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

Qiang Huang¹, Yanhao Wang²*, Anthony K. H. Tung¹

¹School of Computing, National University of Singapore, Singapore
²School 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 (RK-MIPS). Given a query (item) vector, a set of item vectors, and a set of user vectors, the problem of RK-MIPS 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 RK-MIPS 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 (SALSH) 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 RK-MIPS 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 $P \subseteq \mathbb{R}^d$ that has the largest inner product with a query (user) vector $q \in \mathbb{R}^d$, i.e., $p^* = \text{arg max}_{p \in P} \langle p, q \rangle$, as well as its extension $k$MIPS that finds $k$ ($k \geq 1$) vectors with the largest inner products for recommending items to users. Due to its prominence in recommender systems, the $k$MIPS 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 $k$MIPS yet less explored: \textit{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 $k$MIPS 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 $k$MIPS results, called Reverse $k$-Maximum Inner Product Search (RK-MIPS). Formally,

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

Compared with the $k$MIPS, the problem of RK-MIPS 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 RK-MIPS results of a query $q$. 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 $P \cup \{q\}$ for each user $u \in U$ and adding $u$ to the RK-MIPS results of $q$ once $q$ is included in the $k$MIPS results of $u$. 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-
ity of brute-force \(k\text{MIPS}\) and is often computationally prohibitive, especially for large \(n\).

Despite the importance of \(R\text{MIPS}\) 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 \(R\text{MIPS}\). Its primary idea is to efficiently solve a decision version of \(k\text{MIPS}\) for each user. Simpfer maintains a lower-bound array of the \(k\)th largest inner product of size \(k_{\text{max}} (k \in \{1, 2, \cdots, k_{\text{max}}\})\) for each user \(u \in U\) based on the \(O(k_{\text{max}})\) items with the largest \(l_2\)-norms such that each user \(u\) can get a quick "yes"/"no" answer for any query \(q\) on whether it belongs to the \(k\text{MIPS}\) results of \(u\). Moreover, it performs a linear scan using the Cauchy-Schwarz inequality to accelerate the \(k\text{MIPS}\) on item vectors. To reduce the number of user vectors for \(k\text{MIPS}\), 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 \(O(n^2d)\) 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 \(k\text{MIPS}\) (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 \(k\text{MIPS}\) for each \(u \in U\). As such, the time to perform \(R\text{MIPS}\) 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 \(k\text{MIPS}\) 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 \(I : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}\) and a user transformation \(U : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}\) 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 \(I(p)\) and \(U(u)\) will be much smaller than that in the original space. As a result, any \(I(p)\) can be the NNS result of \(U(u)\) even though their inner product \((p, u)\) is very small. Thus, the \(k\text{MIPS}\) results can be arbitrarily bad.

In addition, the \(Rk\text{MIPS}\) 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 \(R\text{MIPS}\) 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 \(R\text{MIPS}\) is the Reverse \(k\)-Nearest Neighbor Search (\(Rk\text{NNS}\)) (Korn and Muthukrishnan 2000; Yang and Lin 2001; Singh, Ferhatosmanoglu, and Tosun 2003; Tao, Papadias, and Lian 2004; Aichert et al. 2006; Arthur and Oudot 2010). Nevertheless, the case of reverse top-\(k\) queries, most existing \(Rk\text{NNS}\) 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 \(R\text{MIPS}\) in high-dimensional spaces. To accelerate the \(k\text{MIPS}\) 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 \(k\text{MIPS}\). We prove that SAH achieves a theoretical guarantee for solving \(R\text{MIPS}\) when \(k = 1\) in subquadratic time and space. In the experiments, we systematically compare SAH with a state-of-the-art \(k\text{MIPS}\) method H2-ALSH (Huang et al. 2018) as well as the only known \(R\text{MIPS}\) method Simpfer (Amagata and Hara 2021). Extensive results over five real-world datasets demonstrate that SAH runs 4–8× faster than them for \(R\text{MIPS}\) while achieving F1-scores of over 90%.

### Background

Before presenting SAH for solving \(R\text{MIPS}\), 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 \(p\) and \(u\) collide in the same bucket if \(h(p) = h(u)\). Let \(\text{Dist}(p, u)\) be a distance function of any two vectors \(p\) and \(u\). 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 \(\text{Dist}(\cdot, \cdot)\) if, for any \(p, u \in \mathbb{R}^d\), it satisfies:

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

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 \(p \in \mathcal{P}\) such that \(\text{Dist}(p, u) \leq c \cdot \text{Dist}(p^*, u)\) in \(O(n^{1+\rho})\) space and \(O(n \rho \log_1/p_2 n)\) query time, where \(\rho = \ln p_1/\ln p_2\) and \(p^* = \arg \min_{p \in \mathcal{P}} \text{Dist}(p, u)\).

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

$$h_{srp}(p) = \text{sgn}(\langle a, p \rangle),$$

(1)

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

$$p(\delta) = \Pr[h_{srp}(p) = h_{srp}(u)] = 1 - \frac{\delta}{\pi}.$$

(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., $I : \mathbb{R}^d \rightarrow \mathbb{R}^d$ (Item transformation) and $U : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ (User transformation), is called $(S_0, \frac{c}{c'-1}, p_1, p_2)$-sensitive to the inner product $\langle \cdot, \cdot \rangle$ for any $p, u \in \mathbb{R}^d$, if for any $p, u \in \mathbb{R}^d$, it satisfies:

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

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

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

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

(3)

$$U(u) = [\lambda u_1, \cdots, \lambda u_d; 0],$$

(4)

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

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

(5)

Let $\theta$ be the angle of $p$ and $u$. As $\langle p, u \rangle / \|u\| = \|p\| \cos \theta \leq \|p\| \leq M$, we have $M - \langle p, u \rangle / \|u\| \geq 0$. Since $M$ and $\|u\|$ 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 $\theta$ of any $p$ and $u$ in high-dimensional spaces is often close to $\pi/2$, incurring a very small $\cos \theta$. Thus, we often have $\|p\| \cos \theta \ll M$. In the worst case, $\max_{p \in \mathcal{P}} \|I(p) - U(u)\| / \min_{p \in \mathcal{P}} \|I(p) - U(u)\| \approx 1$. 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 $I(p)$ can be the NNS result of $U(u)$ even if $\langle p, u \rangle$ is small, which means that the MIPS result of H2-ALSH for $u$ 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 $\mathbb{R}^k$MIPS 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 $\mathbb{R}^{d+1}$. Let $c$ be the centroid of the item set $\mathcal{P}$, i.e., $c = [c_1, \cdots, c_d] = \frac{1}{\mathcal{P}} \sum_{p \in \mathcal{P}} p$. Suppose that $R$ is the radius of the smallest ball centered at $c$ enclosing all $p \in \mathcal{P}$, i.e., $R = \max_{p \in \mathcal{P}} \|p - c\|$. Given any item vector $p = [p_1, \cdots, p_d]$ and user vector $u = [u_1, \cdots, u_d]$, the item transformation $I : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$ and user transformation $U : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$ of SAT are:

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

(6)

$$U(u) = [\lambda u_1, \cdots, \lambda u_d; 0],$$

(7)

where $\lambda = R / \|u\|$.

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

$$\frac{\langle I(p, c), U(u) \rangle}{\|I(p, c)\| \cdot \|U(u)\|} = \frac{\langle p - c, u \rangle}{R \cdot \|u\|}.$$
Lemma 1. Given an inner product threshold \( \arccos(\frac{H}{H^2}) \) where \( H \) is the largest inner product among the item vectors in \( C \) with \( u \), we introduce a data-structure of user vectors with higher probability (Huang et al. 2018; Yan et al. 2018; Liu et al. 2020), we first compute the centroid \( c_j \) and radius \( R_j \), i.e., \( c_j = \frac{1}{|S_j|} \sum_{p \in S_j} p \) and \( R_j = \max_{p \in S_j} ||p - c_j|| \) (Lines 7). According to Equation 6, we apply \( I : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1} \) to convert each \( p \in S_j \) into \( I(p, c_j) \) (Lines 8–11). Finally, we generate a set of SRP-LSH functions and apply SimHash to build the index for all \( I(p, c_j) \)'s (Line 12).

Note that \( M_1, \ldots, M_t \) 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).

Algorithm 2: SA-ALSH

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

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

\[ M_j = \max_{p \in S_j} ||p|| \] (Lines 4–6). For each \( S_j \), we first compute its centroid \( c_j \) and radius \( R_j \), i.e., \( c_j = \frac{1}{|S_j|} \sum_{p \in S_j} p \) and \( R_j = \max_{p \in S_j} ||p - c_j|| \) (Line 7). According to Equation 6, we apply \( I : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1} \) to convert each \( p \in S_j \) into \( I(p, c_j) \) (Lines 8–11). Finally, we generate a set of SRP-LSH functions and apply SimHash to build the index for all \( I(p, c_j) \)'s (Line 12).

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

As SA-ALSH first performs \( k \)-MIPS on the item vectors with the largest \( l_2 \)-norms, which most probably contain the \( k \)-MIPS results of \( u \), 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.

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 \( ||p|| \) for each item \( p \in \mathcal{P} \) and sort \( \mathcal{P} \) in descending order of \( ||p|| \);
2. \( j = 0 \); \( i = 0 \);
3. while \( i < n \) do
4. \( j \leftarrow j + 1 \); \( M_j \leftarrow ||p_j|| \); \( S_j \leftarrow \emptyset \);
5. while \( i < n \) and \( ||p_i|| > b M_j \) do
6. \( S_j \leftarrow S_j \cup \{p_i\} \); \( i \leftarrow i + 1 \);
7. \( c_j = \frac{1}{|S_j|} \sum_{p \in S_j} p \); \( R_j = \max_{p \in S_j} ||p - c_j|| \);
8. \( I_j \leftarrow \emptyset \);
9. foreach item \( p \in S_j \) do
10. \( I(p, c_j) \leftarrow [p_1 - c_1, \ldots, p_d - c_d] \);
11. \( \sqrt{R_j^2 - ||p - c_j||^2} \);
12. \( \mathcal{T}_j \leftarrow \mathcal{T}_j \cup \{I(p, c_j)\} \);
13. Build the \( K \) hash tables for \( \mathcal{T}_j \) using SimHash;
14. \( t \leftarrow j \);
Theorem 2. Given a hash family $\mathcal{H}_{SA}$ of hash functions $h_{SA}()$ as defined by Equation 9, SA-ALSH is a data structure that finds an item vector $p \in \mathcal{P}$ for any user vector $u \in \mathbb{R}^d$ such that $\langle p, u \rangle \leq \langle p^*, u \rangle$ with constant probability in $O(dn^p \log |p| n)$ time and $O(n^{1+p})$ space, where $p = \ln(1/p_1)/\ln(1/p_2)$ and $p^* = \arg \max_{p \in \mathcal{P}} \langle p, u \rangle$.

Cone-Tree Blocking

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

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

Based on Fact 2, we simply assume that all user vectors are unit vectors, i.e., $\|u\| = 1$ for every $u \in \mathcal{U}$. Fact 2 also implies that the MIPS result is only affected by the direction of $u$. 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., $N.S \subseteq \mathcal{U}$. 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 $N.c$ and the maximum angle $N.\omega$ of the root node. The $N.\omega$ 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.

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 $N.c$ 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 $v \in N.S$ (Line 4); (2) we find the point $u_i$ with the minimum inner product of $v$, i.e., $u_i = \arg \min_{u \in S} \langle u, v \rangle$ (Line 5); (3) find another point $u_r$ with the minimum inner product of $u_i$, i.e., $u_r = \arg \min_{u \in S} \langle u, u_i \rangle$ (Line 6). As we assume that all user vectors are unit vectors, the point $u_r$ with the minimum inner product of $u_i$ is also the one with the largest angle. As such, we use line time (i.e., $O(|N| \cdot d)$) to efficiently find a pair of pivot vectors $u_i$ and $u_r$ with a large angle. Then, we assign each $u \in N.S$ to the pivot having a smaller angle with $u$ and thus split $S$ into two subsets $S_i$ and $S_r$ (Lines 7 & 8). For a leaf node $N$, we additionally maintain the angle $\theta_u$ between each $u \in N.S$ and $N.c$ (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 R$k$MIPS. SAH uses SA-ALSH to speed up 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 $k$MIPS.

Algorithm 3: Cone-Tree Construction

Input: Subset $S \subseteq \mathcal{U}$, maximum leaf size $N_0$.
1. $N.S \leftarrow S$; $N.c \leftarrow \frac{1}{|N|} \sum_{u \in N.S} u$;
2. $N.\omega \leftarrow \max_{u \in N.S} \arccos(\langle u, N.c \rangle / \|u\| \|N.c\|)$;
3. if $|N| > N_0$ then
4. Select a point $v \in N.S$ uniformly at random;
5. $u_i \leftarrow \arg \min_{u \in S} \langle u, v \rangle$;
6. $u_r \leftarrow \arg \min_{u \in S} \langle u, u_i \rangle$;
7. $S_i \leftarrow \{u \in S \mid \cos \theta_{u, u_i} \geq \cos \theta_{u, u_r}\}$;
8. $S_r \leftarrow S \setminus S_i$;
9. $N.lc \leftarrow \text{Cone-Tree Construction}(S_i, N_0)$;
10. $N.rc \leftarrow \text{Cone-Tree Construction}(S_r, N_0)$;
11. return $N$;
12. else
13. foreach $u \in N.S$ do
14. \[ \theta_u \leftarrow \arccos(\langle u, N.c \rangle / \|u\| \|N.c\|) \];
15. return $N$;
Algorithm 4: SAH Indexing

Input: item set $P$, user set $U$, $k_{max} \in \mathbb{Z}^+$, number of hash tables $K \in \mathbb{Z}^+$, maximum leaf size $N_0$;
1. Compute $\|p_j\|$ for each item $p_j \in P$;
2. Sort $P$ in descending order of $\|p_j\|$;
3. $P' \leftarrow$ the first $O(k_{max})$ items of $P$;
4. foreach user $u_i \in U$ do
5. $S \leftarrow k_{max}$-MIPS of $u_i$ on $P'$;
6. $S' \leftarrow S$ do
7. $L^j_i \leftarrow \langle u_i, p_j \rangle$;
8. Build an SA-ALSH index for $P \setminus P'$;
9. $T \leftarrow$ Cone-Tree Construction($U$, $N_0$);
10. $B \leftarrow$ Extract all leaf nodes from $T$;
11. foreach block $B \in B$ do
12. foreach user $u_i \in B$ 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 $P$ and a set of user vectors $U$, it first computes $\|p_j\|$ for each $p_j \in P$ and sort $P$ in descending order of $\|p_j\|$ (Lines 1 & 2). Let $P'$ be the first $O(k_{max})$ item vectors in the sorted $P$. It then retrieves the $k_{max}$-MIPS results $S$ on $P'$ and computes a lower-bound array $L^j_i$ of size $k_{max}$ for each $u_i \in U$, i.e., $L^j_i = \langle u_i, p_j \rangle$, where $p_j$ is the item of the $j$th $(1 \leq j \leq k_{max})$ largest inner product in $S$ (Lines 3–7). Next, it calls Algorithm 1 to build an SA-ALSH index for $P \setminus P'$ (Line 8) and calls Algorithm 3 to build a Cone-Tree $T$ for $U$ (Line 9). Then, it extracts each leaf node in the Cone-Tree $T$ as a block $B \in B$ for SAH (Line 10). Finally, it maintains a lower-bound array $L(B)$ of size $k_{max}$ for each block $B \in B$ (Lines 11–14).

Query Phase. The query phase of SAH is shown in Algorithm 5. Since every user $u_i \in U$ might be included in the $k_{MIPS}$ results of the query $q$, SAH checks each block $B \in B$ individually. According to Lemma 2, it verifies if $\|q\| \cos(\phi - N \omega) \setminus \omega < L^k(B)$ or not (Line 3). If yes, it is safe to skip all user vectors in $B$; otherwise, each user $u_i$ in $B$ should be further checked. Then, based on Lemma 3, it determines whether $\|q\| \cos(\phi - \theta_{u_i}) < L^k_i$ (Line 5). If yes, $u_i$ can be pruned; otherwise, it computes the actual inner product $\langle u_i, q \rangle$ and prunes $u_i$ when $\langle u_i, q \rangle < L^k_i$ (Line 6). If $u_i$ still cannot be pruned, since the item vectors are sorted in descending order of their $l_2$-norms, $\|u_i\| \cdot \|p_k\| = \|p_k\|$ is the upper bound of the $k$th largest inner product of $u_i$. If $\langle u_i, q \rangle \geq \|p_k\|$, $u_i$ can be added to the $k_{MIPS}$ results $S$ of $q$; otherwise, SA-ALSH (i.e., Algorithm 2) is called for performing $k_{MIPS}$ on $u_i$ to decide whether $q$ is in the $k_{MIPS}$ results among $P \cup \{q\}$, and $u_i$ is added to $S$ if the answer $ans$ is “yes” (Lines 7–11). Finally, it returns $S$ as the $k_{MIPS}$ results of $q$ (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 $q$, $k \in \mathbb{Z}^+$, item set $P$, user set $U$, Cone-Tree blocks $B$;
1. $S \leftarrow \emptyset$;
2. foreach block $B \in B$ do
3. if $\|q\| \cos(\phi - N \omega) \setminus \omega < L^k(B)$ then continue;
4. foreach user $u_i \in B$ do
5. if $\|q\| \cos(\phi - \theta_{u_i}) < L^k_i$ then continue;
6. if $\langle u_i, q \rangle < L^k_i$ then continue;
7. if $\|p_k\| > \langle u_i, q \rangle$ then
8. $ans \leftarrow SA-ALSH(u_i, q, k)$;
9. if $ans = yes$ then $S \leftarrow S \cup \{u_i\}$;
10. else
11. $S \leftarrow S \cup \{u_i\}$;
12. return $S$;

Theorem 3. Given a set of $n$ item vectors $P$ and a set of $m$ user vectors $U$ $(m = O(n))$, SAH is a data structure that finds each user vector $u \in U$ for any query vector $q \in \mathbb{R}^d$ such that $q$ is the MIPS result of $u$ among $P \cup \{q\}$ with constant probability in $O(dn^{1+p} \log_1/p_z n)$ time and $O(n^{1+p})$ space, where $p = \ln(1/p_1)/\ln(1/p_2)$.

Experiments

Setup. We evaluate the performance of SAH for $k_{MIPS}$ through extensive experiments. We compare SAH with one state-of-the-art $k_{MIPS}$ method $H2-ALSH$ and the only known $k_{MIPS}$ method Simpfer. To provide a more systematic comparison, we integrate the $k_{MIPS}$ 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, Amazon-CDs, MovieLens, 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 timing 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-dl/Simpfer
5. https://grouplens.org/datasets/movielens/
$N_0 = 20$ 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 R$k$MIPS 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 $k$MIPS on item vectors, each in $O(dn^p \log n)$ time, where $0 < \rho < 1$. In contrast, Simpfer retrieves the $k$MIPS 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 $k$MIPS for each user vector is not efficient for R$k$MIPS. 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 $k$MIPS, 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 R$k$MIPS 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 R$k$MIPS 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 R$k$MIPS processing. Thus, its query efficiency is not comparable to other algorithms for solving R$k$MIPS. The indexing time of SAH is slightly (1.05–1.43×) longer than that of Simpfer, primarily because of the additional cost of
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 Simpher 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 $\mu_j = M_j \|u\|$ of the user vector $u$ 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 Simpher. 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 SAH 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 full version (Huang, Wang, and Tung 2022).

**Conclusion**

In this paper, we studied a new yet difficult problem called $R_k$MIPS on high-dimensional data. We proposed the first subquadratic-time algorithm SAH to tackle the $R_k$MIPS 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 the theoretical guarantee for solving $R_k$MIPS only when $k = 1$, it would be interesting to design a subquadratic-time algorithm for approximate $R_k$MIPS 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.


