SAH: Shifting-Aware Asymmetric Hashing for Reverse k Maximum Inner Product Search

Qiang Huang[1], **Yanhao Wang**[2*], **Anthony K. H. Tung**[1]

[1]School of Computing, National University of Singapore, Singapore
[2]School of Data Science and Engineering, East China Normal University, Shanghai, China
huangq@comp.nus.edu.sg, yhwang@dase.ecnu.edu.cn, atung@comp.nus.edu.sg

Abstract

This paper investigates a new yet challenging problem called Reverse k-Maximum Inner Product Search (RkMIPS). Given a query (item) vector, a set of item vectors, and a set of user vectors, the problem of RkMIPS aims to find a set of user vectors whose inner products with the query vector are one of the k largest among the query and item vectors. We propose the first subquadratic-time algorithm, i.e., Shifting-aware Asymmetric Hashing (SAH), to tackle the RkMIPS problem. To speed up the Maximum Inner Product Search (MIPS) on item vectors, we design a shifting-invariant asymmetric transformation and develop a novel sublinear-time Shifting-Aware Asymmetric Locality Sensitive Hashing (SA-ALSH) scheme. Furthermore, we devise a new blocking strategy based on the Cone-Tree to effectively prune user vectors (in a batch). We prove that SAH achieves a theoretical guarantee for solving the RMIPS problem. Experimental results on five real-world datasets show that SAH runs 4$\sim$8$\times$ faster than the state-of-the-art methods for RkMIPS while achieving F1-scores of over 90%. The code is available at https://github.com/HuangQiang/SAH.

Introduction

Recommender systems based on Matrix Factorization (Koren, Bell, and Volinsky 2009) (MF) and Deep Matrix Factorization (Xue et al. 2017) (DMF) models have been prevalent over the last two decades due to their precisely predictive accuracy, superior scalability, and high flexibility in various real-world scenarios. In MF and DMF models, users and items are represented as vectors in a d-dimensional Euclidean space $\mathbb{R}^d$ obtained from a user-item rating matrix. The relevance (or interestingness) of an item to a user is usually measured by the *inner product* of their representing vectors. This naturally gives rise to the Maximum Inner Product Search (MIPS) problem, which finds the vector in a set of n item vectors $\mathcal{P} \subset \mathbb{R}^d$ that has the largest inner product with a query (user) vector $\boldsymbol{q} \in \mathbb{R}^d$, i.e., $\boldsymbol{p}^* = \arg\max_{\boldsymbol{p}\in\mathcal{P}} \langle \boldsymbol{p}, \boldsymbol{q} \rangle$, as well as its extension kMIPS that finds k ($k > 1$) vectors with the largest inner products for recommending items to users. Due to its prominence in recommender systems, the kMIPS problem has attracted significant research interests, and numerous methods have been proposed to improve the search performance (Ram and Gray 2012; Koenigstein, Ram, and Shavitt 2012; Keivani, Sinha, and Ram 2018; Teflioudi and Gemulla 2017; Li et al. 2017; Abuzaid et al. 2019; Shrivastava and Li 2014; Neyshabur and Srebro 2015; Shrivastava and Li 2015; Huang et al. 2018; Yan et al. 2018; Ballard et al. 2015; Yu et al. 2017; Ding, Yu, and Hsieh 2019; Lorenzen and Pham 2020; Pham 2021; Shen et al. 2015; Guo et al. 2016; Dai et al. 2020; Xiang et al. 2021; Morozov and Babenko 2018; Tan et al. 2019; Zhou et al. 2019; Liu et al. 2020; Tan et al. 2021).

In this paper, we investigate a problem relevant to kMIPS yet less explored: *how to find the users who are possibly interested in a given item?* This problem is essential for market analysis from a *reverse* perspective, i.e., the perspective of service providers instead of users. For example, when an e-commerce service promotes a discounted product or launches a new product, a vital issue for designing an effective advertising campaign is identifying the customers who may want to buy this product. In this case, the kMIPS might not be beneficial in finding potential customers: It can be leveraged to find k user vectors having the largest inner products with the item vector. Still, these users might not be the target customers for the item if they are more interested in many other items than it. A more suitable formulation is to find the set of users for whom a query item is included in their kMIPS results, called Reverse k-Maximum Inner Product Search (RkMIPS). Formally,

Definition 1 (RkMIPS (Amagata and Hara 2021))**.** *Given an integer k ($k \geq 1$), a query (item) vector $\boldsymbol{q} \in \mathbb{R}^d$, a set of n item vectors $\mathcal{P} \subset \mathbb{R}^d$, and a set of m user vectors $\mathcal{U} \subset \mathbb{R}^d$, the RkMIPS problem finds every user vector $\boldsymbol{u} \in \mathcal{U}$ such that $\boldsymbol{q}$ belongs to the kMIPS results of $\boldsymbol{u}$ among $\mathcal{P} \cup \{\boldsymbol{q}\}$.*

Compared with the kMIPS, the problem of RkMIPS is much more challenging. The reasons are two folds. First, the sizes of its result sets vary among query vectors rather than being a fixed k. In the worst case, all m users can be included in the RkMIPS results of a query $\boldsymbol{q}$. Second, the number of items and users is typically large in real-world recommender systems. A trivial approach is performing a linear scan over all items in $\mathcal{P} \cup \{\boldsymbol{q}\}$ for each user $\boldsymbol{u} \in \mathcal{U}$ and adding $\boldsymbol{u}$ to the RkMIPS results of $\boldsymbol{q}$ once $\boldsymbol{q}$ is included in the kMIPS results of $\boldsymbol{u}$. For simplicity, we assume $m = O(n)$, i.e., n and m are of the same magnitude. This trivial approach takes $O(n^2 d)$ time, which is much higher than the time complex-

*Corresponding author

ity of brute-force kMIPS and is often computationally prohibitive, especially for large n.

Despite the importance of RkMIPS in real-world scenarios, little work has been devoted to studying this problem. Simpfer (Amagata and Hara 2021) is a pioneer work yet the only known algorithm for solving RkMIPS. Its primary idea is to efficiently solve a decision version of kMIPS for each user. Simpfer maintains a lower-bound array of the kth largest inner product of size k_{max} ($k \in \{1, 2, \cdots, k_{max}\}$) for each user $\boldsymbol{u} \in \mathcal{U}$ based on the $O(k_{max})$ items with the largest l_2-norms such that each user $\boldsymbol{u}$ can get a quick "*yes*"/"*no*" answer for any query $\boldsymbol{q}$ on whether it belongs to the kMIPS results of $\boldsymbol{u}$. Moreover, it performs a linear scan using the Cauchy-Schwarz inequality to accelerate the kMIPS on item vectors. To reduce the number of user vectors for kMIPS, it partitions users into blocks based on their l_2-norms with a fixed-size interval. Nonetheless, it is still a linear scan-based algorithm with the same worst-case time complexity of $O(n^2 d)$ as the trivial approach, and its performance degrades rapidly when n, k, or d is large.

There have been many sublinear-time hashing schemes for solving approximate kMIPS (Shrivastava and Li 2014, 2015; Neyshabur and Srebro 2015; Huang et al. 2018; Yan et al. 2018). One can leverage these schemes to speed up the kMIPS for each $\boldsymbol{u} \in \mathcal{U}$. As such, the time to perform RkMIPS can be subquadratic. However, such an adaptation might still be less efficient and effective in practice. First, as m is often larger than n, it is costly to check all users individually. Second, there is no symmetric (or asymmetric) Locality-Sensitive Hashing (LSH) for MIPS in the original space $\mathbb{R}^d$ (Shrivastava and Li 2014; Neyshabur and Srebro 2015). Existing hashing schemes develop different asymmetric transformations to convert MIPS into Nearest Neighbor Search (NNS) on angular (or Euclidean) distance, i.e., an item transformation $\boldsymbol{I} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ and a user transformation $\boldsymbol{U} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ on the item and user vectors, respectively, where $d' > d$. Unfortunately, these transformations add a large constant in angular (and Euclidean) distance, leading to a significant *distortion error* for the subsequent NNS, i.e., the relative angular (and Euclidean) distance of any $\boldsymbol{I}(\boldsymbol{p})$ and $\boldsymbol{U}(\boldsymbol{u})$ will be much smaller than that in the original space. As a result, any $\boldsymbol{I}(\boldsymbol{p})$ can be the NNS result of $\boldsymbol{U}(\boldsymbol{u})$ even though their inner product $\langle \boldsymbol{p}, \boldsymbol{u} \rangle$ is very small. Thus, the kMIPS results can be arbitrarily bad.

In addition, the RkMIPS problem shares a similar concept with the reverse top-k query (Vlachou et al. 2010, 2011, 2013), since both problems aim to find a set of users such that the query item is one of their top-k results. The methods for reverse top-k queries, however, might not be suitable for solving RkMIPS as they usually assume that the dimensionality d is low (Vlachou et al. 2013), i.e., $d < 10$, whereas d is often dozens to hundreds in recommender systems. Another problem related to RkMIPS is the Reverse k-Nearest Neighbor Search (RkNNS) (Korn and Muthukrishnan 2000; Yang and Lin 2001; Singh, Ferhatosmanoglu, and Tosun 2003; Tao, Papadias, and Lian 2004; Achtert et al. 2006; Arthur and Oudot 2010). Nevertheless, like the case of reverse top-k queries, most existing RkNNS methods are also customized for low-dimensional data.

Our Contributions. In this paper, we propose the first subquadratic-time algorithm called Shifting-aware Asymmetric Hashing (SAH) to tackle the problem of RkMIPS in high-dimensional spaces. To accelerate the kMIPS on item vectors, we develop a provable, sublinear-time scheme called Shifting-Aware Asymmetric Locality-Sensitive Hashing (SA-ALSH) together with a novel shifting-invariant asymmetric transformation to reduce the distortion error significantly. Furthermore, we devise a novel blocking strategy for user vectors based on the Cone-Tree (Ram and Gray 2012). Using the cone structure, we derive two tight upper bounds that can effectively prune user vectors (in a batch). SAH also inherits the basic idea of Simpfer (Amagata and Hara 2021) to leverage the lower bounds for users to obtain a quick "*yes*"/"*no*" decision for the kMIPS. We prove that SAH achieves a theoretical guarantee for solving RkMIPS when $k = 1$ in subquadratic time and space. In the experiments, we systematically compare SAH with a state-of-the-art kMIPS method H2-ALSH (Huang et al. 2018) as well as the only known RkMIPS method Simpfer (Amagata and Hara 2021). Extensive results over five real-world datasets demonstrate that SAH runs 4$\sim$8$\times$ faster than them for RkMIPS while achieving F1-scores of over 90%.

Background

Before presenting SAH for solving RkMIPS, we first introduce the background of Locality-Sensitive Hashing (LSH) and Asymmetric Locality-Sensitive Hashing (ALSH).

Locality-Sensitive Hashing

LSH schemes are one of the most prevalent methods for solving high-dimensional NNS (Indyk and Motwani 1998; Charikar 2002; Datar et al. 2004; Andoni and Indyk 2006; Har-Peled, Indyk, and Motwani 2012; Andoni et al. 2015; Huang et al. 2015; Lei et al. 2019, 2020). Given a hash function h, we say two vectors $\boldsymbol{p}$ and $\boldsymbol{u}$ collide in the same bucket if $h(\boldsymbol{p}) = h(\boldsymbol{u})$. Let $Dist(\boldsymbol{p}, \boldsymbol{u})$ be a distance function of any two vectors $\boldsymbol{p}$ and $\boldsymbol{u}$. Formally,

Definition 2 (LSH Family (Indyk and Motwani 1998))**.** *Given a search radius R ($R > 0$) and an approximation ratio c, a hash family $\mathcal{H}$ is called (R, cR, p_1, p_2)-sensitive to $Dist(\cdot,\cdot)$ if, for any $\boldsymbol{p}, \boldsymbol{u} \in \mathbb{R}^d$, it satisfies:*

- *If $Dist(\boldsymbol{p}, \boldsymbol{u}) \leq R$, then $\Pr_{h \in \mathcal{H}}[h((\boldsymbol{p}) = h(\boldsymbol{u})] \geq p_1$;*
- *If $Dist(\boldsymbol{p}, \boldsymbol{u}) \geq cR$, then $\Pr_{h \in \mathcal{H}}[h(\boldsymbol{p}) = h(\boldsymbol{u})] \leq p_2$.*

An LSH family is valid for NNS only when $c > 1$ and $p_1 > p_2$. With an (R, cR, p_1, p_2)-sensitive hash family, LSH schemes can deal with the NNS in sublinear time and subquadratic space.

Theorem 1 (Indyk and Motwani 1998)**.** *Given a family $\mathcal{H}$ of (R, cR, p_1, p_2)-sensitive hash functions, one can construct a data structure that finds an item vector $\boldsymbol{p} \in \mathcal{P}$ such that $Dist(\boldsymbol{p}, \boldsymbol{u}) \leq c \cdot Dist(\boldsymbol{p}^*, \boldsymbol{u})$ in $O(n^{1+\rho})$ space and $O(dn^\rho \log_{1/p_2} n)$ query time, where $\rho = \ln p_1 / \ln p_2$ and $\boldsymbol{p}^* = \arg\min_{\boldsymbol{p} \in \mathcal{P}} Dist(\boldsymbol{p}, \boldsymbol{u})$.*

SimHash is a classic LSH scheme proposed by Charikar (2002) for solving NNS on angular distance. The angular distance is computed as $\theta(\boldsymbol{p},\boldsymbol{u}) = \arccos(\frac{\langle \boldsymbol{p},\boldsymbol{u}\rangle}{\|\boldsymbol{p}\|\|\boldsymbol{u}\|})$ for any $\boldsymbol{p},\boldsymbol{u} \in \mathbb{R}^d$. Its LSH function is called Sign Random Projection (SRP), i.e.,

$$h_{srp}(\boldsymbol{p}) = sgn(\langle \boldsymbol{a},\boldsymbol{p}\rangle), \tag{1}$$

where $\boldsymbol{a}$ is a d-dimensional vector with each entry drawn i.i.d. from the standard normal distribution $\mathcal{N}(0,1)$; $sgn(\cdot)$ and $\langle\cdot,\cdot\rangle$ denote the sign function and the inner product computation, respectively. Let $\delta = \theta(\boldsymbol{p},\boldsymbol{u})$ be the angular distance of any $\boldsymbol{p}$ and $\boldsymbol{u}$. The collision probability is:

$$p(\delta) = \Pr[h_{srp}(\boldsymbol{p}) = h_{srp}(\boldsymbol{u})] = 1 - \tfrac{\delta}{\pi}. \tag{2}$$

Asymmetric LSH

Since existing LSH schemes for NNS on Euclidean or angular distance are not directly applicable to MIPS, they usually perform asymmetric transformations to reduce MIPS to NNS, known as *Asymmetric LSH* (ALSH).

Definition 3 (ALSH Family (Shrivastava and Li 2014))**.** *Given an inner product threshold S_0 ($S_0 > 0$) and an approximation ratio c, a hash family $\mathcal{H}$, along with two vector transformations, i.e., $\boldsymbol{I} : \mathbb{R}^d \to \mathbb{R}^{d'}$ (Item transformation) and $\boldsymbol{U} : \mathbb{R}^d \to \mathbb{R}^{d'}$ (User transformation), is called $(S_0, \frac{S_0}{c}, p_1, p_2)$-sensitive to the inner product $\langle\cdot,\cdot\rangle$ if, for any $\boldsymbol{p},\boldsymbol{u} \in \mathbb{R}^d$, it satisfies:*

- *If $\langle \boldsymbol{p},\boldsymbol{u}\rangle \geq S_0$, then $\Pr_{h\in\mathcal{H}}[h(\boldsymbol{I}(\boldsymbol{p})) = h(\boldsymbol{U}(\boldsymbol{u}))] \geq p_1$;*
- *If $\langle \boldsymbol{p},\boldsymbol{u}\rangle \leq \frac{S_0}{c}$, then $\Pr_{h\in\mathcal{H}}[h(\boldsymbol{I}(\boldsymbol{p})) = h(\boldsymbol{U}(\boldsymbol{u}))] \leq p_2$.*

The ALSH family is valid for MIPS only when $c > 1$ and $p_1 > p_2$. The transformation $\boldsymbol{I}$ ($\boldsymbol{U}$) is only applied to item vectors $\boldsymbol{p} \in \mathcal{P}$ (user vectors $\boldsymbol{u} \in \mathcal{U}$). The transformations are asymmetric if $\boldsymbol{I}(\boldsymbol{x}) \neq \boldsymbol{U}(\boldsymbol{x}) \neq \boldsymbol{x}$ for any $\boldsymbol{x} \in \mathbb{R}^d$.

H2-ALSH (Huang et al. 2018) is a state-of-the-art ALSH scheme for MIPS. Let $\boldsymbol{p} = [p_1,\cdots,p_d]$ and $\boldsymbol{u} = [u_1,\cdots, u_d]$. It designs a Query Normalized First (QNF) transformation to convert MIPS into NNS on Euclidean distance, which is defined as follows:

$$\boldsymbol{I}(\boldsymbol{p}) = [p_1,\cdots,p_d; \sqrt{M^2 - \|\boldsymbol{p}\|^2}], \tag{3}$$

$$\boldsymbol{U}(\boldsymbol{u}) = [\lambda u_1,\cdots,\lambda u_d; 0],\ \text{where}\ \lambda = M/\|\boldsymbol{u}\|, \tag{4}$$

where M is the maximum among the l_2-norms of all items in $\mathcal{P}$, i.e., $M = \max_{\boldsymbol{p}\in\mathcal{P}} \|\boldsymbol{p}\|$, and $[\cdot;\cdot]$ denotes the concatenation of two vectors. Based on Equations 3 and 4, we have

$$\|\boldsymbol{I}(\boldsymbol{p}) - \boldsymbol{U}(\boldsymbol{u})\|^2 = 2M \cdot (M - \tfrac{\langle \boldsymbol{p},\boldsymbol{u}\rangle}{\|\boldsymbol{u}\|}). \tag{5}$$

Let θ be the angle of $\boldsymbol{p}$ and $\boldsymbol{u}$. As $\langle \boldsymbol{p},\boldsymbol{u}\rangle/\|\boldsymbol{u}\| = \|\boldsymbol{p}\|\cos\theta \leq \|\boldsymbol{p}\| \leq M$, we have $M - \langle \boldsymbol{p},\boldsymbol{u}\rangle/\|\boldsymbol{u}\| \geq 0$. Since M and $\|\boldsymbol{u}\|$ are fixed for a given dataset, the MIPS in $\mathbb{R}^d$ can be converted into the NNS on Euclidean distance in $\mathbb{R}^{d+1}$. Unfortunately, the angle θ of any $\boldsymbol{p}$ and $\boldsymbol{u}$ in high-dimensional spaces is often close to $\pi/2$, incurring a very small $\cos\theta$. Thus, we often have $\|\boldsymbol{p}\|\cos\theta \ll M$. In the worst case, $\max_{\boldsymbol{p}\in\mathcal{P}} \|\boldsymbol{I}(\boldsymbol{p}) - \boldsymbol{U}(\boldsymbol{u})\| / \min_{\boldsymbol{p}\in\mathcal{P}} \|\boldsymbol{I}(\boldsymbol{p}) - \boldsymbol{U}(\boldsymbol{u})\| \to 1$. Suppose that this ratio is less than 2, and we set up $c = 2$ for approximate NNS, which is a typical setting for LSH schemes (Tao et al. 2009; Gan et al. 2012; Huang et al. 2015). Then, any $\boldsymbol{I}(\boldsymbol{p})$ can be the NNS result of $\boldsymbol{U}(\boldsymbol{u})$ even if $\langle \boldsymbol{p},\boldsymbol{u}\rangle$ is small, which means that the MIPS result of H2-ALSH for $\boldsymbol{u}$ can be arbitrarily bad.

H2-ALSH develops a homocentric hypersphere partition strategy to split the item vectors into different blocks with bounded l_2-norms such that the item vectors with smaller l_2-norms correspond to a smaller M. This strategy can alleviate the distortion error but still cannot remedy the issue caused by the angle close to $\pi/2$.

The SAH Algorithm

In this section, we propose the SAH algorithm for performing RkMIPS on high-dimensional data. To be concise, we focus on its intuition and procedure. Omitted proofs can be found in the full version (Huang, Wang, and Tung 2022).

Shifting-invariant Asymmetric Transformation

We first introduce a Shifting-invariant Asymmetric Transformation (SAT) that converts the MIPS in $\mathbb{R}^d$ into the NNS on angular distance in $\mathbb{R}^{d+1}$. Let $\boldsymbol{c}$ be the centroid of the item set $\mathcal{P}$, i.e., $\boldsymbol{c} = [c_1,\cdots,c_d] = \frac{1}{n}\sum_{\boldsymbol{p}\in\mathcal{P}} \boldsymbol{p}$. Suppose that R is the radius of the smallest ball centered at $\boldsymbol{c}$ enclosing all $\boldsymbol{p} \in \mathcal{P}$, i.e., $R = \max_{\boldsymbol{p}\in\mathcal{P}} \|\boldsymbol{p} - \boldsymbol{c}\|$. Given any item vector $\boldsymbol{p} = [p_1,\cdots,p_d]$ and user vector $\boldsymbol{u} = [u_1,\cdots,u_d]$, the item transformation $\boldsymbol{I} : \mathbb{R}^d \to \mathbb{R}^{d+1}$ and user transformation $\boldsymbol{U} : \mathbb{R}^d \to \mathbb{R}^{d+1}$ of SAT are:

$$\boldsymbol{I}(\boldsymbol{p},\boldsymbol{c}) = [p_1 - c_1,\cdots,p_d - c_d; \sqrt{R^2 - \|\boldsymbol{p} - \boldsymbol{c}\|^2}], \tag{6}$$

$$\boldsymbol{U}(\boldsymbol{u}) = [\lambda u_1,\cdots,\lambda u_d; 0], \text{where}\ \lambda = R/\|\boldsymbol{u}\|. \tag{7}$$

As $\|\boldsymbol{I}(\boldsymbol{p},\boldsymbol{c})\| = \|\boldsymbol{U}(\boldsymbol{u})\| = R$, SAT maps each item vector $\boldsymbol{p} \in \mathcal{P}$ and user vector $\boldsymbol{u} \in \mathcal{U}$ in $\mathbb{R}^d$ to the hypersphere $\mathbb{S}^d$ of radius R. Based on Equations 6 and 7,

$$\frac{\langle \boldsymbol{I}(\boldsymbol{p},\boldsymbol{c}),\boldsymbol{U}(\boldsymbol{u})\rangle}{\|\boldsymbol{I}(\boldsymbol{p},\boldsymbol{c})\| \cdot \|\boldsymbol{U}(\boldsymbol{u})\|} = \frac{\langle \boldsymbol{p} - \boldsymbol{c},\boldsymbol{u}\rangle}{R \cdot \|\boldsymbol{u}\|}. \tag{8}$$

The intuition of SAT comes from the fact that the MIPS result of any vector $\boldsymbol{u}$ is *shift-invariant*, i.e., it is always the same no matter where the item vectors are shifted.

Fact 1. *Given a set of vectors $\mathcal{P}$, the MIPS result of any vector $\boldsymbol{u}$ is invariant whether all vectors in $\mathcal{P}$ are shifted by $\boldsymbol{c}$, i.e.,* $\arg\max_{\boldsymbol{p}\in\mathcal{P}}\langle \boldsymbol{p},\boldsymbol{u}\rangle = \arg\max_{\boldsymbol{p}\in\mathcal{P}}\langle \boldsymbol{p} - \boldsymbol{c},\boldsymbol{u}\rangle$.

According to Fact 1, as the term $R \cdot \|\boldsymbol{u}\|$ in Equation 8 is the same for any $\boldsymbol{p} \in \mathcal{P}$, we have $\arg\max_{\boldsymbol{p}\in\mathcal{P}}\langle \boldsymbol{p},\boldsymbol{u}\rangle = \arg\min_{\boldsymbol{p}\in\mathcal{P}} \arccos(\frac{\langle \boldsymbol{I}(\boldsymbol{p},\boldsymbol{c}),\boldsymbol{U}(\boldsymbol{u})\rangle}{\|\boldsymbol{I}(\boldsymbol{p},\boldsymbol{c})\|\cdot\|\boldsymbol{U}(\boldsymbol{u})\|})$. Thus, SAT converts the MIPS in $\mathbb{R}^d$ into the NNS on angular distance in $\mathbb{R}^{d+1}$. Compared with the QNF transformation, SAT introduces a shifting operation to the item vectors, which typically reduces the maximum l_2-norm among item vectors after the item transformation. Moreover, according to Equation 8, the distortion of SAT only comes from the ratio $R/\|\boldsymbol{p} - \boldsymbol{c}\|$, which decreases the error caused by a small $\cos\theta$. Therefore, SAT can significantly reduce the distortion error in practice, as will be validated in our experiments.

Shifting-Aware Asymmetric LSH

ALSH Family. We first describe the hash family $\mathcal{H}_{sa}$ of SA-ALSH for solving MIPS. Let $h_{srp}(\cdot)$ be the SRP-LSH

Algorithm 1: SA-ALSH Indexing

Input: A set of n item vectors $\mathcal{P}$, an interval ratio $b \in (0, 1)$, number of hash tables $K \in \mathbb{Z}^+$;

1 Compute $\|\boldsymbol{p}\|$ for each item $\boldsymbol{p} \in \mathcal{P}$ and sort $\mathcal{P}$ in descending order of $\|\boldsymbol{p}\|$;
2 $j = 0$; $i = 0$;
3 **while** $i < n$ **do**
4     $j \leftarrow j + 1$; $M_j \leftarrow \|\boldsymbol{p}_i\|$; $\mathcal{S}_j \leftarrow \emptyset$;
5   **while** $i < n$ *and* $\|\boldsymbol{p}_i\| > bM_j$ **do**
6         $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup \{\boldsymbol{p}_i\}$; $i \leftarrow i + 1$;
7     $\boldsymbol{c}_j = \frac{1}{|\mathcal{S}_j|} \sum_{\boldsymbol{p} \in \mathcal{S}_j} \boldsymbol{p}$; $R_j = \max_{\boldsymbol{p} \in \mathcal{S}_j} \|\boldsymbol{p} - \boldsymbol{c}_j\|$;
8     $\mathcal{I}_j \leftarrow \emptyset$;
9   **foreach** *item* $\boldsymbol{p} \in \mathcal{S}_j$ **do**
10         $\boldsymbol{I}(\boldsymbol{p}, \boldsymbol{c}_j) \leftarrow [p_1 - c_{j_1}, \cdots, p_d - c_{j_d}; \sqrt{R_j^2 - \|\boldsymbol{p} - \boldsymbol{c}_j\|^2}]$;
11         $\mathcal{I}_j \leftarrow \mathcal{I}_j \cup \{\boldsymbol{I}(\boldsymbol{p}, \boldsymbol{c}_j)\}$;
12   Build the K hash tables for $\mathcal{I}_j$ using SimHash;
13 $t \leftarrow j$;

Algorithm 2: SA-ALSH

Input: User vector $\boldsymbol{u}$, query vector $\boldsymbol{q}$, $k \in \mathbb{Z}^+$;

1 $\mathcal{C} = \emptyset$; $\varphi = -\infty$;
2 **for** $j = 1$ *to* t **do**
3     $\mu_j = M_j \cdot \|\boldsymbol{u}\|$;
4   **if** $\varphi > \mu_j$ **then return** *yes*;
5     $\boldsymbol{U}(\boldsymbol{u}) = [R_j u_1/\|\boldsymbol{u}\|, \cdots, R_j u_d/\|\boldsymbol{u}\|; 0]$;
6     $\mathcal{C} \leftarrow \mathcal{C} \cup \text{SimHash}(\mathcal{S}_j, \boldsymbol{U}(\boldsymbol{u}))$;
7     $\varphi \leftarrow$ the kth largest inner product among the item vectors in $\mathcal{C}$ with $\boldsymbol{u}$;
8   **if** $\langle \boldsymbol{u}, \boldsymbol{q} \rangle < \varphi$ **then return** *no*;
9 **return** *yes*;

function in Equation 1. Given the centroid $\boldsymbol{c}$ of item vectors, $\boldsymbol{I}(\boldsymbol{p}, \boldsymbol{c})$ and $\boldsymbol{U}(\boldsymbol{u})$ in Equations 6 and 7, respectively, the hash family $\mathcal{H}_{sa}$ of hash functions h_{sa} is:

$$h_{sa}(\boldsymbol{x}) = \begin{cases} h_{srp}(\boldsymbol{I}(\boldsymbol{x}, \boldsymbol{c})), & \text{if } \boldsymbol{x} \in \mathcal{P}, \\ h_{srp}(\boldsymbol{U}(\boldsymbol{x})), & \text{if } \boldsymbol{x} \in \mathcal{U}. \end{cases} \tag{9}$$

According to Equation 8, the collision probability of $h_{sa}(\cdot)$ for certain $\boldsymbol{p}, \boldsymbol{u}$ is computed as follows:

$$\Pr[h_{sa}(\boldsymbol{p}) = h_{sa}(\boldsymbol{u})] = p(\arccos(\tfrac{\langle \boldsymbol{p} - \boldsymbol{c}, \boldsymbol{u} \rangle}{R \cdot \|\boldsymbol{u}\|})), \tag{10}$$

where $p(\cdot)$ is the collision probability of the SRP-LSH family as given in Equation 2. Let $p_1 = p(\arccos(\frac{S_0}{R \cdot \|\boldsymbol{u}\|}))$ and $p_2 = p(\arccos(\frac{S_0}{cR \cdot \|\boldsymbol{u}\|}))$. With $\boldsymbol{I}(\boldsymbol{p}, \boldsymbol{c})$ and $\boldsymbol{U}(\boldsymbol{u})$, we show that $\mathcal{H}_{sa}$ is an ALSH family for $\langle \cdot, \cdot \rangle$.

Lemma 1. *Given an inner product threshold S_0 ($S_0 > 0$), an approximation ratio c ($c > 1$), and an $(\arccos(\frac{S_0}{R \cdot \|\boldsymbol{u}\|}), \arccos(\frac{S_0}{cR \cdot \|\boldsymbol{u}\|}), p_1, p_2)$-sensitive SRP hash family $\mathcal{H}$ for the NNS on angular distance, the hash family $\mathcal{H}_{sa}$ of hash functions $h_{sa}(\cdot)$ is $(S_0, \frac{S_0}{c}, p_1, p_2)$-sensitive to $\langle \cdot, \cdot \rangle$.*

We now present SA-ALSH for performing MIPS with the newly designed ALSH family $\mathcal{H}_{sa}$. With the insight that the item vectors with larger l_2-norms belong to the MIPS results of user vectors with higher probability (Huang et al. 2018; Yan et al. 2018; Liu et al. 2020), we introduce a data-dependent partitioning strategy to build the index separately for different norm-based partitions of item vectors.

Indexing Phase. The indexing phase of SA-ALSH is depicted in Algorithm 1. Given a set of item vectors $\mathcal{P}$, we first compute the l_2-norm $\|\boldsymbol{p}\|$ of each $\boldsymbol{p} \in \mathcal{P}$ and sort them in descending order (Line 1). Let b be the interval ratio ($0 < b < 1$). We partition $\mathcal{P}$ into t disjoint subsets $\{\mathcal{S}_j\}_{j=1}^t$ such that $bM_j < \|\boldsymbol{p}\| \leq M_j$ for each $\boldsymbol{p} \in \mathcal{S}_j$, where $M_j = \max_{\boldsymbol{p} \in \mathcal{S}_j} \|\boldsymbol{p}\|$ (Lines 4–6). For each $\mathcal{S}_j$, we first compute its centroid $\boldsymbol{c}_j$ and radius R_j, i.e., $\boldsymbol{c}_j = \frac{1}{|\mathcal{S}_j|} \sum_{\boldsymbol{p} \in \mathcal{S}_j} \boldsymbol{p}$ and $R_j = \max_{\boldsymbol{p} \in \mathcal{S}_j} \|\boldsymbol{p} - \boldsymbol{c}_j\|$ (Line 7). According to Equation 6, we apply $\boldsymbol{I} : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$ to convert each $\boldsymbol{p} \in \mathcal{S}_j$ into $\boldsymbol{I}(\boldsymbol{p}, \boldsymbol{c}_j)$ (Lines 8–11). Finally, we generate a set of SRP-LSH functions and apply SimHash to build the index for all $\boldsymbol{I}(\boldsymbol{p}, \boldsymbol{c}_j)$'s (Line 12).

Note that $M_1, \cdots, M_t$ are sorted in descending order of l_2-norms and thus can be leveraged to estimate the upper bound for pruning item vectors in a batch. The number of partitions t is *automatically* determined by the l_2-norms of item vectors and the interval ratio b. As the item vectors are partitioned into subsets and based on different centroids for the shifting-invariant asymmetric transformation, it is called the Shifting-Aware ALSH (SA-ALSH).

Query Phase. The query phase of SA-ALSH is shown in Algorithm 2. Given a user vector $\boldsymbol{u}$ and a query vector $\boldsymbol{q}$, SA-ALSH aims to identify a set of candidates $\mathcal{C}$ from $\mathcal{S}_1$ to $\mathcal{S}_t$ to determine whether or not the query $\boldsymbol{q}$ is included in the kMIPS results of $\boldsymbol{u}$. Let φ be the largest inner product of $\boldsymbol{p} \in \mathcal{P}$ and $\boldsymbol{u}$ found so far. We initialize $\mathcal{C}$ as an empty set (Line 1). For each $\mathcal{S}_j$, we can determine an upper bound for the item vectors based on M_j and $\|\boldsymbol{u}\|$, i.e., $\mu_j = M_j \cdot \|\boldsymbol{u}\|$ (Line 3). It is because based on the Cauchy-Schwarz inequality, as $bM_j < \|\boldsymbol{p}\| \leq M_j$ for any $\boldsymbol{p} \in \mathcal{S}_j$, we have $\langle \boldsymbol{p}, \boldsymbol{u} \rangle \leq \|\boldsymbol{p}\| \cdot \|\boldsymbol{u}\| \leq M_j \cdot \|\boldsymbol{u}\|$. If $\varphi > \mu_j$, we can prune $\mathcal{S}_j$ and the rest partitions $\{\mathcal{S}_{j+1}, \cdots, \mathcal{S}_t\}$ in a batch and return "yes" (Line 4) since $M_j > M_{j+1}$ and $\mu_j > \mu_{j+1}$, $\boldsymbol{q}$ belongs to the kMIPS results of $\boldsymbol{u}$; otherwise, we apply $\boldsymbol{U} : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$ to convert $\boldsymbol{u}$ into $\boldsymbol{U}(\boldsymbol{u})$ (Line 5), call SimHash to perform NNS on $\mathcal{S}_j$ (Line 6), and update φ (Line 7). If $\langle \boldsymbol{u}, \boldsymbol{q} \rangle < \varphi$, which means that $\boldsymbol{q}$ is not included in the kMIPS results of $\boldsymbol{u}$, we can safely stop and return "no" (Line 8). Finally, we return "yes" as $\boldsymbol{q}$ is kept in the kMIPS results of $\boldsymbol{u}$ (Line 9).

As SA-ALSH first performs kMIPS on the item vectors with the largest l_2-norms, which most probably contain the kMIPS results of $\boldsymbol{u}$, the search process can be stopped early and effectively avoid evaluating a large number of false positives. Based on Theorem 1 and Lemma 1, we demonstrate that SA-ALSH achieves a theoretical guarantee for solving MIPS in sublinear time and subquadratic space.

Theorem 2. *Given a hash family $\mathcal{H}_{SA}$ of hash functions $h_{sa}(\cdot)$ as defined by Equation 9, SA-ALSH is a data structure that finds an item vector $\boldsymbol{p} \in \mathcal{P}$ for any user vector $\boldsymbol{u} \in \mathbb{R}^d$ such that $\langle \boldsymbol{p}, \boldsymbol{u} \rangle \leq \langle \boldsymbol{p}^*, \boldsymbol{u} \rangle / c$ with constant probability in $O(dn^\rho \log_{1/p_2} n)$ time and $O(n^{1+\rho})$ space, where $\rho = \ln(1/p_1)/\ln(1/p_2)$ and $\boldsymbol{p}^* = \arg\max_{\boldsymbol{p} \in \mathcal{P}} \langle \boldsymbol{p}, \boldsymbol{u} \rangle$.*

Cone-Tree Blocking

Suppose that $\theta_{\boldsymbol{p},\boldsymbol{u}}$ is the angle of an item vector $\boldsymbol{p}$ and a user vector $\boldsymbol{u}$. As $\langle \boldsymbol{p}, \boldsymbol{u} \rangle = \|\boldsymbol{p}\| \cdot \|\boldsymbol{u}\| \cos \theta_{\boldsymbol{p},\boldsymbol{u}}$, we have another fact that the l_2-norm $\|\boldsymbol{u}\|$ of $\boldsymbol{u}$ does not affect its MIPS result. Formally,

Fact 2. *Given a set of item vectors $\mathcal{P}$, the MIPS result of any user vector $\boldsymbol{u}$ is independent of its l_2-norm $\|\boldsymbol{u}\|$, i.e., $\arg\max_{\boldsymbol{p} \in \mathcal{P}} \langle \boldsymbol{p}, \boldsymbol{u} \rangle = \arg\max_{\boldsymbol{p} \in \mathcal{P}} \|\boldsymbol{p}\| \cos \theta_{\boldsymbol{p},\boldsymbol{u}}$.*

Based on Fact 2, we simply assume that all user vectors are unit vectors, i.e., $\|\boldsymbol{u}\| = 1$ for every $\boldsymbol{u} \in \mathcal{U}$. Fact 2 also implies that the MIPS result is only affected by the direction of $\boldsymbol{u}$. Motivated by this, we design a new blocking strategy for user vectors based on Cone-Tree (Ram and Gray 2012).

Cone-Tree Structure. We first review the basic structure of Cone-Tree (Ram and Gray 2012). The Cone-Tree is a binary space partition tree. Each node N consists of a subset of user vectors, i.e., $N.S \subset \mathcal{U}$. Let $|N|$ be the number of user vectors in a node N, i.e., $|N| = |N.S|$. Any node N and its two children $N.lc$ and $N.rc$ satisfy two properties: $|N.lc| + |N.rc| = |N|$ and $N.lc.S \cap N.rc.S = \emptyset$. Specifically, $N.S = \mathcal{U}$ if N is the root of the Cone-Tree. Each node maintains a cone structure for its user vectors, i.e., the center $N.\boldsymbol{c} = \frac{1}{|N|} \sum_{\boldsymbol{u} \in N.S} \boldsymbol{u}$ and the maximum angle $N.\omega = \max_{\boldsymbol{u} \in N.S} \arccos(\frac{\langle \boldsymbol{u}, N.\boldsymbol{c} \rangle}{\|\boldsymbol{u}\| \cdot \|N.\boldsymbol{c}\|})$.

Upper Bounds for RkMIPS. Based on the cone structure, we present an upper bound to prune a group of user vectors for RkMIPS. Let ϕ be the angle of $N.\boldsymbol{c}$ and a query $\boldsymbol{q}$.

Lemma 2 (Node-Level Upper Bound)**.** *Let the function $\{\theta\}_+ = \max\{\theta, 0\}$. Given a query $\boldsymbol{q}$ and a node N that contains a subset of user vectors $N.S$ centered at $N.\boldsymbol{c}$ with the maximum angle $N.\omega$, the maximum possible $\langle \boldsymbol{u}, \boldsymbol{q} \rangle$ of any user vector $\boldsymbol{u} \in N.S$ and $\boldsymbol{q}$ is bounded as follows:*

$$\max_{\boldsymbol{u} \in N.S} \langle \boldsymbol{u}, \boldsymbol{q} \rangle \leq \|\boldsymbol{q}\| \cos(\{\phi - N.\omega\}_+). \tag{11}$$

The node-level upper bound can prune all user vectors within a node in a batch, whereas it might not be tight for each user vector. To perform *vector-level pruning*, we further maintain cone structures for the user vectors in each leaf node N. As such, we quickly get an upper bound for each $\boldsymbol{u} \in N.S$. The advantage is that all cones share the same center $N.\boldsymbol{c}$ and only an angle $\theta_{\boldsymbol{u}}$ of each $\boldsymbol{u}$ and $N.\boldsymbol{c}$ is kept.

Lemma 3 (Vector-Level Upper Bound)**.** *Given a query $\boldsymbol{q}$ and a leaf node N that maintains the angle $\theta_{\boldsymbol{u}}$ of each user vector $\boldsymbol{u} \in N.S$ and the center $N.\boldsymbol{c}$, the maximum possible $\langle \boldsymbol{u}, \boldsymbol{q} \rangle$ of each $\boldsymbol{u} \in N.S$ and $\boldsymbol{q}$ is bounded as follows:*

$$\langle \boldsymbol{u}, \boldsymbol{q} \rangle \leq \|\boldsymbol{q}\| \cos(|\phi - \theta_{\boldsymbol{u}}|). \tag{12}$$

Compared with the Cauchy-Schwarz inequality $\langle \boldsymbol{u}, \boldsymbol{q} \rangle \leq \|\boldsymbol{u}\| \cdot \|\boldsymbol{q}\| = \|\boldsymbol{q}\|$ used in Simpfer (Amagata and Hara 2021), since $\cos(|\phi - \theta_{\boldsymbol{u}}|) \leq 1$, Equation 12 is strictly tighter.

Algorithm 3: Cone-Tree Construction

Input: Subset $S \subseteq \mathcal{U}$, maximum leaf size N_0;
1 $N.S \leftarrow S$; $N.\boldsymbol{c} \leftarrow \frac{1}{|N|} \sum_{\boldsymbol{u} \in N.S} \boldsymbol{u}$;
2 $N.\omega \leftarrow \max_{\boldsymbol{u} \in N.S} \arccos(\frac{\langle \boldsymbol{u}, N.\boldsymbol{c} \rangle}{\|\boldsymbol{u}\| \cdot \|N.\boldsymbol{c}\|})$;
3 **if** $|N| > N_0$ **then** ▷ internal node
4   Select a point $\boldsymbol{v} \in N.S$ uniformly at random;
5     $\boldsymbol{u}_l \leftarrow \arg\min_{\boldsymbol{u} \in S} \langle \boldsymbol{u}, \boldsymbol{v} \rangle$;
6     $\boldsymbol{u}_r \leftarrow \arg\min_{\boldsymbol{u} \in S} \langle \boldsymbol{u}, \boldsymbol{u}_l \rangle$;
7     $S_l \leftarrow \{\boldsymbol{u} \in S \mid \cos \theta_{\boldsymbol{u},\boldsymbol{u}_l} \geq \cos \theta_{\boldsymbol{u},\boldsymbol{u}_r}\}$;
8     $S_r \leftarrow S \setminus S_l$;
9     $N.lc \leftarrow$ Cone-Tree Construction(S_l, N_0);
10     $N.rc \leftarrow$ Cone-Tree Construction(S_r, N_0);
11   **return** N;
12 **else** ▷ leaf node
13   **foreach** $\boldsymbol{u} \in N.S$ **do**
14         $\theta_{\boldsymbol{u}} \leftarrow \arccos(\frac{\langle \boldsymbol{u}, N.\boldsymbol{c} \rangle}{\|\boldsymbol{u}\| \cdot \|N.\boldsymbol{c}\|})$;
15   **return** N;

Cone-Tree Blocking. We now present our blocking strategy based on Cone-Tree to partition the set of user vectors $\mathcal{U}$. We first show the pseudocode of Cone-Tree construction in Algorithm 3. The center $N.\boldsymbol{c}$ and the maximum angle $N.\omega$ are maintained within each node N (Lines 1 & 2). For an internal node N, the splitting procedure is performed with three steps: (1) we select a random point $\boldsymbol{v} \in N.S$ (Line 4); (2) we find the point $\boldsymbol{u}_l$ with the minimum inner product of $\boldsymbol{v}$, i.e., $\boldsymbol{u}_l = \arg\min_{\boldsymbol{u} \in S} \langle \boldsymbol{u}, \boldsymbol{v} \rangle$ (Line 5); (3) we find another point $\boldsymbol{u}_r$ with the minimum inner product of $\boldsymbol{u}_l$, i.e., $\boldsymbol{u}_r = \arg\min_{\boldsymbol{u} \in S} \langle \boldsymbol{u}, \boldsymbol{u}_l \rangle$ (Line 6). As we assume that all user vectors are unit vectors, the point $\boldsymbol{u}_r$ with the minimum inner product of $\boldsymbol{u}_l$ is also the one with the largest angle. As such, we use linear time (i.e., $O(|N| \cdot d)$) to efficiently find a pair of pivot vectors $\boldsymbol{u}_l$ and $\boldsymbol{u}_r$ with a large angle. Then, we assign each $\boldsymbol{u} \in N.S$ to the pivot having a smaller angle with $\boldsymbol{u}$ and thus split S into two subsets S_l and S_r (Lines 7 & 8). For a leaf node N, we additionally maintain the angle $\theta_{\boldsymbol{u}}$ between each $\boldsymbol{u} \in N.S$ and $N.\boldsymbol{c}$ (Lines 13 & 14). Suppose N_0 is the maximum leaf size. We start by assigning all user vectors in $\mathcal{U}$ to the root node. The Cone-Tree is built by performing the splitting procedure recursively from the root node until all leaf nodes contain at most N_0 user vectors. By the Cone-Tree construction, each leaf node contains a set of user vectors close to each other. Thus, the leaf nodes are used as the blocks of user vectors.

Shifting-aware Asymmetric Hashing

Finally, we combine SA-ALSH with the Cone-Tree blocking strategy and propose the Shifting-aware Asymmetric Hashing (SAH) algorithm for solving RkMIPS. SAH uses SA-ALSH to speed up the kMIPS on item vectors. Furthermore, it leverages the Cone-Tree blocking strategy along with node and vector-level upper bounds for pruning user vectors. In addition, it inherits the basic idea of Simpfer (Amagata and Hara 2021) to utilize the lower-bound arrays of user vectors to get a quick answer for the kMIPS.

Algorithm 4: SAH Indexing

Input: Item set $\mathcal{P}$, user set $\mathcal{U}$, $k_{max} \in \mathbb{Z}^+$, number of hash tables $K \in \mathbb{Z}^+$, maximum leaf size N_0;

1 Compute $\|\boldsymbol{p}_j\|$ for each item $\boldsymbol{p}_j \in \mathcal{P}$;
2 Sort $\mathcal{P}$ in descending order of $\|\boldsymbol{p}_j\|$;
3 $\mathcal{P}' \leftarrow$ the first $O(k_{max})$ items of $\mathcal{P}$;
4 **foreach** *user* $\boldsymbol{u}_i \in \mathcal{U}$ **do**
5     $\mathcal{S} \leftarrow k_{max}$MIPS of $\boldsymbol{u}_i$ on $\mathcal{P}'$;
6   **foreach** $\boldsymbol{p}^*_j \in \mathcal{S}$ **do**
7         $L^j_i \leftarrow \langle \boldsymbol{u}_i, \boldsymbol{p}^*_j \rangle$;
8 Build an SA-ALSH index for $\mathcal{P} \setminus \mathcal{P}'$;
9 $T \leftarrow$ Cone-Tree Construction($\mathcal{U}$, N_0);
10 $\mathcal{B} \leftarrow$ Extract all leaf nodes from T;
11 **foreach** *block* $B \in \mathcal{B}$ **do**
12   **foreach** *user* $\boldsymbol{u}_i \in B$ **do**
13   **for** $j = 1$ **to** k_{max} **do**
14             $L^j(B) \leftarrow \min\{L^j(B), L^j_i\}$;

Indexing Phase. The indexing phase of SAH is depicted in Algorithm 4. Given a set of item vectors $\mathcal{P}$ and a set of user vectors $\mathcal{U}$, it first computes $\|\boldsymbol{p}_j\|$ for each $\boldsymbol{p}_j \in \mathcal{P}$ and sort $\mathcal{P}$ in descending order of $\|\boldsymbol{p}_j\|$ (Lines 1 & 2). Let $\mathcal{P}'$ be the first $O(k_{max})$ item vectors in the sorted $\mathcal{P}$. It then retrieves the k_{max}MIPS results $\mathcal{S}$ on $\mathcal{P}'$ and computes a lower-bound array L_i of size k_{max} for each $\boldsymbol{u}_i \in \mathcal{U}$, i.e., $L^j_i = \langle \boldsymbol{u}_i, \boldsymbol{p}^*_j \rangle$, where $\boldsymbol{p}^*_j$ is the item of the jth ($1 \leq j \leq k_{max}$) largest inner product in $\mathcal{S}$ (Lines 3–7). Next, it calls Algorithm 1 to builds an SA-ALSH index for $\mathcal{P} \setminus \mathcal{P}'$ (Line 8) and calls Algorithm 3 to build a Cone-Tree T for $\mathcal{U}$ (Line 9). Then, it extracts each leaf node in the Cone-Tree T as a block $B \in \mathcal{B}$ for SAH (Line 10). Finally, it maintains a lower-bound array $L(B)$ of size k_{max} for each block $B \in \mathcal{B}$ (Lines 11–14).

Query Phase. The query phase of SAH is shown in Algorithm 5. Since every user $\boldsymbol{u}_i \in \mathcal{U}$ might be included in the RkMIPS results of the query $\boldsymbol{q}$, SAH checks each block $B \in \boldsymbol{B}$ individually. According to Lemma 2, it first verifies if $\|\boldsymbol{q}\| \cos(\{\phi - N.\omega\}_+) < L^k(B)$ or not (Line 3). If yes, it is safe to skip all user vectors in B; otherwise, each user $\boldsymbol{u}_i$ in B should be further checked. Then, based on Lemma 3, it determines whether $\|\boldsymbol{q}\| \cos(|\phi - \theta_{\boldsymbol{u}_i}|) < L^k_i$ (Line 5). If yes, $\boldsymbol{u}_i$ can be pruned; otherwise, it computes the actual inner product $\langle \boldsymbol{u}_i, \boldsymbol{q} \rangle$ and prunes $\boldsymbol{u}_i$ when $\langle \boldsymbol{u}_i, \boldsymbol{q} \rangle < L^k_i$ (Line 6). If $\boldsymbol{u}_i$ still cannot be pruned, since the item vectors are sorted in descending order of their l_2-norms, $\|\boldsymbol{u}_i\| \cdot \|\boldsymbol{p}_k\| = \|\boldsymbol{p}_k\|$ is the upper bound of the kth largest inner product of $\boldsymbol{u}_i$. If $\langle \boldsymbol{u}_i, \boldsymbol{q} \rangle \geq \|\boldsymbol{p}_k\|$, $\boldsymbol{u}_i$ can be added to the RkMIPS results $\mathcal{S}$ of $\boldsymbol{q}$; otherwise, SA-ALSH (i.e., Algorithm 2) is called for performing kMIPS on $\boldsymbol{u}_i$ to decide whether $\boldsymbol{q}$ is in the kMIPS results among $\mathcal{P} \cup \{\boldsymbol{q}\}$, and $\boldsymbol{u}_i$ is added to $\mathcal{S}$ if the answer ans is "yes" (Lines 7–11). Finally, it returns $\mathcal{S}$ as the RkMIPS results of $\boldsymbol{q}$ (Line 12).

Based on Theorem 2, we prove that SAH has a theoretical guarantee for RMIPS in subquadratic time and space.

Algorithm 5: SAH

Input: query vector $\boldsymbol{q}$, $k \in \mathbb{Z}^+$, item set $\mathcal{P}$, user set $\mathcal{U}$, Cone-Tree blocks $\mathcal{B}$;

1 $\mathcal{S} \leftarrow \emptyset$;
2 **foreach** *block* $B \in \mathcal{B}$ **do**
3   **if** $\|\boldsymbol{q}\| \cos(\{\phi - N.\omega\}_+) < L^k(B)$ **then continue**;
4   **foreach** *user* $\boldsymbol{u}_i \in B$ **do**
5   **if** $\|\boldsymbol{q}\| \cos(|\phi - \theta_{\boldsymbol{u}_i}|) < L^k_i$ **then continue**;
6   **if** $\langle \boldsymbol{u}_i, \boldsymbol{q} \rangle < L^k_i$ **then continue**;
7   **if** $\|\boldsymbol{p}_k\| > \langle \boldsymbol{u}_i, \boldsymbol{q} \rangle$ **then**
8             $ans \leftarrow$ SA-ALSH($\boldsymbol{u}_i, \boldsymbol{q}, k$);
9   **if** $ans = yes$ **then** $\mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{u}_i\}$;
10   **else**
11             $\mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{u}_i\}$;
12 **return** $\mathcal{S}$;

Theorem 3. *Given a set of n item vectors $\mathcal{P}$ and a set of m user vectors $\mathcal{U}$ ($m = O(n)$), SAH is a data structure that finds each user vector $\mathbf{u} \in \mathcal{U}$ for any query vector $\boldsymbol{q} \in \mathbb{R}^d$ such that $\mathbf{q}$ is the MIPS result of $\mathbf{u}$ among $\mathcal{P} \cup \{\boldsymbol{q}\}$ with constant probability in $O(dn^{1+\rho} \log_{1/p_2} n)$ time and $O(n^{1+\rho})$ space, where $\rho = \ln(1/p_1)/\ln(1/p_2)$.*

Experiments

Setup. We evaluate the performance of SAH for RkMIPS through extensive experiments. We compare SAH with one state-of-the-art kMIPS method H2-ALSH[1] and the only known RkMIPS method Simpfer.[2] To provide a more systematic comparison, we integrate the RkMIPS optimizations of Simpfer into H2-ALSH as a new baseline called H2-Simpfer. All methods are implemented in C++ and compiled by g++-8 using -O3 optimization. We conduct all experiments on a server with an Intel® Xeon® Platinum 8170 CPU @ 2.10GHz and 512 GB memory, running on CentOS 7.4. Each method is run on a single thread.

In the experiments, we use five real-world recommendation datasets, i.e., Amazon-Auto,[3] Amazon-CDs,[4] MovieLens,[5] Music100 (Morozov and Babenko 2018), and Netflix (Bennett and Lanning 2007). The numbers of item and user vectors (n, m) in Amazon-Auto, Amazon-CDs, MovieLens, Music100, and Netflix are (925387, 3873247), (64443, 75258), (10681, 71567), (1000000, 1000000), and (17770, 480189), respectively. For each dataset, the dimensionality d is 100, and we randomly select 100 item vectors as queries. The detailed procedures of dataset generation are described in the full version (Huang, Wang, and Tung 2022).

We use the query time to evaluate search efficiency and the F1-score to assess search accuracy. For SAH, we use $K = 128$ hash tables in SA-ALSH and set the leaf size

[1]https://github.com/HuangQiang/H2_ALSH
[2]https://github.com/amgt-d1/Simpfer
[3]https://nijianmo.github.io/amazon/index.html
[4]http://jmcauley.ucsd.edu/data/amazon/index_2014.html
[5]https://grouplens.org/datasets/movielens/

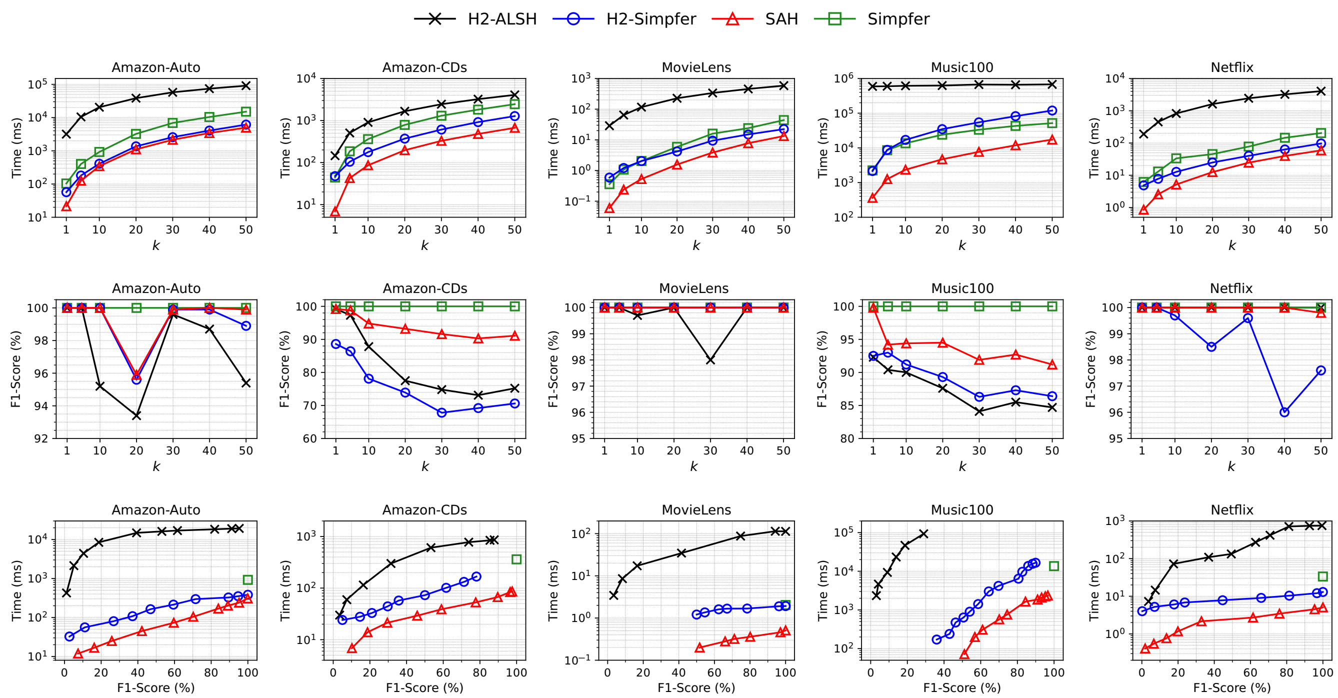

Figure 1: Query performance of different algorithms.

$N_0 = 20$ in Cone-Tree. We fix $b = 0.5$ for SAH, H2-ALSH, and H2-Simpfer and set $k_{max} = 50$ for SAH, Simpfer, and H2-Simpfer; for other parameters in Simpfer and H2-ALSH, we use their default values (Amagata and Hara 2021; Huang et al. 2018). All results are averaged by repeating each experiment five times with different random seeds.

Query Performance. We evaluate the performance of all methods for RkMIPS with varying $k \in \{1, 5, 10, 20, 30, 40, 50\}$. The results are shown in the first two rows of Figure 1.

From the first row of Figure 1, we observe that SAH is about 4∼8× faster than Simpfer. This is because SAH leverages SA-ALSH to conduct the kMIPS on item vectors, each in $O(dn^\rho \log n)$ time, where $0 < \rho < 1$. In contrast, Simpfer retrieves the kMIPS results by performing a linear scan over all item vectors, which requires $O(nd)$ time. Compared with H2-ALSH, the advantage of SAH is more apparent: it runs nearly or over two orders of magnitude faster than H2-ALSH. In particular, on the Music100 dataset, H2-ALSH is about three orders of magnitude slower than other methods, and it requires taking more than one day to complete all queries. This observation justifies that leveraging the existing hashing schemes (e.g., H2-ALSH) to speed up the kMIPS for each user vector is not efficient for RkMIPS. It also validates the effectiveness of our Cone-Tree blocking for pruning user vectors. Besides, SAH always runs (up to 8×) faster than H2-Simpfer. Since both methods utilize sublinear-time algorithms for kMIPS, this finding further confirms the effectiveness of our pruning strategies based on the cone structure.

Moreover, we plot the curves of F1-score vs. k of all methods in the second row of Figure 1. As Simpfer is an exact RkMIPS method, its F1-scores are always 100%. The F1-scores of SAH across all datasets are over 90% and consistently higher than those of H2-ALSH and H2-Simpfer. This advantage is attributed to the reductions in distortion errors coming from the shifting-invariant asymmetric transformation compared with the QNF transformation used in H2-ALSH and H2-Simpfer.

Finally, we illustrate the query time of each algorithm as a function of F1-score when $k = 10$ to compare their trade-offs between time efficiency and query accuracy in the third row of Figure 1. For each LSH-based algorithm, we present its query time and F1-score by varying the number of probed buckets until either the F1-score reaches 100%, or the total number of probed buckets exceeds 50%. Since Simpfer is an exact algorithm without hash-based partitioning, we only plot its query time @100% F1-score. The results are consistent with the ones for varying k in the first two rows of Figure 1. We observe that (1) SAH uniformly achieves the best trade-off between efficiency and accuracy in all cases; (2) the query time of SAH is still much lower than that of Simpfer when its F1-score approaches 100%. These results confirm the superior effectiveness and efficiency of SAH for RkMIPS in a more detailed manner.

Indexing Time. We present the indexing time of each algorithm in Table 1. H2-ALSH always takes the least time for index construction because it only builds an index on item vectors. Nevertheless, it cannot prune any user vector in the RkMIPS processing. Thus, its query efficiency is not comparable to other algorithms for solving RkMIPS. The indexing time of SAH is slightly (1.05∼1.43×) longer than that of Simpfer, primarily because of the additional cost of

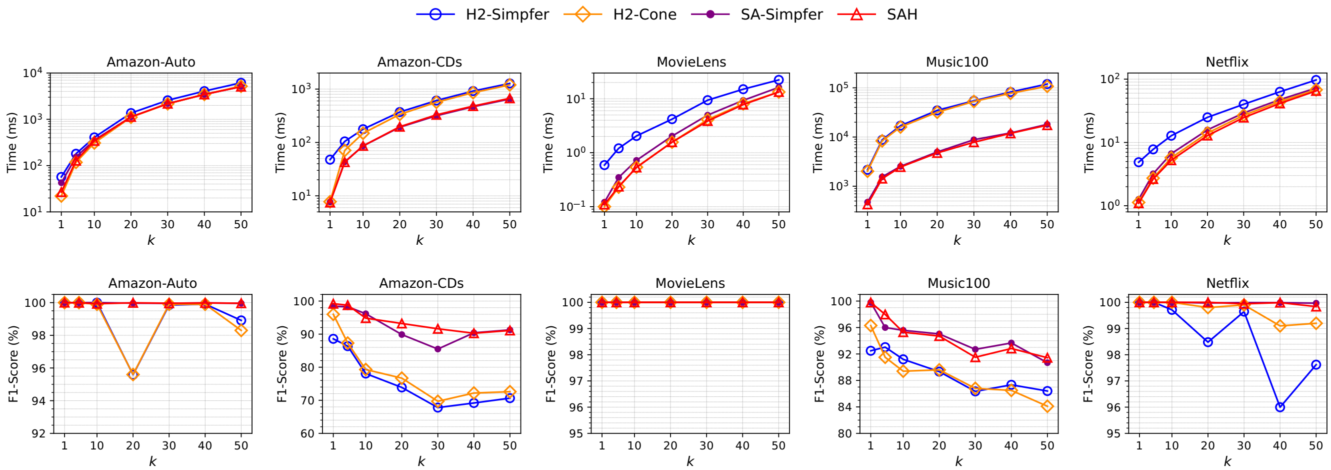

Figure 2: Ablation Study for SAH with varying $k \in \{1, 5, 10, 20, 30, 40, 50\}$.

Dataset	Simpfer	H2-ALSH	H2-Simpfer	SAH
Amazon-Auto	329.8	**19.7**	294.1	428.0
Amazon-CDs	9.4	**1.3**	9.7	11.5
MovieLens	2.6	**0.1**	1.9	2.7
Music100	32.8	**9.3**	32.2	46.6
Netflix	18.5	**0.2**	13.3	20.8

Table 1: Indexing time (in seconds) of each algorithm.

building the Cone-Tree. Nevertheless, the indexing phase of SAH can always be completed within 7.2 minutes, even on the Amazon-Auto dataset with nearly 1M items and 3.8M users. This finding indicates that the index construction of SAH is scalable to large datasets.

Ablation Study. We conduct an ablation study for SAH. To validate the effectiveness of SA-ALSH and the Cone-Tree blocking strategy separately, we first remove the Cone-Tree blocking strategy from SAH and integrate the Simpfer optimizations into SA-ALSH as a new baseline called SA-Simpfer. Then, we replace SA-ALSH with H2-ALSH while retaining the Cone-Tree blocking strategy as another baseline called H2-Cone. We show the results of H2-Simpfer, H2-Cone, SA-Simpfer, and SAH in Figure 2.

From the first row of Figure 2, we discover that H2-Simpfer takes longer query time than SA-Simpfer in almost all cases. This observation validates that SA-ALSH is more efficient than H2-ALSH for performing kMIPS, which would be because SA-ALSH incurs less distortion error than H2-ALSH, so that it finds the kMIPS results as early as possible and triggers the upper bound $\mu_j = M_j \|\boldsymbol{u}\|$ of the user vector $\boldsymbol{u}$ for early pruning. Moreover, we find that the query time of H2-Cone and SAH is uniformly shorter than H2-Simpfer and SA-Simpfer, respectively. This finding confirms the effectiveness of the Cone-Tree blocking compared with the norm-based blocking in Simpfer. For some datasets such as Amazon-Auto and Music100, the advantage of the Cone-Tree blocking is not very apparent because the effectiveness of upper bounds in Lemmas 2 and 3 to prune unnecessary inner product evaluations relies on the angle distribution of user vectors, and they might be less useful when the angles between most pairs of user vectors are close to $\pi/2$.

From the second row of Figure 2, we discover that the F1-scores of H2-Simpfer and H2-Cone are worse than those of SA-Simpfer and SAH, and their results are also less stable. This discovery empirically justifies the effectiveness of SAT in reducing the distortion error so that SA-Simpfer and SAH, where SA-ALSH is used for kMIPS, have higher F1-scores than H2-Simpfer and H2-Cone. These results are consistent with Figure 1. Finally, H2-Cone and SA-Simpfer are inferior to SAH in almost all cases, which validates that the integration of SA-ALSH and Cone-Tree-based blocking further improves the performance upon using them individually with existing blocking methods or ALSH schemes.

In addition, we also study the impact of the parameters of SAH and validate the effectiveness of SA-ALSH for kMIPS. Due to space limitations, the results and analyses are left to the full version (Huang, Wang, and Tung 2022).

Conclusion

In this paper, we studied a new yet difficult problem called RkMIPS on high-dimensional data. We proposed the first subquadratic-time algorithm SAH to tackle the RkMIPS efficiently and effectively in two folds. First, we developed a novel sublinear-time hashing scheme SA-ALSH to accelerate the kMIPS on item vectors. With the shifting-invariant asymmetric transformation, the distortion errors were reduced significantly. Second, we devised a new Cone-Tree blocking strategy that effectively pruned user vectors (in a batch). Extensive experiments on five real-world datasets confirmed the superior performance of SAH in terms of search accuracy and efficiency. Our work will likely contribute to opening up a new research direction and providing a practical solution to this challenging problem.

In future work, since the SAH algorithm achieves a theoretical guarantee for solving RkMIPS only when $k = 1$, it would be interesting to design a subquadratic-time algorithm for approximate RkMIPS with any $k > 1$.

Acknowledgments

We thank Yangyang Guo for helping us with dataset generation. This research is supported by the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative and the National Natural Science Foundation of China under grant No. 62202169. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of the National Research Foundation, Singapore.

References

Abuzaid, F.; Sethi, G.; Bailis, P.; and Zaharia, M. 2019. To Index or Not to Index: Optimizing Exact Maximum Inner Product Search. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 1250–1261.

Achtert, E.; Böhm, C.; Kröger, P.; Kunath, P.; Pryakhin, A.; and Renz, M. 2006. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 515–526.

Amagata, D.; and Hara, T. 2021. Reverse Maximum Inner Product Search: How to efficiently find users who would like to buy my item? In *The Fifteenth ACM Conference on Recommender Systems (RecSys)*, 273–281.

Andoni, A.; and Indyk, P. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 459–468.

Andoni, A.; Indyk, P.; Laarhoven, T.; Razenshteyn, I. P.; and Schmidt, L. 2015. Practical and Optimal LSH for Angular Distance. In *Advances in Neural Information Processing Systems 28 (NIPS)*, 1225–1233.

Arthur, D.; and Oudot, S. Y. 2010. Reverse Nearest Neighbors Search in High Dimensions using Locality-Sensitive Hashing. arXiv:1011.4955.

Ballard, G.; Kolda, T. G.; Pinar, A.; and Seshadhri, C. 2015. Diamond Sampling for Approximate Maximum All-Pairs Dot-Product (MAD) Search. In *2015 IEEE International Conference on Data Mining (ICDM)*, 11–20.

Bennett, J.; and Lanning, S. 2007. The Netflix Prize. In *Proceedings of KDD Cup and Workshop 2007 (KDDCup)*, 3–6.

Charikar, M. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, 380–388.

Dai, X.; Yan, X.; Ng, K. K. W.; Liu, J.; and Cheng, J. 2020. Norm-Explicit Quantization: Improving Vector Quantization for Maximum Inner Product Search. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, 51–58.

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. S. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry (SCG)*, 253–262.

Ding, Q.; Yu, H.; and Hsieh, C. 2019. A Fast Sampling Algorithm for Maximum Inner Product Search. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics (AISTATS)*, 3004–3012.

Gan, J.; Feng, J.; Fang, Q.; and Ng, W. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 541–552.

Guo, R.; Kumar, S.; Choromanski, K.; and Simcha, D. 2016. Quantization based Fast Inner Product Search. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 482–490.

Har-Peled, S.; Indyk, P.; and Motwani, R. 2012. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory Comput.*, 8: 321–350.

Huang, Q.; Feng, J.; Zhang, Y.; Fang, Q.; and Ng, W. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proc. VLDB Endow.*, 9(1): 1–12.

Huang, Q.; Ma, G.; Feng, J.; Fang, Q.; and Tung, A. K. H. 2018. Accurate and Fast Asymmetric Locality-Sensitive Hashing Scheme for Maximum Inner Product Search. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, 1561–1570.

Huang, Q.; Wang, Y.; and Tung, A. K. H. 2022. SAH: Shifting-aware Asymmetric Hashing for Reverse k-Maximum Inner Product Search. arXiv:2211.12751.

Indyk, P.; and Motwani, R. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*, 604–613.

Keivani, O.; Sinha, K.; and Ram, P. 2018. Improved maximum inner product search with better theoretical guarantee using randomized partition trees. *Mach. Learn.*, 107(6): 1069–1094.

Koenigstein, N.; Ram, P.; and Shavitt, Y. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM)*, 535–544.

Koren, Y.; Bell, R. M.; and Volinsky, C. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8): 30–37.

Korn, F.; and Muthukrishnan, S. 2000. Influence Sets Based on Reverse Nearest Neighbor Queries. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 201–212.

Lei, Y.; Huang, Q.; Kankanhalli, M.; and Tung, A. K. 2020. Locality-sensitive hashing scheme based on longest circular co-substring. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2589–2599.

Lei, Y.; Huang, Q.; Kankanhalli, M. S.; and Tung, A. K. H. 2019. Sublinear Time Nearest Neighbor Search over Generalized Weighted Space. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 3773–3781.

Li, H.; Chan, T. N.; Yiu, M. L.; and Mamoulis, N. 2017. FEXIPRO: Fast and Exact Inner Product Retrieval in Recommender Systems. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD)*, 835–850.

Liu, J.; Yan, X.; Dai, X.; Li, Z.; Cheng, J.; and Yang, M. 2020. Understanding and Improving Proximity Graph Based Maximum Inner Product Search. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, 139–146.

Lorenzen, S. S.; and Pham, N. 2020. Revisiting Wedge Sampling for Budgeted Maximum Inner Product Search. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part I*, 439–455.

Morozov, S.; and Babenko, A. 2018. Non-metric Similarity Graphs for Maximum Inner Product Search. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 4726–4735.

Neyshabur, B.; and Srebro, N. 2015. On Symmetric and Asymmetric LSHs for Inner Product Search. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 1926–1934.

Pham, N. 2021. Simple Yet Efficient Algorithms for Maximum Inner Product Search via Extreme Order Statistics. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 1339–1347.

Ram, P.; and Gray, A. G. 2012. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 931–939.

Shen, F.; Liu, W.; Zhang, S.; Yang, Y.; and Shen, H. T. 2015. Learning Binary Codes for Maximum Inner Product Search. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 4148–4156.

Shrivastava, A.; and Li, P. 2014. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In *Advances in Neural Information Processing Systems 27 (NIPS)*, 2321–2329.

Shrivastava, A.; and Li, P. 2015. Improved Asymmetric Locality Sensitive Hashing (ALSH) for Maximum Inner Product Search (MIPS). In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, 812–821.

Singh, A.; Ferhatosmanoglu, H.; and Tosun, A. S. 2003. High dimensional reverse nearest neighbor queries. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM)*, 91–98.

Tan, S.; Xu, Z.; Zhao, W.; Fei, H.; Zhou, Z.; and Li, P. 2021. Norm Adjusted Proximity Graph for Fast Inner Product Retrieval. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 1552–1560.

Tan, S.; Zhou, Z.; Xu, Z.; and Li, P. 2019. On Efficient Retrieval of Top Similarity Vectors. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 5235–5245.

Tao, Y.; Papadias, D.; and Lian, X. 2004. Reverse kNN Search in Arbitrary Dimensionality. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB)*, 744–755.

Tao, Y.; Yi, K.; Sheng, C.; and Kalnis, P. 2009. Quality and efficiency in high dimensional nearest neighbor search. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 563–576.

Teflioudi, C.; and Gemulla, R. 2017. Exact and Approximate Maximum Inner Product Search with LEMP. *ACM Trans. Database Syst.*, 42(1): 5:1–5:49.

Vlachou, A.; Doulkeridis, C.; Kotidis, Y.; and Nørvåg, K. 2010. Reverse top-k queries. In *2010 IEEE 26th International Conference on Data Engineering (ICDE)*, 365–376.

Vlachou, A.; Doulkeridis, C.; Kotidis, Y.; and Nørvåg, K. 2011. Monochromatic and Bichromatic Reverse Top-k Queries. *IEEE Trans. Knowl. Data Eng.*, 23(8): 1215–1229.

Vlachou, A.; Doulkeridis, C.; Nørvåg, K.; and Kotidis, Y. 2013. Branch-and-bound algorithm for reverse top-k queries. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 481–492.

Xiang, L.; Yan, X.; Lu, L.; and Tang, B. 2021. GAIPS: Accelerating Maximum Inner Product Search with GPU. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 1920–1924.

Xue, H.; Dai, X.; Zhang, J.; Huang, S.; and Chen, J. 2017. Deep Matrix Factorization Models for Recommender Systems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, 3203–3209.

Yan, X.; Li, J.; Dai, X.; Chen, H.; and Cheng, J. 2018. Norm-Ranging LSH for Maximum Inner Product Search. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2956–2965.

Yang, C.; and Lin, K. D. 2001. An Index Structure for Efficient Reverse Nearest Neighbor Queries. In *Proceedings of the 17th International Conference on Data Engineering (ICDE)*, 485–492.

Yu, H.; Hsieh, C.; Lei, Q.; and Dhillon, I. S. 2017. A Greedy Approach for Budgeted Maximum Inner Product Search. In *Advances in Neural Information Processing Systems 30 (NIPS)*, 5453–5462.

Zhou, Z.; Tan, S.; Xu, Z.; and Li, P. 2019. Möbius Transformation for Fast Inner Product Search on Graph. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 8216–8227.