need to focus on the running time on $\mathcal{M}_1$, and there is an obvious improvement, which yields about 30% total time improvement under the same recall. This clarifies that our second idea described in Section 3.1 also functions well in practice.

5 Conclusion

Due to the proliferation of multi-modal data and embedding techniques, objects often contain multiple high-dimensional data and the importance of multi-metric search has been increasing. This paper hence addressed the multi-metric k nearest neighbor search problem in high dimensions. The state-of-the-art algorithm for this problem does not consider the correlation between multi-metric spaces, so it cannot exploit the search results of the other metric spaces. We overcame this inefficiency issue by proposing a new framework for this problem. Our framework can use any proximity graph that can employ an arbitrary start node and

be used in any metric space. Our analysis of the time complexity of our framework suggests that our framework functions better than the baseline algorithm (under correlated multi-metric spaces). We conducted experiments by using real-world datasets, and the results demonstrate that our framework outperforms the baseline algorithm.

Acknowledgements. This work was partially supported by AIP Acceleration Research JPMJCR23U2, JST, and JSPS KAKENHI Grant Number 24K14961.

References

1. Amagata, D.: Diversity maximization in the presence of outliers. In: AAAI. pp. 12338–12345 (2023)
2. Amagata, D.: Fair k-center clustering with outliers. In: AISTATS. pp. 10–18 (2024)
3. Amagata, D., Arai, Y., Fujita, S., Hara, T.: Learned k-nn distance estimation. In: SIGSPATIAL. pp. 1–4 (2022)
4. Amagata, D., Hara, T., Xiao, C.: Dynamic set knn self-join. In: ICDE. pp. 818–829 (2019)
5. Amagata, D., Onizuka, M., Hara, T.: Fast and exact outlier detection in metric spaces: a proximity graph-based approach. In: SIGMOD. pp. 36–48 (2021)
6. Amagata, D., Onizuka, M., Hara, T.: Fast, exact, and parallel-friendly outlier detection algorithms with proximity graph in metric spaces. *The VLDB Journal* **31**(4), 797–821 (2022)
7. Arai, Y., Amagata, D., Fujita, S., Hara, T.: Lgtm: A fast and accurate knn search algorithm in high-dimensional spaces. In: DEXA. pp. 220–231 (2021)
8. Arthur, D., Vassilvitskii, S.: K-means++ the advantages of careful seeding. In: SODA. pp. 1027–1035 (2007)
9. Aumüller, M., Bernhardsson, E., Faithfull, A.: Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In: SISAP. pp. 34–49 (2017)
10. Bustos, B., Keim, D., Schreck, T.: A pivot-based index structure for combination of feature vectors. In: SAC. pp. 1180–1184 (2005)
11. Bustos, B., Kreft, S., Skopal, T.: Adapting metric indexes for searching in multi-metric spaces. *Multimedia Tools and Applications* **58**(3), 467–496 (2012)
12. Bustos, B., Skopal, T.: Dynamic similarity search in multi-metric spaces. In: MIR. pp. 137–146 (2006)
13. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. *ACM Computing Surveys* **33**(3), 273–321 (2001)
14. Chen, L., Gao, Y., Song, X., Li, Z., Zhu, Y., Miao, X., Jensen, C.S.: Indexing metric spaces for exact similarity search. *ACM Computing Surveys* **55**(6), 1–39 (2022)
15. Chen, L., Gao, Y., Zheng, B., Jensen, C.S., Yang, H., Yang, K.: Pivot-based metric indexing. *Proceedings of the VLDB Endowment* **10**(10), 1058–1069 (2017)
16. Ciaccia, P., Patella, M.: The m2-tree: Processing complex multi-feature queries with just one index. In: DELOS (2000)
17. Ciaccia, P., Patella, M., Zezula, P., et al.: M-tree: An efficient access method for similarity search in metric spaces. In: VLDB. pp. 426–435 (1997)
18. Dong, W., Moses, C., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In: WWW. pp. 577–586 (2011)
19. Franzke, M., Emrich, T., Züfle, A., Renz, M.: Indexing multi-metric data. In: ICDE. pp. 1122–1133 (2016)

20. Fu, C., Xiang, C., Wang, C., Cai, D.: Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment* **12**(5), 461–474 (2019)
21. Harwood, B., Drummond, T.: Fanng: Fast approximate nearest neighbour graphs. In: *CVPR*. pp. 5713–5722 (2016)
22. Jayaram Subramanya, S., Devvrit, F., Simhadri, H.V., Krishnawamy, R., Kadekodi, R.: Diskann: Fast accurate billion-point nearest neighbor search on a single node. *NeurIPS* **32** (2019)
23. Jo, S., Trummer, I.: Demonstration of thalamusdb: Answering complex sql queries with natural language predicates on multi-modal data. In: *SIGMOD*. pp. 179–182 (2023)
24. Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., Lin, X.: Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* **32**(8), 1475–1488 (2020)
25. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **42**(4), 824–836 (2018)
26. Malkov, Y., Ponomarenko, A., Logvinov, A., Krylov, V.: Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* **45**, 61–68 (2014)
27. Patroumpas, K., Zeakis, A., Skoutas, D., Santoro, R.: Multi-attribute similarity search for interactive data exploration. In: *EDBT/ICDE Workshop* (2021)
28. Tian, Y., Zhao, X., Zhou, X.: Db-lsh: Locality-sensitive hashing with query-based dynamic bucketing. In: *ICDE*. pp. 2250–2262 (2022)
29. Wang, J., Yi, X., Guo, R., Jin, H., Xu, P., Li, S., Wang, X., Guo, X., Li, C., Xu, X., et al.: Milvus: A purpose-built vector data management system. In: *SIGMOD*. pp. 2614–2627 (2021)
30. Wang, M., Xu, X., Yue, Q., Wang, Y.: A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment* **14**(11), 1964–1978 (2021)
31. Wang, R., Deng, D.: Deltapq: lossless product quantization code compression for high dimensional similarity search. *Proceedings of the VLDB Endowment* **13**(13), 3603–3616 (2020)
32. Yu, J., Wang, Z., Vasudevan, V., Yeung, L., Seyedhosseini, M., Wu, Y.: Coca: Contrastive captioners are image-text foundation models. *arXiv preprint arXiv:2205.01917* (2022)
33. Zhu, Y., Chen, L., Gao, Y., Jensen, C.S.: Pivot selection algorithms in metric spaces: a survey and experimental study. *The VLDB Journal* **31**(1), 23–47 (2022)
34. Zhu, Y., Chen, L., Gao, Y., Ma, R., Zheng, B., Zhao, J.: H_{jg}: An effective hierarchical joint graph for approximate nearest neighbour search in multi-metric spaces. In: *ICDE*. pp. 4275–4287 (2024)
35. Zhu, Y., Chen, L., Gao, Y., Zheng, B., Wang, P.: Desire: An efficient dynamic cluster-based forest indexing for similarity search in multi-metric spaces. *Proceedings of the VLDB Endowment* **15**(10), 2121–2133 (2022)