Accelerating Graph Construction for MIPS without Search Accuracy Loss

Yasuhiro Fujiwara
NTT, Inc.
Kanagawa, Japan
yasuhiro.fujiwara@ntt.com

Ángel López García-Arias
NTT, Inc.
Kanagawa, Japan
angel.lopez@ntt.com

Yu Mitsuzumi
NTT, Inc.
Kanagawa, Japan
yu.mitsuzumi@ntt.com

Yasutoshi Ida
NTT, Inc.
Tokyo, Japan
yasutoshi.ida@ntt.com

Atsutoshi Kumagai
NTT, Inc.
Tokyo, Japan
atsutoshi.kumagai@ntt.com

Masahiro Nakano
NTT, Inc.
Kanagawa, Japan
ma.nakano@ntt.com

Makoto Nakatsuji
NTT, Inc.
Kanagawa, Japan
makoto.nakatsuji@ntt.com

Akisato Kimura
NTT, Inc.
Kanagawa, Japan
akisato@ieee.org

Abstract

Maximum Inner Product Search (MIPS) is a fundamental task to identify the vector yielding the highest inner product with a given query. Since MIPS is the computational bottleneck in many applications, various approximation approaches have been proposed. Among them, ip-NSW and its variants are popular graph-based approaches that achieve high search accuracy in two steps. The graph construction step inserts vectors one by one into an empty graph by connecting to nodes with high inner products to build a proximity graph. The search step recursively explores the graph by selecting linked nodes to identify nodes of high inner products to the query. Although the search step can be processed efficiently, the graph construction step unfortunately incurs a high computational cost due to the large number of inner product computations required. This paper proposes a general approach to accelerate the graph construction step of ip-NSW and variants. We improve efficiency by skipping unnecessary inner product computations using the upper bounds of inner products. Importantly, our approach is guaranteed to construct exactly the same graphs as those produced by ip-NSW and its variants, thereby maintaining the same search accuracy. This indicates that our approach is a friend to ip-NSW and its variants rather than a competitor. Experimental results show that our approach reduces the graph construction time without sacrificing the search accuracy of ip-NSW and its variants.

Keywords

Efficient; Algorithm, Graph construction

1 Introduction

The rapid growth of unstructured data, such as images, text, and audio, has triggered a need for vector database management systems to process high-dimensional vectors efficiently [12–14, 23, 29, 37, 38, 46, 51]. In the literature, many vector database management systems have been proposed, such as Milvus [42], Manu [15], AnalyticDB-V [44], PASE [47], and Faiss [21]. Maximum Inner Product Search (MIPS) is a core task of vector database management used in knowledge-grounded dialog [28], recommender systems [50], and scene understanding [4]. Given a set of n vectors $\mathbf{x}_i = [x_i[1], \dots, x_i[d]]$ $(i = 1, 2, \dots, n)$ in a d-dimensional space, MIPS identifies the vector $\mathbf{x}^*$ that maximizes the inner product with a given query vector $\mathbf{q} = [q[1], \dots, q[d]]$. Formally, for set $\mathbb{X}$ of n vectors, MIPS solves

$$\mathbf{x}^* = \text{argmax}_{\mathbf{x}_i \in \mathbb{X}}\, p(\mathbf{q}, \mathbf{x}_i). \tag{1}$$

In this equation, $p(\mathbf{q}, \mathbf{x}_i) = \|\mathbf{q}\|\|\mathbf{x}_i\| \cos(\varphi_{\mathbf{q},\mathbf{x}_i}) = \mathbf{q}\mathbf{x}_i^\top$ is the inner product between vector $\mathbf{q}$ and $\mathbf{x}_i$ where $\|\cdot\|$ is l_2-norm of a vector and $\varphi_{\mathbf{q},\mathbf{x}_i}$ is the angle between $\mathbf{q}$ and $\mathbf{x}_i$. A naive approach computes the inner product between the query and each vector, but this is computationally infeasible for large-scale, high-dimensional data.

To address this challenge, a variety of approximate MIPS approaches have been developed. Quantization-based approaches accelerate inner product computations by encoding vectors into compact sequences of codewords [8, 16, 49]. Clustering-based and tree-based approaches partition the high-dimensional space using clusters or tree structures, respectively [3, 7, 10, 24, 35]. Hashing-based approaches divide the space into subregions using hash functions [31, 36, 45]. Graph-based approaches, such as ip-NSW and its variants [30, 39, 40, 54], are particularly effective due to their ability to reduce the number of candidate vectors while maintaining high accuracy. These graph-based approaches construct proximity graphs where each node represents a vector, and edges link to nodes with high inner products. To approximately perform MIPS, these approaches employ two key steps: (1) The graph construction step incrementally builds the graph by linking a node to its similar nodes based on inner products. (2) The search step performs a greedy search over the graph for a given query to identify similar nodes. Theoretically, these approaches approximate the s-Delaunay graph, known to yield the optimal solutions for MIPS [30]. As a result, the graph-based approaches achieve high accuracy in MIPS.

However, ip-NSW and its variants have high computational costs in the graph construction step, as noted in prior works [7, 10, 53]. For instance, [7] describes this problem as "the poor scalability and slow index construction of graph methods." The graph construction step iteratively inserts vectors one by one into an initially empty graph. For each insertion, it performs a greedy walk to find top-k neighbors based on inner products.

EDBT '26, Tampere (Finland)

10.48786/edbt.2026.24

This involves repeatedly computing inner products between the inserting vectors and nodes in the graph. After it obtains the top-k nodes, it further computes inner products between the top-k nodes to select nodes connected to the inserting vector. These processes result in a large number of inner product computations, making the graph construction step expensive. While ip-NSW and its variants offer high search efficiency and accuracy after the graph construction step, the cost of building the graph remains a bottleneck, limiting their practicality in many applications.

This paper proposes a general approach that reduces graph construction time for ip-NSW and its variants to mitigate this problem, while ensuring that the resulting graphs are identical to those produced by the original approaches. Our key idea is to compute upper bounds of inner products by approximating vectors, allowing us to avoid unnecessary computations. Specifically, we use the upper bounds in two ways to ensure correctness: (1) our approach identifies and skips nodes that cannot lie on the greedy walk path, and (2) the proposed approach prunes top-k nodes that cannot be connected to the inserting vector. By reducing the number of inner product computations, we can improve the efficiency of graph construction. Since our approach produces exactly the same graph structure as ip-NSW and its variants, it does not compromise search accuracy for faster construction. In addition, the proposed approach is also applicable to reduce search time, even though our approach was primarily designed to accelerate the graph construction step. This is because the graph construction and search steps rely on the same algorithm for identifying top-k nodes. It is important to emphasize that our approach is not a competitor to ip-NSW and its variants; it is a *friend* method that improves the efficiency of ip-NSW and its variants while retaining their effectiveness. In summary, our main contributions are as follows:

- We propose a method to compute upper-bounding inner products to prune unnecessary computations in not only the graph construction step but the search step.
- The proposed approach is guaranteed to produce exactly the same graphs as ip-NSW and its variants.
- Experiments demonstrate that our approach can reduce graph construction time without degrading the search accuracy of ip-NSW and its variants.

The remainder of this paper is organized as follows: Section 2 overviews the background. Section 3 introduces our approach. Section 4 describes related work. Section 5 presents the experiments performed. Section 6 concludes the paper.

2 Preliminaries

We provide a brief overview of ip-NSW and its variants. Although they employ multiple hierarchical layers to enhance performance, we follow previous works [30, 39, 54] and assume a single-layer graph for a simple explanation. Table 1 lists the main symbols.

MIPS is different from the l_2-distance-based Approximate Nearest Neighbor Search (ANNS) since the inner product is a non-metric measure. For example, l_2-distance $D(\mathbf{x}_i, \mathbf{x}_j)$ is always positive if $\mathbf{x}_i \neq \mathbf{x}_j$, i.e., $D(\mathbf{x}_i, \mathbf{x}_j) > 0$. On the other hand, even if $\mathbf{x}_i \neq \mathbf{x}_j$, the inner product could be negative. This is because $p(\mathbf{q}, \mathbf{x}_i) = \|\mathbf{q}\|\|\mathbf{x}_i\| \cos(\varphi_{\mathbf{q},\mathbf{x}_i})$ and we could have $\cos(\varphi_{\mathbf{q},\mathbf{x}_i}) < 0$. In addition, $D(\mathbf{x}_i, \mathbf{x}_j)$ satisfies the identity of indiscernibles: $D(\mathbf{x}_i, \mathbf{x}_j) = 0 \Leftrightarrow \mathbf{x}_i = \mathbf{x}_j$. However, $p(\mathbf{x}_i, \mathbf{x}_j) = 0$ can hold even if $\mathbf{x}_i \neq \mathbf{x}_j$ since we could have $p(\mathbf{q}, \mathbf{x}_i) = \|\mathbf{q}\|\|\mathbf{x}_i\| \cos(\varphi_{\mathbf{q},\mathbf{x}_i}) = 0$ if $\cos(\varphi_{\mathbf{q},\mathbf{x}_i}) = 0$. Moreover, the inner product does not satisfy the triangle inequality [11], i.e., we could have $p(\mathbf{x}_i, \mathbf{x}_j) \nleq p(\mathbf{x}_i, \mathbf{x}_k) + p(\mathbf{x}_j, \mathbf{x}_k)$, while $D(\mathbf{x}_i, \mathbf{x}_j) \leq D(\mathbf{x}_i, \mathbf{x}_k) + D(\mathbf{x}_j, \mathbf{x}_k)$ holds for l_2-distance. Consequently, ANNS techniques designed for metric spaces perform poorly when applied to MIPS [2, 40].

Table 1: Definitions of main symbols.

Symbol	Definition
n	Number of vectors
d	Number of dimensions
k	Number of top-ranked vectors
m	Number of edges connected to each node
$p(\mathbf{q}, \mathbf{x}_i)$	Inner product of $\mathbf{q}$ and $\mathbf{x}_i$
$\overline{p}(\mathbf{q}, \mathbf{x}_i)$	Upper bound of $p(\mathbf{q}, \mathbf{x}_i)$
$\varphi_{\mathbf{q},\mathbf{x}_i}$	Angle between $\mathbf{q}$ and $\mathbf{x}_i$
$\|\mathbf{x}_i\|$	l_2-norm of vector $\mathbf{x}_i$
$\mathbf{x}_i$	i-th vector
$\mathbf{q}$	Query vector
$\tilde{\mathbf{x}}_i$	Approximated vector of $\mathbf{x}_i$
$\Delta\mathbf{x}_i$	Error vector of $\mathbf{x}_i$
$\mathbb{X}$	Set of n vectors
$\mathbb{G}$	Proximity graph
$\mathbb{K}$	Set of top-k nodes
$\mathbb{C}$	Set of connected nodes

ip-NSW is a widely used graph-based approach for MIPS, consisting of the graph construction step and the search step [30]. While the search step is highly efficient, the graph construction step is computationally expensive [7, 10, 53]. Algorithms 1, 2, and 3 are the procedures used in the graph construction step. In these algorithms, $\mathbb{G}$ denotes the constructed proximity graph, $\mathbb{K}$ represents the set of top-k nodes for vector $\mathbf{x}$, and $\mathbb{C}$ denotes the set of nodes connected to $\mathbf{x}$. As shown in Algorithm 1, the graph construction step proceeds by inserting vectors into an empty graph iteratively. For each inserting vector $\mathbf{x}$, it first identifies its top-k neighbors through a greedy traversal of the existing graph starting from node $\mathbf{s}$, as shown in Algorithm 2. In Algorithm 2, a dummy node has an inner product of $-\infty$ with $\mathbf{x}$. This traversal iteratively computes the inner products of nodes directly connected to the top-k nodes, navigating toward nodes with higher inner products. Note that, if $\mathbb{G} = \emptyset$, node $\mathbf{s}$ is set as a dummy node. Next, the graph construction step determines which top-k nodes should be connected to $\mathbf{x}$ by computing their inner products, as shown in Algorithm 3.

Theoretically, the graph constructed by ip-NSW approximates s-Delaunay graph, which yields the optimal solutions for MIPS [30]. As a result, ip-NSW achieves high search accuracy, making it a useful approach for MIPS. However, the graph construction step has a high computational cost due to the large number of inner product computations. Note that Algorithm 2 is also used in the search step to find top-k nodes.

Several variants of ip-NSW have been proposed to improve search performance. Tan et al. introduced IPDG to approximate s-Delaunay graph effectively [40]. The key distinction between IPDG and ip-NSW lies in the edge selection strategy. ip-NSW determines whether to connect node $\mathbf{u}$ based on the condition $p(\mathbf{u}, \mathbf{v}) > p(\mathbf{x}, \mathbf{u})$, as shown in line 7 of Algorithm 3. IPDG instead applies the following criterion to prune candidate neighbors during graph construction in line 7 of Algorithm 3:

$$p(\mathbf{u}, \mathbf{v}) > p(\mathbf{v}, \mathbf{v}). \qquad (2)$$

NAPG is another variant of ip-NSW that adjusts vector norms during edge selection to enhance search accuracy [39]. It modifies

Algorithm 1 Graph Construction

```
Input: vectors $\mathbb{X}$, number of top-$k$ nodes $k$, and number of edges $m$
Output: proximity graph $\mathbb{G}$
1: initialize $\mathbb{G}$ as $\emptyset$;
2: for each $\mathbf{x} \in \mathbb{X}$ do
3:     compute top-$k$ nodes $\mathbb{K}$ in $\mathbb{G}$ by Algorithm 2;
4:     insert $\mathbf{x}$ to $\mathbb{G}$ as a node;
5:     add $m$ edges of $\mathbf{x}$ to $\mathbb{G}$ obtained by Algorithm 3;
6: return $\mathbb{G}$;
```

Algorithm 2 Top-k Search

```
Input: vector $\mathbf{x}$, number of top-$k$ nodes $k$, and proximity graph $\mathbb{G}$
Output: top-$k$ nodes $\mathbb{K}$
1: randomly select node $\mathbf{s}$ from the graph;
2: $\mathbb{K} \leftarrow \{\mathbf{s}\}$;
3: mark $\mathbf{s}$ as checked and the rest nodes as unchecked;
4: add $k-1$ dummy nodes to $\mathbb{K}$;
5: repeat
6:     for each unchecked $\mathbf{u}$ connected to $\mathbb{K}$ do
7:         $\mathbf{v} \leftarrow \mathrm{argmin}_{\mathbf{v}' \in \mathbb{K}}\, p(\mathbf{x}, \mathbf{v}')$;
8:         compute $p(\mathbf{x}, \mathbf{u})$;
9:         if $p(\mathbf{x}, \mathbf{u}) > p(\mathbf{x}, \mathbf{v})$ then
10:            add $\mathbf{u}$ to $\mathbb{K}$;
11:            subtract $\mathbf{v}$ from $\mathbb{K}$;
12:        mark $\mathbf{u}$ as checked;
13: until $\mathbb{K}$ converges
14: return $\mathbb{K}$;
```

the inequality used in line 7 of Algorithm 3 to:

$$p(\mathbf{u}, \mathbf{v}) > \alpha p(\mathbf{x}, \mathbf{u}), \tag{3}$$

where α is a scaling parameter typically set to $\alpha > 1$, making the scales of $p(\mathbf{u}, \mathbf{v})$ and $\alpha p(\mathbf{x}, \mathbf{u})$ comparable for effective MIPS. Möbius-Graph proposed by Zhou et al. reduces search time by transforming vectors before graph construction [54]. After applying Möbius transformation to each vector $\mathbf{x}$ as $\mathbf{x} = \mathbf{x}/\|\mathbf{x}\|^2$ and adding a zero vector, it constructs the graph using the same algorithm as ip-NSW, but applied to the transformed vectors.

Although ip-NSW and its variants achieve high search performance after the graph construction step, they suffer from high computational costs during the graph construction step. This inefficiency stems from the top-k search procedure of Algorithm 2 and the edge selection process of Algorithm 3, which involve a large number of inner product computations. As shown in Algorithm 2, the top-k search iteratively computes an inner product of each neighboring node to top-k nodes in $O(d)$ time, and it needs $O(\log k)$ time to identify node $\mathbf{v}$ having the lowest inner product. Therefore, letting l be the average number of traversed nodes, the total time complexity for computing top-k nodes becomes $O(nl(d + \log k))$. In selecting edges, each inner product between top-k nodes requires $O(d)$ time, and sorting the top-k nodes takes $O(k \log k)$ time. As a result, since the number of inner production computations is $O(mk)$ in Algorithm 3, the total time for edge selection is $O(nk(md + \log k))$. Consequently, the overall time complexity of the graph construction step is $O(nl(d + \log k))$, which results in high computational overhead, particularly for large-scale and high-dimensional datasets, as discussed in the previous studies [7, 10, 53]. Note that, since the search step exploits Algorithm 2 to find top-k nodes, it takes $O(l(d + \log k))$ time; the search step is more efficient than the graph construction step. In terms of memory usage, storing vectors, edges, and top-k nodes

Algorithm 3 Edge Selection

```
Input: vector $\mathbf{x}$, set of top-$k$ nodes $\mathbb{K}$, and number of edges $m$
Output: connected nodes $\mathbb{C}$
1: initialize $\mathbb{C}$ as $\emptyset$;
2: sort nodes in $\mathbb{K}$ in descending order of inner products with $\mathbf{x}$;
3: for each node $\mathbf{u}$ in sorted set $\mathbb{K}$ do
4:     flag $\leftarrow$ True;
5:     for each $\mathbf{v} \in \mathbb{C}$ do
6:         compute $p(\mathbf{u}, \mathbf{v})$;
7:         if $p(\mathbf{u}, \mathbf{v}) > p(\mathbf{x}, \mathbf{u})$ then
8:             flag $\leftarrow$ False;
9:             break;
10:    if flag = True then
11:        add $\mathbf{u}$ to $\mathbb{C}$;
12:    if $|\mathbb{C}| = m$ then
13:        break;
14: return $\mathbb{C}$;
```

requires $O(nd)$, $O(nm)$, and $O(k)$ space, respectively. Thus, the total memory cost is $O(n(m + d))$.

3 Proposed Method

This section presents the proposed approach. While we primarily focus on improving the efficiency of the graph construction step in ip-NSW, the proposed approach is also applicable to its variants. As discussed in the previous section, the graph construction step incurs a high computational cost due to the large number of inner product computations. To address this problem, the proposed approach prunes unnecessary inner product computations in the graph construction step. This section is organized as follows. We first introduce the key idea of using upper bounds, discuss the associated technical challenges, and present our solution in Sections 3.1, 3.2, and 3.3, respectively. We then describe approaches for tightening the upper bounds in Sections 3.4 and 3.5. Finally, we present the graph construction algorithm in Section 3.6 and discuss its extensions in Section 3.7.

3.1 Key Idea: Upper Bound-Based Pruning

The proposed approach is motivated by the observation that the graph construction step computes inner products to verify whether they exceed certain thresholds. For instance, Algorithm 2 computes inner product $p(\mathbf{x}, \mathbf{u})$ to evaluate the condition $p(\mathbf{x}, \mathbf{u}) > p(\mathbf{x}, \mathbf{v})$ when determining the top-k nodes (lines 8-9). Similarly, Algorithm 3 computes inner product $p(\mathbf{u}, \mathbf{v})$ to check whether $p(\mathbf{u}, \mathbf{v}) > p(\mathbf{x}, \mathbf{u})$ during edge selection (lines 6-7). To improve efficiency, we propose to evaluate these conditions using efficiently computable upper bounds of inner products. Let $\overline{p}(\mathbf{x}, \mathbf{u})$ be the upper bound of $p(\mathbf{x}, \mathbf{u})$. If $\overline{p}(\mathbf{x}, \mathbf{u}) \le p(\mathbf{x}, \mathbf{v})$, then the condition $p(\mathbf{x}, \mathbf{u}) > p(\mathbf{x}, \mathbf{v})$ cannot be satisfied, and we can safely skip the exact computation of $p(\mathbf{x}, \mathbf{u})$. Thus, exact inner product computation is only necessary when $\overline{p}(\mathbf{x}, \mathbf{u}) > p(\mathbf{x}, \mathbf{v})$. Similarly, during edge selection, the computation of $p(\mathbf{u}, \mathbf{v})$ can be omitted if $\overline{p}(\mathbf{u}, \mathbf{v}) \le p(\mathbf{x}, \mathbf{u})$. By estimating the upper bounds, the proposed approach reduces the number of exact inner product computations in the graph construction step.

3.2 Difficulties of Computing Upper Bounds

A simple approach to computing upper bounds on inner products is to approximate each vector. Let $\tilde{\mathbf{x}}$ denote the d'-dimensional approximation of $\mathbf{x}$. Then, the inner product between vectors $\mathbf{x}$ and $\mathbf{y}$ can be approximated by $p(\mathbf{x}, \mathbf{y}) \approx p(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$. However, this direct approximation does not guarantee an upper bound on the inner product. To derive a guaranteed upper bound, we can use error vector $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}'$, where $\mathbf{x}'$ is the d-dimensional reconstructed from $\tilde{\mathbf{x}}$. Using error vectors, the inner product can be decomposed, and we can compute an upper bound as follows:

$$\begin{aligned} p(\mathbf{x},\mathbf{y}) &= p(\mathbf{x}'+\Delta\mathbf{x},\mathbf{y}'+\Delta\mathbf{y}) \\ &= p(\mathbf{x}',\mathbf{y}') + p(\mathbf{x}',\Delta\mathbf{y}) + p(\Delta\mathbf{x},\mathbf{y}') + p(\Delta\mathbf{x},\Delta\mathbf{y}) \\ &\le p(\mathbf{x}',\mathbf{y}') + \|\mathbf{x}'\|\|\Delta\mathbf{y}\| + \|\Delta\mathbf{x}\|\|\mathbf{y}'\| + \|\Delta\mathbf{x}\|\|\Delta\mathbf{y}\|. \end{aligned} \tag{4}$$

This bound, however, is inefficient since it requires $O(d)$ time to compute $p(\mathbf{x}',\mathbf{y}')$ even though it takes $O(1)$ time to compute $\|\mathbf{x}'\|\|\Delta\mathbf{y}\|$, $\|\mathbf{x}'\|\|\Delta\mathbf{y}\|$, and $\|\Delta\mathbf{x}\|\|\Delta\mathbf{y}\|$. Moreover, this bound would be loose, since it separately overestimates three terms: $p(\mathbf{x}',\Delta\mathbf{y})$, $p(\Delta\mathbf{x},\mathbf{y}')$, and $p(\Delta\mathbf{x},\Delta\mathbf{y})$.

3.3 Our Solution: PCA-Based Upper Bound

To enhance efficiency and reduce the number of estimated terms, thereby tightening the upper bound, the proposed approach adopts Principal Component Analysis (PCA) [6], although several approximation methods have been proposed in the literature, such as random projection [26]. Let $\mathbf{w}_i$ denote the i-th eigenvector of the covariance matrix, corresponding to the i-th largest eigenvalue. Using PCA, a vector $\mathbf{x}$ is reconstructed as $\mathbf{x}' = \sum_{i=1}^{d'}(\mathbf{x}\mathbf{w}_i^\top)\mathbf{w}_i$, and the error vector becomes $\Delta\mathbf{x} = \sum_{j=d'+1}^{d}(\mathbf{x}\mathbf{w}_j^\top)\mathbf{w}_j$. Due to the orthogonality of PCA eigenvectors, we have $\mathbf{w}_i\mathbf{w}_j^\top = 0$ if $i \ne j$. Consequently, we have the following equation for $p(\mathbf{x}',\Delta\mathbf{y})$:

$$p(\mathbf{x}',\Delta\mathbf{y}) = \textstyle\sum_{i=1}^{d'}(\mathbf{x}\mathbf{w}_i^\top)\mathbf{w}_i \sum_{j=d'+1}^{d}((\mathbf{y}\mathbf{w}_j^\top)\mathbf{w}_j)^\top = 0. \tag{5}$$

Similarly, we have the following equation for $p(\Delta\mathbf{x},\mathbf{y}')$:

$$p(\Delta\mathbf{x},\mathbf{y}') = \textstyle\sum_{j=d'+1}^{d}(\mathbf{x}\mathbf{w}_j^\top)\mathbf{w}_j^\top \sum_{i=1}^{d'}((\mathbf{y}\mathbf{w}_i^\top)\mathbf{w}_i)^\top = 0. \tag{6}$$

In addition, we have the following equation for $p(\mathbf{x}',\mathbf{y}')$ since we have $\mathbf{w}_i\mathbf{w}_j^\top = 1$ if $i = j$:

$$p(\mathbf{x}',\mathbf{y}') = \textstyle\sum_{i=1}^{d'}(\mathbf{x}\mathbf{w}_i^\top)\mathbf{w}_i \sum_{i=1}^{d'}((\mathbf{y}\mathbf{w}_i^\top)\mathbf{w}_i)^\top = p(\tilde{\mathbf{x}},\tilde{\mathbf{y}}). \tag{7}$$

Thus, using PCA, Equation (4) simplifies to:

$$\begin{aligned} &p(\mathbf{x},\mathbf{y}) = p(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) + p(\Delta\mathbf{x},\Delta\mathbf{y}) \\ &= p(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) + \|\Delta\mathbf{x}\|\|\Delta\mathbf{y}\|\cos(\varphi_{\Delta\mathbf{x},\Delta\mathbf{y}}) \le p(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) + \|\Delta\mathbf{x}\|\|\Delta\mathbf{y}\|. \end{aligned} \tag{8}$$

Although the upper bound given by Equation (4) requires $O(d)$ time, we can compute the upper bound of this equation in $O(d')$ time since it takes $O(d')$ time to compute $p(\tilde{\mathbf{x}},\tilde{\mathbf{y}})$. As a result, we can improve the efficiency. Furthermore, unlike Equation (4), which requires estimating three terms, this formulation estimates only one term, $p(\Delta\mathbf{x},\Delta\mathbf{y})$, improving the bound's tightness. In the proposed approach, we set the reduced dimensionality to $d' = \log d$. The approximate vector is then $\tilde{\mathbf{x}} = [\tilde{x}[1],\ldots,\tilde{x}[\log d]]$ where $\tilde{x}[i] = \mathbf{x}\mathbf{w}_i^\top$. Using PCA's orthogonality, we can compute the error vector as $\Delta\mathbf{x} = \mathbf{x} - \sum_{i=1}^{\log d}\tilde{x}[i]\mathbf{w}_i$.

3.4 Error Segmentation

Despite the improved bound obtained, it may still be loose under the following conditions: (1) error vector $\Delta\mathbf{x}$ and $\Delta\mathbf{y}$ have large norms, and (2) $\cos(\varphi_{\Delta\mathbf{x},\Delta\mathbf{y}})$ is small. To address the first issue, we divide the d-dimensional error vectors into $\log d$ equal-length segments. If $\Delta\mathbf{x}_i = [\Delta x_i[1],\ldots,\Delta x_i[d/\log d]]$ is the i-th subvector of $\Delta\mathbf{x}$, its j-th element is given by $\Delta x_i[j] = \Delta x[j+(i-1)d/\log d]$. To address the second issue, we introduce reference vector $\mathbf{r}_i$ for the i-th segment. Using the inequality $\varphi_{\Delta\mathbf{x}_i,\Delta\mathbf{y}_i} \ge |\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i} - \varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}|$, we obtain $\cos(\varphi_{\Delta\mathbf{x}_i,\Delta\mathbf{y}_i}) \le \cos(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i} - \varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i})$. Note that reference vector $\mathbf{r}_i$ is computed as the mean of all error subvectors in the i-th segment; $\mathbf{r}_i = \frac{1}{n}\sum_{\Delta\mathbf{x}_j\in\Delta\mathbb{X}_i}\Delta\mathbf{x}_j$ where $\Delta\mathbb{X}_i$ is the set of error subvectors for the i-th segment. By segmenting the error vectors and using reference vectors, we derive $\overline{p}(\mathbf{x},\mathbf{y})$ as follows:

Definition 3.1. For vector $\mathbf{x}$ and $\mathbf{y}$ of d dimensions, $\overline{p}(\mathbf{x},\mathbf{y})$ is given as follows:

$$\begin{aligned} \overline{p}(\mathbf{x},\mathbf{y}) = p(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) &+ \textstyle\sum_{i=1}^{\log d}\|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\cos(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\cos(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}) \\ &+ \textstyle\sum_{i=1}^{\log d}\|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\sin(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\sin(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}). \end{aligned} \tag{9}$$

The following lemma shows the upper bound property of $\overline{p}(\mathbf{x},\mathbf{y})$ for $p(\mathbf{x},\mathbf{y})$:

Lemma 3.2. *$\overline{p}(\mathbf{x},\mathbf{y})$ provides an upper bound of inner product $p(\mathbf{x},\mathbf{y})$, i.e., $\overline{p}(\mathbf{x},\mathbf{y}) \ge p(\mathbf{x},\mathbf{y})$.*

Proof Since $\Delta\mathbf{x}_i$ is the i-th subvector of error vector $\Delta\mathbf{x}$, given the inequality $\varphi_{\Delta\mathbf{x}_i,\Delta\mathbf{y}_i} \ge |\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i} - \varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}|$, we can bound residual inner product $p(\Delta\mathbf{x},\Delta\mathbf{y})$ as follows:

$$\begin{aligned} p(\Delta\mathbf{x},\Delta\mathbf{y}) &= \textstyle\sum_{i=1}^{\log d} p(\Delta\mathbf{x}_i,\Delta\mathbf{y}_i) \\ &= \textstyle\sum_{i=1}^{\log d}\|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\cos(\varphi_{\Delta\mathbf{x}_i,\Delta\mathbf{y}_i}) \\ &\le \textstyle\sum_{i=1}^{\log d}\|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\cos(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i} - \varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}) \\ &= \textstyle\sum_{i=1}^{\log d}\|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\cos(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\cos(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}) \\ &\quad + \textstyle\sum_{i=1}^{\log d}\|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\sin(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\sin(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}). \end{aligned} \tag{10}$$

Therefore, we have $p(\mathbf{x},\mathbf{y}) \le \overline{p}(\mathbf{x},\mathbf{y})$ from Equation (8). □

Lemma 3.3. *Upper bound $\overline{p}(\mathbf{x},\mathbf{y})$ can be computed in $O(\log d)$ time using the approximate vectors, the error vectors, and their angles to the reference vectors.*

Proof It needs $O(\log d)$ time to compute inner product $p(\tilde{\mathbf{x}},\tilde{\mathbf{y}})$. In addition, $\|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\cos(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\cos(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i})$ is computed in $O(1)$ time. As a result, $\sum_{i=1}^{\log d}\|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\cos(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\cos(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i})$ is computed at $O(\log d)$ time. Similarly, it takes $O(\log d)$ time to compute $\sum_{i=1}^{\log d}\|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\sin(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\sin(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i})$. Therefore, the total computational cost to compute $\overline{p}(\mathbf{x},\mathbf{y})$ is $O(\log d)$. □

These lemmas show that we can compute upper bound $p(\mathbf{x},\mathbf{y})$ efficiently after PCA.

3.5 Incremental Computation

By estimating upper bounds, the proposed approach can prune unnecessary inner product computations during the top-k node search and edge selection processes. For instance, in the top-k search, we skip the exact computation of $p(\mathbf{x},\mathbf{u})$ if $\overline{p}(\mathbf{x},\mathbf{u}) \le p(\mathbf{x},\mathbf{v})$. Similarly, in the edge selection, we avoid computing $p(\mathbf{u},\mathbf{v})$ if $\overline{p}(\mathbf{u},\mathbf{v}) \le p(\mathbf{x},\mathbf{u})$. However, if these conditions do not hold, we need to compute exact inner products to guarantee the exactness of the graph construction results. To improve efficiency, we exploit the observation for these pruning conditions that, as the upper bound would be tight, we can reduce the number of exact inner product computations. In this section, we propose the approach to enhancing the estimation quality by incrementally computing upper bounds.

As shown in Equation (10), the proposed approach estimates the upper bound in each segment by using the norms of the subvectors and their angles to the reference vector. Specifically,

Algorithm 4 Inner Product Computation

```
Input: vector $\mathbf{x}$ and $\mathbf{y}$, approximated vector $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$, subvector $\Delta\mathbf{x}_i$ and
    $\Delta\mathbf{y}_i$, angle $\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i}$ and $\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}$, and threshold $\theta$
Output: inner product $p(\mathbf{x},\mathbf{y})$
 1: compute $\overline{p}(\mathbf{x},\mathbf{y})$ from Equation (9);
 2: $\overline{p}_0(\mathbf{x},\mathbf{y}) \leftarrow \overline{p}(\mathbf{x},\mathbf{y})$;
 3: for $i \leftarrow 1$ to $\log d$ do
 4:     if $\overline{p}_{i-1}(\mathbf{x},\mathbf{y}) \leq \theta$ then
 5:         $p(\mathbf{x},\mathbf{y}) \leftarrow \overline{p}_{i-1}(\mathbf{x},\mathbf{y})$;
 6:         return $p(\mathbf{x},\mathbf{y})$;
 7:     else
 8:         compute $\overline{p}_i(\mathbf{x},\mathbf{y})$ from Equation (15);
 9: $p(\mathbf{x},\mathbf{y}) \leftarrow \overline{p}_i(\mathbf{x},\mathbf{y})$;
10: return $p(\mathbf{x},\mathbf{y})$;
```

the proposed approach estimates inner product $p(\Delta\mathbf{x}_i, \Delta\mathbf{y}_i)$ in the i-th segment as follows:

$$p(\Delta\mathbf{x}_i,\Delta\mathbf{y}_i) \leq \|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\cos(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\cos(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}) + \|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\sin(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\sin(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}). \tag{11}$$

To improve the estimation quality, the proposed approach iteratively unfolds each segment by exactly computing the inner product of the segment. Let $\overline{p}_i(\mathbf{x},\mathbf{y})$ denote the upper bound obtained after unfolding the first i segments. From Equations (8) and (10), $\overline{p}_i(\mathbf{x},\mathbf{y})$ is given as follows:

$$\begin{aligned}\overline{p}_i(\mathbf{x},\mathbf{y}) &= p(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) + \textstyle\sum_{j=1}^{i} p(\Delta\mathbf{x}_j,\Delta\mathbf{y}_j)\\ &+ \textstyle\sum_{j=i+1}^{\log d}\|\Delta\mathbf{x}_j\|\|\Delta\mathbf{y}_j\|\cos(\varphi_{\Delta\mathbf{x}_j,\mathbf{r}_j})\cos(\varphi_{\Delta\mathbf{y}_j,\mathbf{r}_j})\\ &+ \textstyle\sum_{j=i+1}^{\log d}\|\Delta\mathbf{x}_j\|\|\Delta\mathbf{y}_j\|\sin(\varphi_{\Delta\mathbf{x}_j,\mathbf{r}_j})\sin(\varphi_{\Delta\mathbf{y}_j,\mathbf{r}_j}).\end{aligned} \tag{12}$$

When $i = 0$ in this equation, we have $\overline{p}_i(\mathbf{x},\mathbf{y}) = \overline{p}(\mathbf{x},\mathbf{y})$ since no segments are unfolded. When $i = \log d$, Equation (8) yields the following equation since $\sum_{j=1}^{\log d} p(\Delta\mathbf{x}_j,\Delta\mathbf{y}_j) = p(\Delta\mathbf{x},\Delta\mathbf{y})$:

$$\overline{p}_i(\mathbf{x},\mathbf{y}) = p(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) + p(\Delta\mathbf{x},\Delta\mathbf{y}) = p(\mathbf{x},\mathbf{y}). \tag{13}$$

Therefore, the exact inner product can be obtained from the unfolded upper bound. Moreover, this equation shows that our approach does not require direct access to the original vectors $\mathbf{x}$ and $\mathbf{y}$ to compute the exact inner product; it can be obtained solely from the approximated vectors and their corresponding error vectors. Furthermore, it is clear that we have $\overline{p}_i(\mathbf{x},\mathbf{y}) \leq \overline{p}_{i-1}(\mathbf{x},\mathbf{y})$ from Equation (11) and (12). This indicates that the upper bound becomes tighter as each segment is unfolded.

Although $\overline{p}_i(\mathbf{x},\mathbf{y})$ provides a tight bound, computing it directly from Equation (12) incurs a high cost of $O(\log d + id/\log d)$ since it needs to compute the inner products of the unfolded segments. To efficiently compute $\overline{p}_i(\mathbf{x},\mathbf{y})$, we use the following equation derived from Equation (12):

$$\begin{aligned}&\overline{p}_i(\mathbf{x},\mathbf{y}) - \overline{p}_{i-1}(\mathbf{x},\mathbf{y})\\ =&p(\Delta\mathbf{x}_i,\Delta\mathbf{y}_i) - \|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\cos(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\cos(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i})\\ &- \|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\sin(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\sin(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}).\end{aligned} \tag{14}$$

Therefore, we incrementally update $\overline{p}_i(\mathbf{x},\mathbf{y})$ from $\overline{p}_{i-1}(\mathbf{x},\mathbf{y})$ as

$$\begin{aligned}\overline{p}_i(\mathbf{x},\mathbf{y}) &= \overline{p}_{i-1}(\mathbf{x},\mathbf{y}) + p(\Delta\mathbf{x}_i,\Delta\mathbf{y}_i)\\ &- \|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\cos(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\cos(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i})\\ &- \|\Delta\mathbf{x}_i\|\|\Delta\mathbf{y}_i\|\sin(\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i})\sin(\varphi_{\Delta\mathbf{y}_i,\mathbf{r}_i}).\end{aligned} \tag{15}$$

Since it takes $O(d/\log d)$ time to compute $p(\Delta\mathbf{x}_i,\Delta\mathbf{y}_i)$, this incremental update requires $O(d/\log d)$ time. Moreover, when $i = \log d$ and all segments are unfolded, this process yields the exact inner product as $p(\mathbf{x},\mathbf{y}) = \overline{p}_{\log d}(\mathbf{x},\mathbf{y})$ in $O(d)$ time.

Algorithm 5 Preprocess

```
Input: vectors $\mathbb{X}$
Output: approximated vector $\tilde{\mathbf{x}}$, subvector $\Delta\mathbf{x}_i$, and angle $\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i}$ for
    each vector
 1: compute PCA by randomly sampling $\log n$ vectors;
 2: for each $\mathbf{x} \in \mathbb{X}$ do
 3:     compute $\tilde{\mathbf{x}}$ and $\Delta\mathbf{x}$;
 4:     discard vector $\mathbf{x}$ to free memory;
 5: sort the dimensions of the error vectors in descending order of the
    average errors;
 6: for each $\Delta\mathbf{x} \in \Delta\mathbb{X}$ do
 7:     split $\Delta\mathbf{x}$ into $\log d$ subvectors;
 8: for $i = 1$ to $\log d$ do
 9:     compute $\mathbf{r}_i$;
10:     for each $\Delta\mathbf{x}_i \in \Delta\mathbb{X}_i$ do
11:         compute $\varphi_{\Delta\mathbf{x}_i,\mathbf{r}_i}$ and $\|\Delta\mathbf{x}_i\|$;
```

Algorithm 4 presents our procedure for computing the upper bound of the inner product in order to check whether it exceeds given threshold θ. Note that θ is set as $p(\mathbf{x},\mathbf{v})$ in the top-k search and it is set as $p(\mathbf{x},\mathbf{u})$ in the edge selection. It begins by initializing $\overline{p}_0(\mathbf{x},\mathbf{y})$ as $\overline{p}_0(\mathbf{x},\mathbf{y}) = \overline{p}(\mathbf{x},\mathbf{y})$ (lines 1-2). During the iterative process, if $\overline{p}_{i-1}(\mathbf{x},\mathbf{y}) \leq \theta$, it terminates the iteration since the upper bound guarantees that $p(\mathbf{x},\mathbf{y}) > \theta$ cannot hold (lines 4-6). Otherwise, it incrementally updates upper bound $\overline{p}_i(\mathbf{x},\mathbf{y})$ using Equation (15) (lines 7-8). After the iteration completes, it sets $\overline{p}(\mathbf{x},\mathbf{y})$ to $\overline{p}_i(\mathbf{x},\mathbf{y})$, reflecting that the exact inner product is obtained from the upper bound after the iteration (lines 9-10).

As shown in Algorithm 4, the upper bound is refined incrementally by computing exact inner products starting from the initial segment. To effectively improve the estimation quality in the iteration, our approach sorts the dimensions of the error vectors so that segments with larger approximation errors are unfolded earlier in the iteration. This sorting strategy is detailed in the next section.

3.6 Graph Construction Algorithm

Our graph construction algorithm begins by computing the approximated vectors, error vectors, and angle and norm of each subvector (components required to estimate the upper bounds) before constructing the graph. As shown in Algorithm 5, we first compute the approximated vectors and their error vectors via PCA (lines 1-4). To improve efficiency, we randomly sample $\log n$ vectors for PCA computation. As shown in Equation (13), the proposed approach does not rely on the original vectors to compute inner products. Therefore, our approach discards each original vector $\mathbf{x}$ after its approximated and error vectors have been computed, thereby freeing memory (line 4). To enhance the estimation quality, we sort the dimensions of error vectors in descending order based on their average error (line 5). The average error of the i-th dimension is computed as $\frac{1}{n}\sum_{j=1}^{n}|\Delta x_j[i]|$. Subsequently, we divide each error vector into subvectors and compute the angles between the subvectors and their corresponding reference vectors (lines 6-11). The graph is then constructed using Algorithm 6, which follows the same procedures as the original approach of Algorithm 1, by iteratively inserting vectors into an empty graph. However, unlike the original approach, we leverage Algorithm 4 to reduce exact inner product computations during the top-k search and edge selection. In particular, instead of directly computing $p(\mathbf{x},\mathbf{u})$ in line 8 of Algorithm 2, we invoke Algorithm 4, using $p(\mathbf{x},\mathbf{v})$ as threshold θ, to efficiently

Algorithm 6 Proposed Graph Construction

```
Input: vectors $\mathbb{X}$, number of top-$k$ nodes $k$, and number of edges $m$
Output: proximity graph $\mathbb{G}$
1: compute approximated vector, subvectors, and angles for each vector by Algorithm 5;
2: initialize $\mathbb{G}$ as $\emptyset$;
3: for each $\mathbf{x} \in \mathbb{X}$ do
4:     compute top-$k$ nodes $\mathbb{K}$ in $\mathbb{G}$ by Algorithm 2 where Algorithm 4 is used in line 8 by setting $\theta \leftarrow p(\mathbf{x}, \mathbf{v})$;
5:     add $\mathbf{x}$ to $\mathbb{G}$ as a node;
6:     add $m$ edges of $\mathbf{x}$ to $\mathbb{G}$ obtained by Algorithm 3 where Algorithm 4 is used in line 6 by setting $\theta \leftarrow p(\mathbf{x}, \mathbf{u})$;
7: return $\mathbb{G}$;
```

identify the top-k nodes (line 4). Likewise, in the edge selection, we compute $p(\mathbf{u}, \mathbf{v})$ using Algorithm 4, with threshold θ set to $p(\mathbf{x}, \mathbf{u})$ in line 6 of Algorithm 3 (line 6).

As shown in Algorithm 1, the proposed approach constructs the graph by inserting vectors one by one into an initially empty graph, in the same manner as ip-NSW. Therefore, the proposed approach supports multi-threaded CPU acceleration by inserting vectors in parallel, analogous to the original ip-NSW implementation (https://github.com/stanis-morozov/ip-nsw). Our algorithm has the following properties:

Theorem 3.4. *Algorithm 6 yields the same graph construction results as the original ip-NSW algorithm.*

Proof Algorithm 6 employs Algorithm 4 with threshold $\theta = p(\mathbf{x}, \mathbf{v})$ in the top-k search of Algorithm 2. Thus, exact inner product $p(\mathbf{x}, \mathbf{u})$ is skipped if $\overline{p}_{i-1}(\mathbf{x}, \mathbf{u}) \leq p(\mathbf{x}, \mathbf{v})$ holds during the i-th iteration of Algorithm 4. Given the upper bounding property, this indicates $p(\mathbf{x}, \mathbf{u}) \leq \overline{p}_{i-1}(\mathbf{x}, \mathbf{u}) \leq p(\mathbf{x}, \mathbf{v})$; node $\mathbf{u}$ cannot be a top-k node for $\mathbf{x}$. Therefore, the pruning does not affect the search result. Similarly, Algorithm 4 is used with $\theta = p(\mathbf{x}, \mathbf{u})$ during the edge selection of Algorithm 3. If $\overline{p}_{i-1}(\mathbf{u}, \mathbf{v}) \leq p(\mathbf{x}, \mathbf{u})$, the exact computation of $p(\mathbf{u}, \mathbf{v})$ is skipped. Since the edge to node $\mathbf{u}$ is selected when $p(\mathbf{u}, \mathbf{v}) > p(\mathbf{x}, \mathbf{u})$, this pruning does not affect the edge selection. Therefore, Algorithm 6 constructs the same graph as ip-NSW. □

Theorem 3.5. *If s denotes the average number of dimensions included in the unfolded segments, then the time and space complexities of Algorithm 6 are $O(d(n \log d + d \log n) + nl(s + \log k))$ and $O(n(m + d))$, respectively.*

Proof During preprocessing, $O(d^2 \log n)$ time is required for PCA and $O(nd \log d)$ time for approximated and error vectors. Sorting the dimensions of error vectors takes $O(nd + d \log d)$, and splitting the error vectors and computing angles needs $O(nd)$ time. Thus, preprocessing costs $O(d^2 \log n + nd \log d)$ time. In the top-k search, identifying nodes with the lowest inner products takes $O(nl \log k)$ time. Initializing upper bounds requires $O(nl \log d)$ time, and updating them takes $O(nls)$. For edge selection, sorting top-k nodes requires $O(nk \log k)$ time. Initializing upper bounds takes $O(nmk \log d)$, and updating them takes $O(nmks)$. As a result, the overall time complexity of Algorithm 6 is $O(d(n \log d + d \log n) + nl(s + \log k))$.

Regarding memory usage, our approach additionally needs $O(d^2)$ space for the covariance matrix to compute PCA. Storing the approximated and error vectors takes $O(n \log d)$ and $O(nd)$, respectively. Additionally, it needs $O(n \log d)$ spaces to hold the angle and norm of each subvector. Furthermore, $O(nm)$ and $O(k)$ space are needed to store edges and top-k nodes. Therefore, the total memory cost of Algorithm 6 is $O(n(m + d))$. □

These theorems show that our approach constructs the graph efficiently, guarantees equivalent construction results, and incurs the same memory cost as the original approach.

3.7 Extensions of the Proposed Approach

Although we have focused on the graph construction of ip-NSW, our approach can also be extended to its variants. As discussed in Section 2, IPDG computes $p(\mathbf{u}, \mathbf{v})$ during edge selection [40]. Since our approach can compute upper bounds for $p(\mathbf{u}, \mathbf{v})$, it can be used to improve the efficiency of IPDG. Similarly, it can reduce the computational cost of NAPG, which computes $p(\mathbf{u}, \mathbf{v})$ during edge selection [39]. Furthermore, because Möbius-Graph employs the same graph construction algorithm as ip-NSW, the proposed approach can be directly applied without modification.

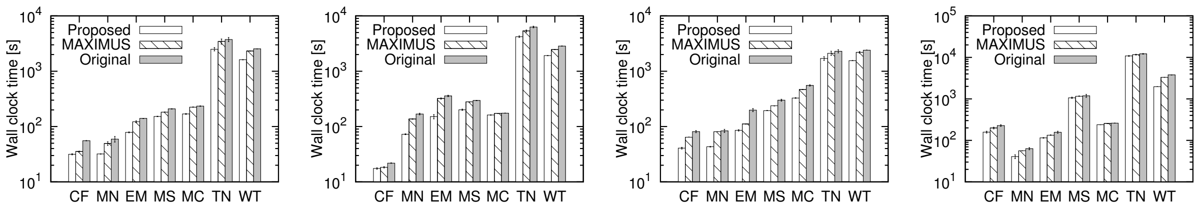
Moreover, our approach is also applicable to the search step, although our primary focus has been on reducing the cost of graph construction. This is because the search procedures in ip-NSW and its variants rely on Algorithm 2, where Algorithm 4 can be employed to reduce processing time. Furthermore, our approach can be extended to support non-metric similarity measures beyond the inner product, as discussed in the appendix.

Additionally, our approach efficiently constructs the graph in a streaming setting where vectors arrive continuously, one by one. As shown in Algorithm 6, the graph is incrementally constructed by inserting each vector into an initially empty graph. Consequently, it supports graph construction over a stream of n vectors with a time complexity of $O(nl(\log k + \log d + s))$ assuming that preprocessing of PCA is performed. For the preprocessing cost of $O(d^2 \log n + nd \log d)$, PCA is computed from a random sample of $\log n$ vectors, as shown in Algorithm 5, Therefore, we update the PCA for all arrived vectors only when the number of vectors reaches 2^i $(i = 1, \ldots, \log n)$. At other arrival counts, our approach computes the PCA only for the vector that has just arrived at $O(d)$ time. As a result, it can perform efficient graph construction even if the number of vectors grows incrementally.

4 Related Work

Since ip-NSW and its variants achieve strong performance in both efficiency and accuracy, they have become essential components of information systems, supporting a wide range of applications. For example, Möbius-Graph is used in sponsored search [52]. Sponsored search is a promotional model used by search engines and e-commerce platforms to allocate limited advertisement slots to advertisers in response to user queries [33, 34]. Since a key technical challenge of sponsored search lies in efficiently and accurately retrieving relevant advertisements from massive candidate pools, Möbius-Graph is an impactful solution that powers large-scale sponsored search in digital advertising. Furthermore, Möbius-Graph is used in plant disease detection [32]. Since it can efficiently and effectively retrieve the most relevant leaf images, it enables accurate disease classification. In addition to sponsored search and plant disease detection, ip-NSW and its variants are widely used in various applications, including item recommendation [11], data summarization [43], and video search [48].

Nearest neighbor search based on l_2-distance has long been a popular topic in the database community. Traditional approaches employ tree-based structures such as R-tree [17], K-D tree [5], and SR-tree [22], achieving $O(\log n)$ query time. Although accurate, these approaches become inefficient in high dimensions due to



(1-1) ip-NSW (1-2) IPDG (1-3) NAPG (1-4) Möbius-Graph

Figure 1: Graph construction time of each approach.

Table 2: Characteristics of the experimental data.

Dataset (Abbreviation)	#Vectors	#Dimensions
CIFAR-10 (CF)	60,000	3,072
MNIST (MN)	70,000	780
EMNIST (EM)	131,600	784
Msong (MS)	994,185	420
Microsoft (MC)	1,200,192	136
Tiny5M (TN)	5,000,000	384
Weather (WT)	16,951,828	103

the curse of dimensionality [18]. Furthermore, unlike l_2-distance, the inner product is a non-metric similarity measure lacking the non-negativity, the identity of indiscernibles, and the triangle inequality [11], as described in Section 2.

To perform exact MIPS, LEMP partitions vectors into buckets of similar l_2-norms to transform the MIPS problem into multiple smaller cosine similarity search problems [41]. It leverages vector norms to prune buckets that cannot exceed a given threshold and employs adaptive per-bucket approaches to reduce computations. FEXIPRO is an exact MIPS algorithm designed for recommender systems [25]. This approach accelerates MIPS through an SVD-based transformation that reorders dimensions so that leading ones dominate, enabling early pruning. MAXIMUS achieves exact MIPS by clustering vectors using k-means into representative centroids [1]. It computes upper bounds on inner products for vectors within each cluster to generate a sorted list of candidates.

Several approaches have been proposed to approximately perform MIPS. Quantization-based approaches, such as norm-explicit quantization, anisotropic quantization, and Query-aware Quantization, transform each vector into a short sequence of codewords using predefined codebooks [8, 16, 49]. They efficiently compute inner products via lookup tables that store pre-computed inner products between codewords. Clustering-based approaches perform clustering to partition the search space [7, 10]. Hashing-based approaches, such as ALSH, SIMPLE-LSH, and RANGE-LSH, map vectors into buckets using hash functions designed so that similar vectors fall into the same bucket [31, 36, 45]. As noted in previous studies [10, 30, 39, 40, 54], graph-based approaches outperform the alternatives in MIPS, achieving superior performance in both accuracy and efficiency.

5 Experimental Evaluation

We conducted experiments to evaluate the effectiveness of our approach when applied to ip-NSW and its variants. In the following, "Original" refers to the baseline approaches: ip-NSW, IPDG, NAPG, and Möbius-Graph. "Proposed" denotes our approach integrated into each of these approaches. "MAXIMUS" represents the results of applying MAXIMUS [1] to the original

Table 3: Preprocess and graph construction time [s].

Process	CF	MN	EM	MS	MC	TN	WT
Preprocess	6.6	1.0	1.8	5.9	2.2	26.4	21.3
Graph construction	24.9	31.0	76.8	146.6	166.7	2499.2	1612.2
Preprocess ratio [%]	21.0	3.1	2.3	3.9	1.3	1.0	1.3

Table 4: Memory sizes [GB].

	CF	MN	EM	MS	MC	TN	WT
w/o discard	1.19	0.37	0.74	3.30	1.46	15.41	16.48
Proposed	0.62	0.19	0.39	1.76	0.86	8.28	9.98
Original	0.58	0.18	0.37	1.68	0.77	7.83	8.84

algorithms. It improves MIPS by computing upper bounds after clustering vectors. It outperforms LEMP [41] and FEXIPRO [25] in finding vectors of high inner products, as shown in [1]. We evaluated all approaches on datasets summarized in Table 2. CIFAR-10 (CF), MNIST (MN), EMNIST (EM), and Microsoft (MC) datasets were obtained from OpenML (https://api.openml.org/), Msong (MS) and Tiny5M (TN) from IRISA (https://www.irisa.fr/), and Weather (WT) from Kaggle (https://www.kaggle.com/). Following prior work [39, 40], we set the number of top-k nodes to $k = 100$ and the number of edges per node to $m = 16$. For NAPG, we computed α using the three norm ranges described in [39]. In MAXIMUS, the numbers of clusters and clustering iterations were set to eight and three, respectively, following [1]. All approaches were implemented in C++ and evaluated on a Linux server with Intel Xeon Platinum 8280 CPUs (2.70 GHz) using a single thread. It had 1.5 TB of RAM and a 1 TB hard drive. Each approach was implemented by modifying the original ip-NSW code, which employs a multi-layer graph for MIPS, although a single-layer graph was assumed in earlier sections.

5.1 Graph Construction Time

Figure 1 shows the average graph construction time with the standard deviations of ip-NSW and its variants, based on ten trials. Table 3 reports the average preprocessing and graph construction times of our approach for ip-NSW. Table 4 shows the memory requirements of the proposed and original approaches for ip-NSW. In this table, "w/o discard" denotes a variant of our approach that does not discard the original vectors during preprocessing. Additionally, Figure 2 presents the graph construction time for ip-NSW on the MNIST dataset as the number of threads increases, evaluating the effectiveness of the parallelization approach.

Figure 1 shows that the proposed approach improves the efficiency of the graph-based approaches. Specifically, our approach reduced graph construction time by up to 52.9% compared to

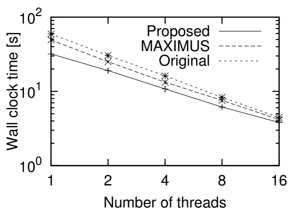


Figure 2: Graph construction time vs. number of threads.

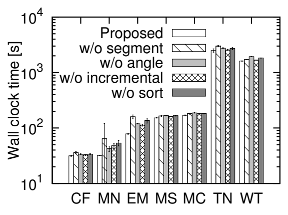


Figure 3: Ablation study for each component.

MAXIMUS and 57.6% compared to the original approach. The original approach performs a greedy search to identify the top-k nodes and computes inner products between the top-k nodes to select edges, leading to a large number of expensive inner product computations. In contrast, the proposed approach reduces this cost by pruning unnecessary computations using upper bounds. Specifically, our approach effectively computes the bounds by segmenting error vectors and leveraging reference vectors. Furthermore, the proposed approach incrementally computes the bounds after sorting the error vector dimensions to enhance estimation accuracy. As shown in Algorithm 6, the proposed approach requires preprocessing vectors to enable bound computation prior to graph construction. However, as shown in Table 3, this preprocessing overhead is relatively small compared to the processing time to construct the graph. Although preprocessing time increases with vector dimensionality, the ratio of preprocessing time to total construction time decreases as the dataset size grows. This is because PCA can be efficiently computed by sampling $\log n$ vectors, as described in Algorithm 5. While MAXIMUS also employs upper bounds, it is slower than our approach since it fails to compute the upper bounds effectively.

Additionally, Table 4 demonstrates that our approach can effectively reduce the memory usage by discarding the original vectors during preprocessing. As discussed in Theorem 3.5, compared to the original approach, our approach additionally stores the covariance matrix, the angle and norm of each subvector, and the approximated and error vectors. Among these, the error vectors require the largest memory space of $O(nd)$, the same as the original vectors. However, as shown in Algorithm 5, the original vectors are discarded during preprocessing since the exact inner product can be obtained from Equation (13). Consequently, the proposed approach incurs a small memory overhead, as confirmed in Table 4.

Furthermore, Figure 2 demonstrates that our approach effectively reduces graph construction time as the number of threads increases. This is because the proposed approach inserts vectors in parallel into an initially empty graph using multi-threading, as described in Section 3.6. Besides, since each insertion is performed more efficiently than in MAXIMUS and the original approach, our approach achieves greater efficiency in the multi-threaded setting, as shown in Figure 2.

As shown in prior work [11, 32, 43, 48, 52], proximity graphs are typically constructed in the indexing phase to enable efficient MIPS in real-world systems. Since our approach reduces construction time by more than half, it effectively doubles the number of indexed vectors that can be processed within the same time budget. As a result, our approach improves the scalability of the real-world systems. For example, a system that indexes 1 million vectors can index over 2 million vectors within the same time budget, enabling deployment on substantially larger datasets. Furthermore, our approach increases refresh rates with lowered operational costs. In sponsored search and recommendation, graph construction is often executed periodically (e.g., daily or weekly) to incorporate newly added vectors. Halving the construction time directly enables more frequent index refreshes, providing fresher results with reduced cloud computing expenditure during the indexing phase.

Table 5: Recall of each approach [%].

Data	Approach	ip-NSW	IPDG	NAPG	Möbius-Graph
CF	Proposed	99.3±3.1	30.0±0.3	**99.9±0.8**	85.2±6.8
	MAXIMUS	99.3±3.1	30.0±0.3	**99.9±0.8**	85.2±6.8
	Original	99.3±3.1	30.0±0.3	**99.9±0.8**	85.2±6.8
MN	Proposed	77.1±19.3	85.4±19.7	87.3±15.8	**99.4±2.8**
	MAXIMUS	77.1±19.3	85.4±19.7	87.3±15.8	**99.4±2.8**
	Original	77.1±19.3	85.4±19.7	87.3±15.8	**99.4±2.8**
EM	Proposed	83.2±16.4	92.5±10.4	91.6±11.0	**97.5±5.8**
	MAXIMUS	83.2±16.4	92.5±10.4	91.6±11.0	**97.5±5.8**
	Original	83.2±16.4	92.5±10.4	91.6±11.0	**97.5±5.8**
MS	Proposed	**99.8±0.2**	65.1±16.1	38.2±13.8	45.2±38.4
	MAXIMUS	**99.8±0.2**	65.1±16.1	38.2±13.8	45.2±38.4
	Original	**99.8±0.2**	65.1±16.1	38.2±13.8	45.2±38.4
MC	Proposed	**89.0±0.2**	68.6±25.1	49.8±44.1	5.7±20.9
	MAXIMUS	**89.0±0.2**	68.6±25.1	49.8±44.1	5.7±20.9
	Original	**89.0±0.2**	68.6±25.1	49.8±44.1	5.7±20.9
TN	Proposed	96.7±6.6	98.8±4.4	**99.7±2.7**	15.3±16.9
	MAXIMUS	96.7±6.6	98.8±4.4	**99.7±2.7**	15.3±16.9
	Original	96.7±6.6	98.8±4.4	**99.7±2.7**	15.3±16.9
WT	Proposed	56.1±24.9	**59.5±22.5**	0.0±0.0	42.8±29.3
	MAXIMUS	56.1±24.9	**59.5±22.5**	0.0±0.0	42.8±29.3
	Original	56.1±24.9	**59.5±22.5**	0.0±0.0	42.8±29.3

5.2 Effectiveness of Each Component

We conducted ablation studies to show the effectiveness of each component in our approach. Figure 3 shows the graph construction time for ip-NSW under four settings: "w/o segment", "w/o angle", "w/o incremental", and "w/o sort", which respectively turn off error vector segmentation, reference vectors, incremental computation, and dimension-wise sorting. As shown in the figure, "w/o segment" yields the slowest performance on the CIFAR-10 (CF), MNIST (MN), EMNIST (EM), and Tiny5M (TN) datasets. This indicates that segmenting error vectors is particularly effective for high-dimensional data since they have higher dimensionality. The upper bound is computed using the norms and angles of subvectors, as shown in Equation (11). Since error vectors have large norms, if a dataset has high dimensionality, we can effectively reduce their norms by segmenting error vectors, making the upper bound effective in high-dimensional settings. Besides, "w/o angle" performs worst on the Msong (MS), Microsoft (MC), and Weather (WT) datasets, highlighting the importance of reference vectors for them. When subvectors are relatively short, their angles influence the inner product more than their norms. Thus, incorporating reference vectors allows for more accurate upper-bound estimation on such datasets.

5.3 Graph Construction Result

As discussed in Section 3.6, a key advantage of our approach is that it guarantees identical graphs to those produced by the original approaches. To validate this advantage, we evaluated the search accuracy by randomly selecting 10,000 query vectors,

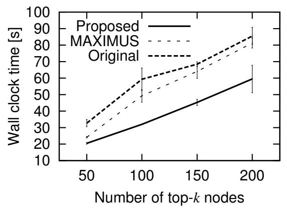


Figure 4: Graph construction time vs. number of top-k nodes.

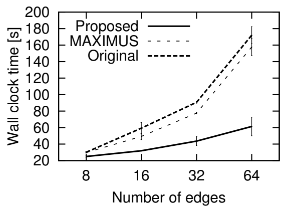


Figure 5: Graph construction time vs. number of edges.

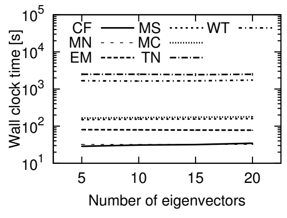


Figure 6: Graph construction time vs. number of eigenvectors.

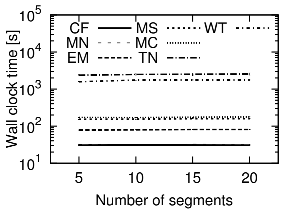


Figure 7: Graph construction time vs. number of segments.

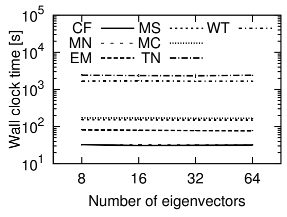


Figure 8: Graph construction time vs. number of samples.

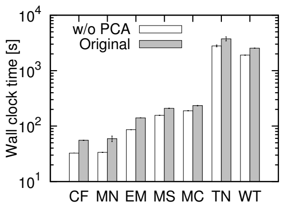


Figure 9: Graph construction time without using PCA.

following the prior study [24]. Table 5 presents the top-10 search recall results with the standard deviations. This table shows that the proposed approach, as well as MAXIMUS, achieves identical recall to the original ip-NSW and its variants. This is because they use upper bounds to safely prune redundant inner product computations in the graph construction step without altering the resulting graph. Furthermore, recall varies across datasets depending on the graph construction strategy. Specifically, ip-NSW achieves the highest recall on Msong (MS) and Microsoft (MC), IPDG is the most accurate on Weather (WT), NAPG performs best on CIFAR-10 (CF) and Tiny5M (TN), and Möbius-Graph leads on MNIST (MN) and EMNIST (EM). These differences stem from the distinct construction strategies employed by each method. This suggests that the graph construction strategy should be tailored to datasets to improve search accuracy. Although graph construction is a computational bottleneck for ip-NSW and its variants, our approach improves efficiency without sacrificing search accuracy, as demonstrated in Figure 1 and Table 5. Therefore, our approach enhances the practical usability of ip-NSW and its variants across diverse datasets.

5.4 Number of Top-k Nodes and Edges

This experiment evaluates the graph construction time of ip-NSW with varying numbers of top-k nodes k and edges m. Figure 4 and 5 present the processing times for the MNIST dataset under different k and m settings, respectively; similar trends were observed across other datasets. As shown in Figure 4, our approach consistently outperforms previous approaches, even as k increases. This efficiency stems from using upper bounds to prune unnecessary inner product computations during top-k node search. While MAXIMUS also employs upper bounds, its less effective computation limits its efficiency. Moreover, the original approach relies on a greedy search, increasing computation time as k grows. Figure 5 further shows that the proposed approach reduces processing time for edge selection, even with larger values of m, due to its pruning approach.

5.5 Eigenvectors, Segments, and Samples

In our approach, the default settings for the number of eigenvectors used in PCA and the number of segments are set to $\log d$. Additionally, PCA is computed by sampling $\log n$ vectors. In this experiment, we assessed the graph construction time of ip-NSW by varying the number of eigenvectors, segments, and samples to demonstrate the robustness of our approach concerning these parameters. Figure 6, 7, and 8 illustrate the graph construction times where we varied the number of eigenvectors, segments, and samples, respectively.

As shown in Figure 6, the graph construction time remained unaffected by the number of eigenvectors, indicating that our approach is not sensitive to this parameter in terms of efficiency. We use eigenvectors to approximate vectors, and the efficiency of upper bound computations improves as the number of eigenvectors decreases. However, increasing the number of eigenvectors reduces the number of exact inner product computations by enhancing the quality of the upper bound estimates. This trade-off between the number of eigenvectors and exact inner product computations results in our approach's insensitivity to the number of eigenvectors. Similarly, as shown in Figure 7, the graph construction time was unaffected by the number of segments, reflecting the trade-off between estimation quality and computational cost. Figure 8 further demonstrates that the graph construction time remains stable when varying the number of sampling vectors. Although increasing the number of samples increases the PCA computation cost, it reduces the number of exact inner product computations by improving the quality of the upper bound estimates. Note that our approach maintains consistent search accuracy for the different parameter settings since we can obtain the upper bounds regardless of these parameter settings.

5.6 Upper Bound without PCA

We leverage PCA to improve the tightness of upper bounds. However, when PCA fails to approximate vectors effectively, the efficiency of graph construction may degrade. To assess this impact, we evaluated the performance of our approach without using PCA to estimate the upper bounds, by examining the graph construction time for ip-NSW. In Figure 9, "w/o PCA" is the results of our approach when upper bounds are estimated without PCA.

The figure shows that our approach still constructs the graph more efficiently than the original approach, even without PCA. Specifically, it achieved up to a 43.3% reduction in graph construction time compared to the original approach. When PCA is not used, we have $p(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = 0$ in Equation (9). Nevertheless, this

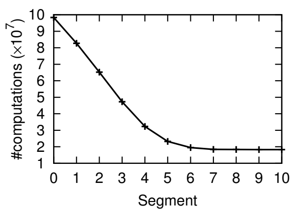


Figure 10: Number of inner product computations in each segment.

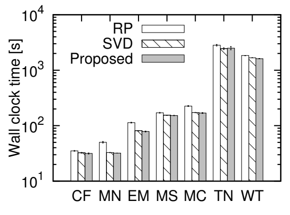


Figure 11: Graph construction time of each approximation approach.

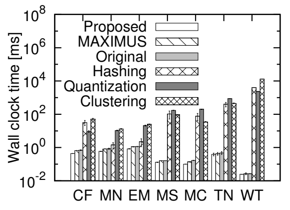


Figure 12: Search time of each approach.

Table 6: Recall of each MIPS approach [%].

Data	Proposed	Hashing	Quantization	Clustering
CF	**99.9 ± 0.8**	1.5 ± 4.3	22.2 ± 11.3	16.7 ± 24.4
MN	**99.4 ± 2.8**	11.2 ± 10.4	31.9 ± 19.5	76.1 ± 18.1
EM	**97.5 ± 5.8**	11.9 ± 10.3	17.9 ± 15.4	60.1 ± 20.1
MS	**99.8 ± 0.2**	0.1 ± 1.5	17.0 ± 14.8	99.7 ± 0.2
MC	**89.0 ± 26.5**	0.3 ± 2.1	14.3 ± 16.8	71.4 ± 41.1
TN	**99.7 ± 2.7**	0.5 ± 2.9	0.5 ± 2.3	10.7 ± 14.5
WT	**59.5 ± 22.5**	0.0 ± 0.0	0.0 ± 0.0	50.9 ± 19.6

equation still enables the computation of upper bounds using segmented error vectors and their corresponding reference vectors. Our approach sorts error vector dimensions in descending order of average error, allowing early termination of the incremental upper bound computation. This improves estimation quality and reduces graph construction time, even without PCA.

5.7 Effectiveness of Upper Bound

We prune unnecessary inner product computations during graph construction by estimating upper bounds. Specifically, we segment the error vectors and iteratively unfold each segment by computing its inner product to improve the bound estimates progressively. To evaluate the effectiveness of this approach, we examined the number of computations of $\overline{p}_i(\mathbf{x}, \mathbf{y})$, the upper bound obtained after unfolding the first i segments. Figure 10 presents the results using ip-NSW for the MNIST dataset. As we use $\log d$ segments, the number of segments is 10 for the MNIST dataset. When no segments are unfolded, we have $i = 0$ and $\overline{p}_i(\mathbf{x}, \mathbf{y}) = \overline{p}(\mathbf{x}, \mathbf{y})$. We needs to compute $\overline{p}(\mathbf{x}, \mathbf{y})$ for all the inner product computations required in the original algorithm, as shown in Algorithm 6. Therefore, segment "0" in Figure 10 reflects the number of exact inner product computations in the original approach. Conversely, when all segments are unfolded, we have $i = \log d$ and $\overline{p}_i(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}, \mathbf{y})$. Therefore, the last segment in the figure corresponds to the number of exact inner product computations in our approach.

Figure 10 shows that the number of upper bound computations decreases as more segments are unfolded. This is because, by Equations (11) and (12), we have $\overline{p}_i(\mathbf{x}, \mathbf{y}) \leq \overline{p}_{i-1}(\mathbf{x}, \mathbf{y})$, allowing for progressively tighter estimates through incremental computation. This approach enables adaptive control over the number of unfolded segments: fewer segments are unfolded for low inner products, while more segments are used for higher inner products. As a result, our approach significantly reduces the number of exact inner product computations compared to the original approach. Specifically, it achieves a reduction of up to 81.4%, as shown in Figure 10.

5.8 Upper Bound with Other Approximations

As described in Section 3.3, our approach employs PCA to compute upper bounds by approximating vectors, leveraging its orthogonality property. Since SVD also has this property, it can be used in place of PCA for upper bound computation without modification. In contrast, since random projection lacks this property, projecting vectors must be orthogonalized to substitute it for PCA. In this experiment, we evaluated the graph construction time of the SVD-based and random projection-based approaches. Figure 11 reports the construction time of ip-NSW, where "SVD" and "RP" denote the SVD-based and random projection-based approaches, respectively. For the random projection-based approach, orthogonalization is performed using the Gram-Schmidt algorithm [9].

As shown in Figure 11, the random projection-based approach takes more time for graph construction than the SVD- and PCA-based approaches. Random projection relies on a randomly generated projection matrix, whereas PCA and SVD exploit the intrinsic structure of the data to obtain more effective vector approximations. Consequently, since random projection produces larger approximation errors than PCA and SVD, it leads to slower graph construction. Regarding the comparison between SVD and PCA, the PCA-based approach achieves slightly higher efficiency. Even though both methods extract low-dimensional structure from the data, PCA centers each dimension before performing the approximation. This centering step normalizes the contribution of different dimensions. On the other hand, SVD, applied without centering, tends to emphasize dimensions with larger magnitudes. As a result, PCA yields lower approximation error, which in turn reduces graph construction time. Note that search accuracy remains unchanged when SVD is used instead of PCA, since the upper bounds are still computed.

5.9 Search Performance

While this paper primarily focuses on improving the efficiency of the graph construction step, our approach also enhances the search step as described in Section 3.7. This is because the search process relies on Algorithm 2 for top-k search, and we can accelerate this algorithm by leveraging Algorithm 4 by pruning inner product computations. In this experiment, we evaluated the search time of each approach and compared our approach against three non-graph-based baselines: a hashing-based approach by Yan et al. [45], a quantization-based approach by Guo et al. [16], and a clustering-based approach by Bruch et al. [7]. The hashing-based approach partitions vectors according to the norm distribution and computes hashes for each vector. This approach yields higher search accuracy than other hashing-based approaches, such as ALSH [36] and SIMPLE-LSH [31]. The quantization-based approach approximates inner products by penalizing errors parallel to query vectors more than orthogonal errors. It has been adopted in recent applications, such as image recognition [20], text retrieval [27], and item recommendation [19]. The clustering-based approach uses spherical k-means to identify candidate vectors. It is employed in Faiss [10], a widely used library for MIPS. For each dataset, we selected the most accurate graph-based variant based on Table 5: ip-NSW for Msong (MS) and Microsoft (MC), IPDG for Weather (WT), NAPG for CIFAR-10 (CF) and Tiny5M (TN), and Möbius-Graph for MNIST (MN) and EMNIST (EM). The same as the previous paper [45], the hashing-based approach partitioned vectors into 32 sub-datasets and used 16 bits to perform MIPS. Following prior work [16, 49], we configured the quantization-based approach with 16 codewords and 25 codebooks. The clustering-based approach used $4\sqrt{n}$ clusters and $0.01n$ candidates, as suggested by [7]. Figure 12 reports the search time, and Table 6 shows the recall for the top-10 search. In this table, we omit results for MAXIMUS and the original approach, as they achieve the same search accuracy as our approach, as shown in Figure 5. "Hashing", "Quantization", and "Clustering" are the results of the hashing-based, quantization-based, and clustering-based approaches, respectively.

As shown in Figure 12, our approach reduces search time by up to 34.1% compared to MAXIMUS and 39.9% compared to the original approach. Moreover, it is up to 163000, 94100, and 526000 times faster than the hashing-based, quantization-based, and clustering-based approaches, respectively. This is because we can improve the efficiency of the search step by effectively pruning inner product computations during top-k search. Although the hashing-based approach can efficiently obtain candidate vectors, it needs to compute exact inner products of all candidate vectors to find the top-k nodes. The quantization-based approach incurs a high cost since it computes inner products for all vectors, although it efficiently computes each inner product. Even though the clustering-based approach reduces the candidate vectors, it requires a large number of clusters to effectively perform MIPS, leading to high query-to-cluster assignment overhead.

Table 6 shows that non-graph baselines have lower search accuracy. This is because the hashing-based approach determines the hash functions independently from data distributions. Besides, the quantization-based approach approximates inner products using lookup tables containing inner products between codewords; it does not compute exact inner products. Furthermore, the clustering-based approach could erroneously prune top-k vectors in performing MIPS if a query and its top-k vectors are in different clusters. As shown in Table 6, our approach maintains the same recall as the original approach. This is because we use upper bounds to prune unnecessary inner product computations for the vectors that cannot be a top-k vector. Since graph-based approaches have high search performance in performing MIPS, as shown in the previous papers [10, 30, 39, 40, 54], our approach achieves high search efficiency and accuracy.

5.10 Data Stream

As discussed in Section 3.7, our approach is well-suited to streaming scenarios in which vectors arrive continuously. This suitability stems from the approach that PCA is updated for all vectors only when the number of arrived vectors reaches 2^i for $i = 1, \ldots, \log n$. In this section, we evaluate the effectiveness of this approach. Figure 13 shows the graph construction time of ip-NSW on the MNIST dataset under the streaming setting.

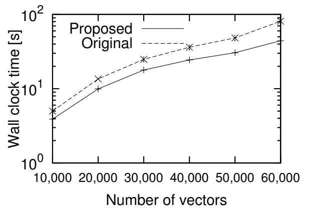


Figure 13: Graph construction time for data stream.

As shown in Figure 13, our approach constructs the graph more efficiently than the original approach in the streaming setting. In particular, it reduces the graph construction time by up to 45.4% compared to the original approach. When the number of arrived vectors reaches 2^i, our approach updates the PCA for all vectors. However, this computation can be performed efficiently by randomly sampling $\log n$ vectors, as described in Section 3.6. Moreover, since PCA updates are triggered only when the number of arrived vectors reaches 2^i, they occur infrequently as the stream grows. Consequently, the efficiency of our approach in streaming scenarios improves as the number of vectors increases.

6 Conclusions

MIPS is a fundamental task in many applications. ip-NSW and its variants are widely used for MIPS due to their high search performance. However, they incur a high computational cost in the graph construction step due to the large number of inner product computations. To address this problem, we propose an approach that leverages the upper bounds of inner products to prune unnecessary computations. Experimental results demonstrate that our approach improves construction efficiency while maintaining search accuracy across ip-NSW and its variants.

Appendix

Our approach is applicable to other non-metric measures than the inner product. For vectors $\mathbf{x}$ and $\mathbf{y}$, cosine similarity is defined as follows:

$$p(\mathbf{x}, \mathbf{y})/(\|\mathbf{x}\|\|\mathbf{y}\|). \tag{16}$$

Let $\dot{\mathbf{x}} = [x[1] - x_{\text{ave}}, \ldots, x[d] - x_{\text{ave}}]$ denote the mean-centered vector of $\mathbf{x}$. The Pearson correlation coefficient can be written as follows:

$$\frac{\sum_{i=1}^{d}(x[i]-x_{\text{ave}})(y[i]-y_{\text{ave}})}{\sqrt{\sum_{i=1}^{d}(x[i]-x_{\text{ave}})^2 \sum_{i=1}^{d}(y[i]-y_{\text{ave}})^2}} = p(\dot{\mathbf{x}}, \dot{\mathbf{y}})/(\|\dot{\mathbf{x}}\|\|\dot{\mathbf{y}}\|).$$

For binary vectors $\mathbf{x}, \mathbf{y} \in \{0, 1\}^d$, the Jaccard coefficient is expressed as follows by using the inner product:

$$p(\mathbf{x}, \mathbf{y})/(\|\mathbf{x}\|_1 + \|\mathbf{y}\|_1 - p(\mathbf{x}, \mathbf{y})),$$

where $|\mathbf{x}|_1$ denotes the l_1-norm of $\mathbf{x}$. Similarly, the Dice coefficient, another non-metric measure for binary vectors, is defined as follows:

$$2p(\mathbf{x}, \mathbf{y})/(\|\mathbf{x}\|_1 + \|\mathbf{y}\|_1).$$

Since these non-metric measures are expressed in terms of the inner product and norms, we can compute an upper bound using $\overline{p}(\mathbf{x}, \mathbf{y})$. Thus, our approach naturally extends to them.

Artifacts

We used public datasets in the experiment. We are unable to release the source code due to a pending patent application.

References

[1] Firas Abuzaid, Geet Sethi, Peter Bailis, and Matei Zaharia. 2019. To Index or Not to Index: Optimizing Exact Maximum Inner Product Search. In *ICDE*. 1250–1261.

[2] Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. 2020. ANN-Benchmarks: A benchmarking Fool for Approximate Nearest Neighbor Algorithms. *Inf. Syst.* 87 (2020).

[3] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the Xbox Recommender System Using a Euclidean Transformation for Inner-product Spaces. In *RecSys*. 257–264.

[4] Ivana Balazevic, David Steiner, Nikhil Parthasarathy, Relja Arandjelovic, and Olivier J. Hénaff. 2023. Towards In-context Scene Understanding. In *NeurIPS*.

[5] Jon Louis Bentley. 1990. K-d Trees for Semidynamic Point Sets. In *SoCG*. 187–197.

[6] Christopher M. Bishop. 2016. *Pattern Recognition and Machine Learning*. Springer.

[7] Sebastian Bruch, Franco Maria Nardini, Amir Ingber, and Edo Liberty. 2024. Bridging Dense and Sparse Maximum Inner Product Search. *ACM Trans. Inf. Syst.* 42, 6 (2024), 151:1–151:38.

[8] Xinyan Dai, Xiao Yan, Kelvin Kai Wing Ng, Jiu Liu, and James Cheng. 2020. Norm-Explicit Quantization: Improving Vector Quantization for Maximum Inner Product Search. In *AAAI*. 51–58.

[9] James W. Demmel. 2017. *Applied Numerical Linear Algebra*. Orient Blackswan.

[10] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *CoRR* abs/2401.08281 (2024).

[11] Chao Feng, Defu Lian, Xiting Wang, Zheng Liu, Xing Xie, and Enhong Chen. 2023. Reinforcement Routing on Proximity Graph for Efficient Recommendation. *ACM Trans. Inf. Syst.* 41, 1 (2023), 8:1–8:27.

[12] Yasuhiro Fujiwara, Yasutoshi Ida, Junya Arai, Mai Nishimura, and Sotetsu Iwamura. 2016. Fast Algorithm for the Lasso based L1-Graph Construction. *Proc. VLDB Endow.* 10, 3 (2016), 229–240.

[13] Yasuhiro Fujiwara, Go Irie, Shari Kuroyama, and Makoto Onizuka. 2014. Scaling Manifold Ranking Based Image Retrieval. *Proc. VLDB Endow.* 8, 4 (2014), 341–352.

[14] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Yasutoshi Ida, and Machiko Toyoda. 2015. Adaptive Message Update for Fast Affinity Propagation. In *KDD*. ACM, 309–318.

[15] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, Zhenshan Cao, Yanliang Qiao, Ting Wang, Bo Tang, and Charles Xie. 2022. Manu: A Cloud Native Vector Database Management System. *Proc. VLDB Endow.* 15, 12 (2022), 3548–3561.

[16] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *ICML*. 3887–3896.

[17] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. 47–57.

[18] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. 2012. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory Comput.* 8, 1 (2012), 321–350.

[19] Xu Huang, Defu Lian, Jin Chen, Liu Zheng, Xing Xie, and Enhong Chen. 2023. Cooperative Retriever and Ranker in Deep Recommenders. In *WWW*. 1150–1161.

[20] Ahmet Iscen, Alireza Fathi, and Cordelia Schmid. 2023. Improving Image Recognition by Retrieving from Web-Scale Image-Text Data. In *CVPR*. 19295–19304.

[21] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 3 (2021), 535–547.

[22] Norio Katayama and Shin'ichi Satoh. 1997. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *SIGMOD*. 369–380.

[23] Timo Kersten, Viktor Leis, Alfons Kemper, Thomas Neumann, Andrew Pavlo, and Peter A. Boncz. 2018. Everything You Always Wanted to Know About Compiled and Vectorized Queries But Were Afraid to Ask. *Proc. VLDB Endow.* 11, 13 (2018), 2209–2222.

[24] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient Retrieval of Recommendations in a Matrix Factorization Framework. In *CIKM*. 535–544.

[25] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: Fast and Exact Inner Product Retrieval in Recommender Systems. In *SIGMOD*. 835–850.

[26] Ping Li. 2019. Sign-Full Random Projections. In *AAAI*. 4205–4212.

[27] Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. 2021. Sparse, Dense, and Attentional Representations for Text Retrieval. *Trans. Assoc. Comput. Linguistics* 9 (2021), 329–345.

[28] Mayank Mishra, Dhiraj Madan, Gaurav Pandey, and Danish Contractor. 2022. Variational Learning for Unsupervised Knowledge Grounded Dialogs. In *IJCAI*. 4303–4309.

[29] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Ali Mousavi, Ihab F. Ilyas, Umar Farooq Minhas, Jeffrey Pound, and Theodoros Rekatsinas. 2023. High-Throughput Vector Similarity Search in Knowledge Graphs. *Proc. ACM Manag. Data* 1, 2 (2023), 197:1–197:25.

[30] Stanislav Morozov and Artem Babenko. 2018. Non-metric Similarity Graphs for Maximum Inner Product Search. In *NeurIPS*. 4726–4735.

[31] Behnam Neyshabur and Nathan Srebro. 2015. On Symmetric and Asymmetric LSHs for Inner Product Search. In *ICML*. 1926–1934.

[32] Yingshu Peng and Yi Wang. 2022. Leaf Disease Image Retrieval with Object Detection and Deep Metric Learning. *Frontiers in Plant Science* 13 (2022).

[33] Antonio Penta and Massimo Motta. 2022. *Market Effects of Sponsored Search Auctions*. Working Papers 1356. Barcelona School of Economics.

[34] Ping Qiu, Zhao Cai, Hing Kai Chan, Xiangtianrui Kong, and Ye Shi. 2023. Disentangling the Impact of Bidding Price on Click-through Rate and Product Sales in E-commerce Search Advertising. In *PACIS*. 125.

[35] Parikshit Ram and Alexander G. Gray. 2012. Maximum Inner-Product Search Using Cone Trees. In *KDD*. 931–939.

[36] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In *NIPS*. 2321–2329.

[37] Samriddhi Singla, Ahmed Eldawy, Tina Diao, Ayan Mukhopadhyay, and Elia Scudiero. 2021. Experimental Study of Big Raster and Vector Database Systems. In *ICDE*. 2243–2248.

[38] Michael Stonebraker and Andrew Pavlo. 2024. What Goes Around Comes Around... And Around.. *SIGMOD Rec.* 53, 2 (2024), 21–37.

[39] Shulong Tan, Zhaozhuo Xu, Weijie Zhao, Hongliang Fei, Zhixin Zhou, and Ping Li. 2021. Norm Adjusted Proximity Graph for Fast Inner Product Retrieval. In *KDD*. 1552–1560.

[40] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. 2019. On Efficient Retrieval of Top Similarity Vectors. In *EMNLP-IJCNLP*. 5235–5245.

[41] Christina Teflioudi and Rainer Gemulla. 2017. Exact and Approximate Maximum Inner Product Search with LEMP. *ACM Trans. Database Syst.* 42, 1 (2017), 5:1–5:49.

[42] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *SIGMOD*. 2614–2627.

[43] Yanhao Wang, Michael Mathioudakis, Yuchen Li, and Kian-Lee Tan. 2021. Minimum Coresets for Maxima Representation of Multidimensional Data. In *PODS*. 138–152.

[44] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data. *Proc. VLDB Endow.* 13, 12 (2020), 3152–3165.

[45] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. 2018. Norm-Ranging LSH for Maximum Inner Product Search. In *NeurIPS*. 2956–2965.

[46] Tiannuo Yang, Wen Hu, Wangqi Peng, Yusen Li, Jianguo Li, Gang Wang, and Xiaoguang Liu. 2024. VDTuner: Automated Performance Tuning for Vector Data Management Systems. In *ICDE*. 4357–4369.

[47] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. PASE: PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension. In *SIGMOD*. 2241–2253.

[48] Tan Yu, Yi Yang, Yi Li, Xiaodong Chen, Mingming Sun, and Ping Li. 2020. Combo-Attention Network for Baidu Video Advertising. In *KDD*. 2474–2482.

[49] Jin Zhang, Defu Lian, Haodi Zhang, Baoyun Wang, and Enhong Chen. 2023. Query-Aware Quantization for Maximum Inner Product Search. In *AAAI*. 4875–4883.

[50] Jin Zhang, Qi Liu, Defu Lian, Zheng Liu, Le Wu, and Enhong Chen. 2022. Anisotropic Additive Quantization for Fast Inner Product Search. In *AAAI*. 4354–4362.

[51] Yunan Zhang, Shige Liu, and Jianguo Wang. 2024. Are There Fundamental Limitations in Supporting Vector Data Management in Relational Databases? A Case Study of PostgreSQL. In *ICDE*. 3640–3653.

[52] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. 2020. Distributed Hierarchical GPU Parameter Server for Massive Scale Deep Learning Ads Systems. In *MLSys*.

[53] Xi Zhao, Bolong Zheng, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S. Jensen. 2023. FARGO: Fast Maximum Inner Product Search via Global Multi-Probing. *Proc. VLDB Endow.* 16, 5 (2023), 1100–1112.

[54] Zhixin Zhou, Shulong Tan, Zhaozhuo Xu, and Ping Li. 2019. Möbius Transformation for Fast Inner Product Search on Graph. In *NeurIPS*. 8216–8227.