Simpler is Much Faster: Fair and Independent Inner Product Search

Kazuyoshi Aoyama
aoyama.kazuyoshi@ist.osaka-u.ac.jp
Osaka University

Daichi Amagata
amagata.daichi@ist.osaka-u.ac.jp
Osaka University

Sumio Fujita
sufujita@yahoo-corp.jp
Yahoo Japan Corporation

Takahiro Hara
hara@ist.osaka-u.ac.jp
Osaka University

ABSTRACT

The problem of inner product search (IPS) is important in many fields. Although maximum inner product search (MIPS) is often considered, its result is usually skewed and static. Users are hence hard to obtain diverse and/or new items by using the MIPS problem. Motivated by this, we formulate a new problem, namely the fair and independent IPS problem. Given a query, a threshold, and an output size k, this problem randomly samples k items from a set of items such that the inner product of the query and item is not less than the threshold. For each item that satisfies the threshold, this problem is fair, because the probability that such an item is outputted is equal to that for each other item. This fairness can yield diversity and novelty, but this problem faces a computational challenge. Some existing (M)IPS techniques can be employed in this problem, but they require $O(n)$ or $o(n)$ time, where n is the dataset size. To scale well to large datasets, we propose a simple yet efficient algorithm that runs in $O(\log n + k)$ expected time. We conduct experiments using real datasets, and the results demonstrate that our algorithm is up to 330 times faster than baselines.

CCS CONCEPTS

• **Information systems** → **Retrieval efficiency**.

KEYWORDS

inner product search, fairness, independence

ACM Reference Format:
Kazuyoshi Aoyama, Daichi Amagata, Sumio Fujita, and Takahiro Hara. 2023. Simpler is Much Faster: Fair and Independent Inner Product Search. In *Proceedings of the 46th Int'l ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23), July 23–27, 2023, Taipei, Taiwan.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3539618.3592061

1 INTRODUCTION

The inner product search (IPS) problem is important in many fields, e.g., information retrieval [18, 32, 35], recommender systems [6, 7, 30], data mining [19, 27], databases [22, 24, 29], artificial intelligence [13, 36], and machine learning [14, 23, 34]. These fields usually consider the k maximum inner product search (k-MIPS) problem, which, given a query vector and an output size k, returns the k vectors having the maximum inner product with the query. Although this problem considers user preferences, the search results are static, because, for the same queries (and k), the search results are always the same (as long as the vector set is static). In addition, the k-MIPS results tend to be skewed [17], although diversity and novel items are usually required, as claimed in real services [1, 8].

To remove the above limitations, this paper formulates a new problem, namely the *fair and independent inner product search* (or FI-IPS) problem. Given a query vector $\mathbf{q}$, a threshold τ, a set $\mathbf{X}$ of n vectors, and an output size k, this problem returns a set $\mathbf{Q}_k$ of k vectors such that $\mathbf{Q}_k \subseteq \mathbf{Q} = \{\mathbf{x} \mid \mathbf{x} \in \mathbf{X}, \mathbf{x} \cdot \mathbf{q} \geq \tau\}$ while satisfying the following: (i) $\Pr[\mathbf{x} \in \mathbf{Q}_k] = \Pr[\mathbf{x}' \in \mathbf{Q}_k]$ for any $\mathbf{x}$, $\mathbf{x}' \in \mathbf{Q}$ and (ii) $\mathbf{Q}_k$ is independent of the previous query results. The inner product threshold τ still takes the user preference into account, because τ is regarded as a user preference threshold. The probability condition guarantees fairness, as each vector in $\mathbf{Q}$ is returned with an equal probability. The independence condition ensures that each vector in $\mathbf{Q}_k$ is randomly chosen from $\mathbf{Q}$ (so the same query vectors may not have the same search results). This problem provides merits to both users and service providers.

(1) *Search results can be diverse.* Trivially, returning all vectors in $\mathbf{Q}$ may overwhelm users, because $|\mathbf{Q}|$ can be large as $|\mathbf{Q}| = O(n)$. To select k vectors (from $\mathbf{Q}$), some existing works use diversity maximization [4, 5, 17, 21, 26]. However, it is not always easy to select an objective function for diversity maximization that best fits applications [28]. In this case, k vectors randomly sampled from $\mathbf{Q}$ would be a good solution, because these k vectors are uniformly distributed in the space of $\mathbf{Q}$. The FI-IPS problem ensures this.

In addition, thanks to the independence property, the FI-IPS problem returns a different result for every request of the same user. This user, therefore, can know not only items with the highest inner products but also novel items for him/her.

(2) *Search results do not suffer from unfair ranking.* If we use a ranking-based approach (e.g., k-MIPS) to limit the result size, we may suffer from *unfairness* (usually derived from the unfairness hidden in datasets) [25]. For example, assume an e-commerce platform where items from some companies tend to be ranked at higher positions than those of the other companies. In this case, the latter companies have a complaint about this platform and leave. The FI-IPS problem alleviates this concern, as it does not employ such a ranking and all items $\mathbf{x}$ satisfying $\mathbf{x} \cdot \mathbf{q} \geq \tau$ have an equal probability of being outputted.

SIGIR '23, July 23–27, 2023, Taipei, Taiwan

ACM ISBN 978-1-4503-9408-6/23/07.
https://doi.org/10.1145/3539618.3592061

(3) *Queries can be anonymous.* Because of the randomness and the independence of the FI-IPS problem, it is hard to recover a given query. This effect contributes to easy-to-implement privacy-preserving query processing [9].

Challenge. Although the FI-IPS problem has the above advantages, it is not trivial to process an FI-IPS query efficiently. A straightforward algorithm for this problem is to compute $\mathbf{Q}$, but this algorithm requires $O(n)$ time, which is too long for applications that want only $k \ll n$ vectors. Even if we employ some existing techniques for fair and independent search problems, they still need $o(n) = O(n^{1-\epsilon})$ time, where $\epsilon \in (0, 1)$ is a constant, see Section 2.

Contribution. We overcome the above challenge and propose an efficient algorithm that guarantees the correctness and runs in $O(\log n + k)$ expected time. The FI-IPS problem trivially requires $\Omega(k)$ time, so our algorithm needs only an additional $O(\log n)$ factor. Furthermore, even if applications require a success probability of nearly 1, our algorithm still needs only $O(k \log n)$ time. We conduct experiments using real datasets, and the results confirm that our algorithm is significantly faster than baseline algorithms.

2 PRELIMINARY

Problem definition. Let $\mathbf{X}$ be a set of d-dimensional vectors (e.g., item embeddings), and $|\mathbf{X}| = n$. We assume that d is high and, for simplicity, $d = O(1)$. For ease of presentation, we first define the inner product search (IPS) problem.

Definition 1 (Inner product search problem). *Given a query vector $\mathbf{q}$, $\mathbf{X}$, and a threshold τ, this problem retrieves $\mathbf{Q} = \{\mathbf{x} \mid \mathbf{x} \in \mathbf{X}, \mathbf{x} \cdot \mathbf{q} \geq \tau\}$.*

Note that τ is a user-tolerable inner product (e.g., a user may specify $\tau = 4$ in 5-star rating systems).

The IPS problem cannot limit the output size and $|\mathbf{Q}|$ can be large, as $|\mathbf{Q}| = O(n)$. Returning all vectors in $\mathbf{Q}$ therefore not only overwhelms users but also incurs a large computational cost. In addition, many systems and users require the output size to be limited to k (e.g., $k \leq 20$), so it is important to consider how to select k vectors from $\mathbf{Q}$. Our idea is to choose k vectors uniformly at random for *fairness*, which follows existing works [2, 3, 20, 33]. Based on this idea, we propose the fair and independent IPS (FI-IPS) problem.

Definition 2 (Fair and independent IPS problem). *Given a query vector $\mathbf{q}$, $\mathbf{X}$, a threshold τ, and an output size k, this problem retrieves a set of k vectors $\mathbf{Q}_k \subseteq \mathbf{Q}$[1] and satisfies the following:*

- $\Pr[\mathbf{x} \in \mathbf{Q}_k] = \Pr[\mathbf{x}' \in \mathbf{Q}_k]$ *for any* $\mathbf{x}, \mathbf{x}' \in \mathbf{Q}$ *and*
- $\mathbf{Q}_k$ *is independent of the other query results.*

The first condition guarantees that each vector in $\mathbf{Q}$ has an equal probability of being included in $\mathbf{Q}_k$. The second condition guarantees that $\Pr[\mathbf{x} \in \mathbf{Q}_k]$ is independent of any other query results. For example, let $\mathbf{Q}_k^i$ be a result of a query $\mathbf{q}^i$, and we can have $\mathbf{Q}_k^i \neq \mathbf{Q}_k^j$ even if $\mathbf{q}^i = \mathbf{q}^j$. That is, under a recommendation scenario, each user can have a different recommendation list for every recommendation request. This is not guaranteed without the second condition (independence), suggesting its importance.

[1]Definition 2 assumes that τ is a reasonable value for satisfying $|\mathbf{Q}| \geq k$. If $|\mathbf{Q}| < k$, we return $\mathbf{Q}$.

Linear-Scan. A straightforward approach to solving the FI-IPS problem is to first obtain $\mathbf{Q}$ by solving the IPS problem and then randomly pick k vectors from $\mathbf{Q}$. This satisfies the fairness and independence conditions. Moreover, we can employ a state-of-the-art algorithm[2] (e.g., FEXIPRO [22]) to solve the IPS problem efficiently.

Large-scale systems need to handle a lot of users. Each search request (i.e., query) hence should be processed within a millisec (or less) order. However, the above approach is hard to achieve this for large datasets, because it incurs at least $\Omega(|\mathbf{Q}|)$ (at most $O(n)$) time. We usually have $k \ll |\mathbf{Q}|$, so $\tilde{O}(k)$ time algorithm is required, where $\tilde{O}(\cdot)$ hides any polylog factors.

Existing fair search algorithms in high dimensions are based on *random permutation* and *locality-sensitive hashing* (LSH) [9–11, 15]. A random permutation is a random shuffling of the order of vectors, making each vector at a *rank* with probability $1/n$. This approach is used to try to satisfy the fairness and independence conditions. (For details, refer to a nice summary [28]). LSH is a hashing technique, and similar vectors tend to have the same (or similar) hash values by using an LSH function. This technique has a probabilistic performance guarantee, and, for a given query, its similar vector can be found in time sub-linearly to n (but it incurs a super-linear space complexity). LSH, unfortunately, does not guarantee that the state-of-the-art techniques [9–11, 15] can always access each $\mathbf{x} \in \mathbf{Q}$ with equal probability.

3 FI-IPS ALGORITHM

As seen above, the existing techniques require $O(n)$ or $o(n)$ time, and [9–11, 15] do not always guarantee fairness. We overcome these limitations and propose an algorithm that runs in $O(\log n + k)$ expected time and guarantees fairness and independence.

Our main idea for satisfying the fairness, independence, and fast time is to (i) identify a superset of $\mathbf{Q}$ (but with a much smaller size than n) with a small computational cost and then (2) pick random vectors from the superset. Different from enumerating each vector in this superset (which requires $O(n)$ time in the worst case), we identify only the range (e.g., $[\alpha, \beta]$ and $1 \leq \alpha \leq \beta \leq n$) where each $\mathbf{x}_i \in \mathbf{Q}$ exists, to implement the first idea.

Algorithm description. Assume that $\mathbf{X}$ is sorted in descending order of Euclidean norm (i.e., $\|\mathbf{x}\|$). This is done *offline*, because it does not use any query information. Without loss of generality, we assume that $\|\mathbf{x}_1\| \geq \|\mathbf{x}_2\| \geq ... \geq \|\mathbf{x}_n\|$.

Given a query vector $\mathbf{q}$, a threshold τ, and an output size k,

① we first compute $j = \max_{i \in [1,n]} i$ such that $\|\mathbf{x}_i\|\|\mathbf{q}\| \geq \tau$. From Cauchy–Schwarz inequality ($\mathbf{x} \cdot \mathbf{q} \leq \|\mathbf{x}\|\|\mathbf{q}\|$), every vector $\mathbf{x}_{j+b}$ ($b \in [1, n-j]$) *cannot* satisfy $\mathbf{x}_{j+b} \cdot \mathbf{q} \geq \tau$. This means that we can focus only on $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_j\}$, and notice that $\mathbf{Q} \subseteq \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_j\}$.

② We next pick a random integer $i \in [1, j]$. If $\mathbf{x}_i \cdot \mathbf{q} \geq \tau$ and $\mathbf{x}_i \notin \mathbf{Q}_k$, we add $\mathbf{x}_i$ into $\mathbf{Q}_k$.

③ We iterate the above operation until we have $|\mathbf{Q}_k| = k$.

[2]State-of-the-art exact IPS algorithms are based on linear scan [22, 29]. Trivially, approximation algorithms which do not guarantee that all vectors such that $\mathbf{x} \cdot \mathbf{q} \geq \tau$ are included in the search result cannot solve the FI-IPS problem correctly. This paper does not assume GPU settings [31, 32], because parallelization does not overcome the essential drawback of $O(n)$ time.

Algorithm 1: Our FI-IPS Algorithm

Input: $\mathbf{X}$, $\mathbf{q}$, τ, and k
Output: $\mathbf{Q}_k$

```
j ← max i s.t. ‖x_i‖‖q‖ ≥ τ        ▷ ‖x_a‖ ≥ ‖x_{a+1}‖ for a ∈ [1, n − 1]
while |Q_k| < k do
    i ← a random integer in [1, j]
    if x_i · q ≥ τ then
        Q_k ← Q_k ∪ {x_i}
```

Algorithm 1 summarizes the above operations[3]. We below analyze the performance of our algorithm.

Fairness and independence. For each vector $\mathbf{x}_i \in \mathbf{Q}$, its sampling probability is always $1/j$. Our algorithm hence yields $\Pr[\mathbf{x} \in \mathbf{Q}_k] = \Pr[\mathbf{x}' \in \mathbf{Q}_k]$ for any $\mathbf{x}, \mathbf{x}' \in \mathbf{Q}$. In addition, it is straightforward to see that $\mathbf{Q}_k$ returned by our algorithm is a set of random samples of $\mathbf{Q}$. Thus, our algorithm guarantees independence.

Space complexity. Our algorithm maintains only $\mathbf{X}$, so the space complexity of our algorithm is trivially $O(n)$.

Time complexity. Although our algorithm is surprisingly simple and the above performances are easy to obtain, its time complexity is not trivial. Notice that operation ② of our algorithm may pick i such that $\mathbf{x}_i \cdot \mathbf{q} < \tau$ or i has been previously picked. Therefore, it is not clear how many iterations of ② are required to obtain a correct $\mathbf{Q}_k$. We achieve this non-trivial task and prove that our algorithm runs in $\tilde{O}(k)$ expected time.

Theorem 1. *Our algorithm correctly returns $\mathbf{Q}_k$ in $O(\log n + k)$ expected time.*

Proof. Computing j can be done in $O(\log n)$ time by a binary search. Next, we analyze the expected iteration number of ② that is necessary to return k *distinct* vectors in $\mathbf{Q}$. Assume that we have $\mathbf{Q}_k$ such that $|\mathbf{Q}_k| = k' < k$. Let $A_{k'}$ be the number of iterations required to find $\mathbf{x}$, where $\mathbf{x} \cdot \mathbf{q} \geq \tau$ and $\mathbf{x} \notin \mathbf{Q}_k$. Notice that $A_{k'}$ is a random variable. We have

$$\Pr[A_{k'} = m] = \left(\frac{j - |\mathbf{Q}| + k'}{j}\right)^{m-1} \times \frac{|\mathbf{Q}| - k'}{j}.$$

Let $z = 1 - \frac{|\mathbf{Q}|-k'}{j}$, then

$$E[A_{k'}] = \sum_{m=1}^{\infty} m z^{m-1}(1-z) = \frac{1}{1-z} \quad \left(\because \sum_{m=1}^{\infty} m z^{m-1} = \frac{1}{(1-z)^2}\right)$$
$$= \frac{j}{|\mathbf{Q}| - k'} = O(1),$$

since $j = O(n)$, $|\mathbf{Q}| = O(n)$, and $k' = O(k)$. Now it is clear that the expected number of iterations for having $|\mathbf{Q}_k| = k$ is $\sum_{k'=0}^{k-1} E[A_{k'}] = O(k)$. This theorem hence holds. □

We can achieve the same running time as this *expected* time, with at least constant probability.

Theorem 2. *Our algorithm correctly returns $\mathbf{Q}_k$ in $O(\log n + k)$ time with at least constant probability.*

Proof. Again assume that we have $\mathbf{Q}_k$ such that $|\mathbf{Q}_k| = k' < k$. Let B be an iteration that fails to obtain $\mathbf{x}$, where $\mathbf{x} \cdot \mathbf{q} \geq \tau$ and $\mathbf{x} \notin \mathbf{Q}_k$. Clearly, $\Pr[B] = 1 - \frac{|\mathbf{Q}|-k'}{j}$. Assume that we have a sequence of s failure iterations. This failure probability is bounded by

$$\Pr[B \cup \ldots \cup B] \leq \sum_{s} \Pr[B] = s\left(1 - \frac{|\mathbf{Q}| - k'}{j}\right)$$

from union bound. By considering $s(1 - \frac{|\mathbf{Q}|-k'}{j}) = c$ such that $c \in (0, 1]$ and $c = O(1)$, we have $s = O(1)$. Then, we see that our algorithm can obtain $\mathbf{x}$ with at least constant probability in $O(1)$ time. By repeating this sequence k times, this theorem is clear. □

Next, we consider the case where a higher success probability is required.

Theorem 3. *Our algorithm correctly returns $\mathbf{Q}_k$ in $O(k \log n)$ time with probability at least $(1 - \frac{1}{n})^k$.*

Proof. Consider a similar setting to that in the proof of Theorem 2. When we have a sequence of s iterations, the probability that there is at least one success iteration is $1-(1-p)^s$, where $p = \frac{|\mathbf{Q}|-k'}{j}$. We want to make it *high probability*, i.e., $1-1/n$. By setting $s = \log_{\frac{1}{1-p}} n$, this high probability is obtained. It is straightforward to see that $s = O(\log n)$ from the proof of Theorem 1. Then, by using the same rationale of the proof of Theorem 2, we can conclude that Theorem 3 also holds. □

Remark 1. *For $n \geq 10,000$ and $k \leq 100$ (which is a reasonable setting), $(1 - \frac{1}{n})^k \geq 0.99$. This means that our algorithm runs in at most $\tilde{O}(k)$ time in almost all cases.*

The above theorems demonstrate that our algorithm runs in $\tilde{O}(k)$ time and *theoretically outperforms the baseline algorithms employing the state-of-the-art fair search techniques.*

4 EXPERIMENT

This section reports our experimental results. All experiments were conducted on a a Ubuntu 18.04 LTS machine with 128GB RAM and 3.0GHz Core i9-10980XE CPU.

Datasets. We used the following four real datasets.

- Amazon-Kindle [16]: A rating dataset for books in Amazon Kindle. The number of items is 430,530.
- Amazon-Movie [16]: A rating dataset for movies in Amazon. The number of items is 200,941.
- MovieLens[4]: This is the MovieLens 25M dataset, and the number of items is 59,047.
- Netflix[5]: This is a rating dataset used in Netflix Prize, and the number of items is 17,770.

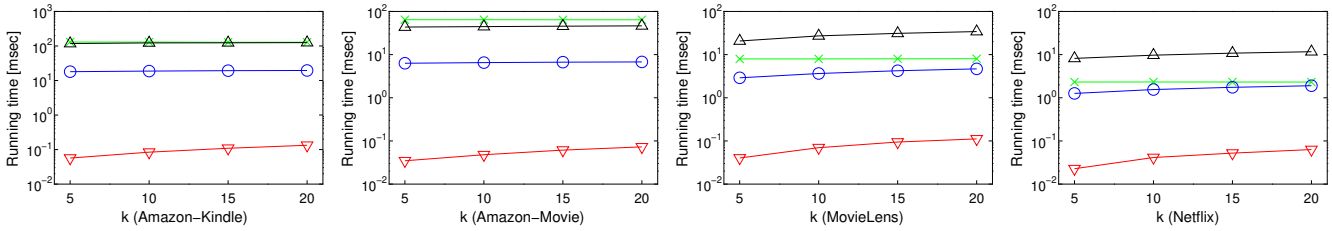
We used Matrix Factorization [12] to generate query (user) vectors and item vectors in an inner product space. The dimensionality of each vector was 200.

Competitors. We compared our algorithm with

- Linear-Scan: the algorithm introduced in Section 2 that runs in $O(n)$ time,
- NNS [10]: A state-of-the-art fair search algorithm that runs in $o(kn \log n)$ expected time, and

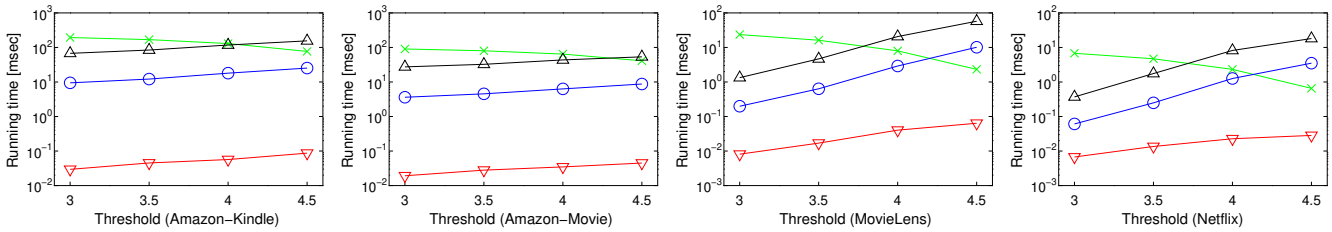
[3] We assume that $|\mathbf{Q}| \geq k$. By setting the maximum number of iterations of ② as $O(\log n)$, the worst case time of our algorithm still holds even if $|\mathbf{Q}| < k$.

[4] https://grouplens.org/datasets/movielens/
[5] https://www.cs.uic.edu/liub/Netflix-KDD-Cup-2007.html



(a) Amazon-Kindle (b) Amazon-Movie (c) MovieLens (d) Netflix

Figure 1: Impact of k (output size): "×" shows Linear-Scan, "○" shows NNS, "△" shows NNIS, and "▽" shows Ours.



(a) Amazon-Kindle (b) Amazon-Movie (c) MovieLens (d) Netflix

Figure 2: Impact of τ (threshold): "×" shows Linear-Scan, "○" shows NNS, "△" shows NNIS, and "▽" shows Ours.

- NNIS [9]: A state-of-the-art fair and independent search algorithm that runs in $o(kn \log^5 n)$ expected time.

All algorithms were implemented in C++ and compiled by g++ 7.5.0 with -O3 optimization.

All evaluated algorithms can provide a correct solution, i.e., they return $\mathbf{Q}_k$ such that $|\mathbf{Q}_k| = k$ and each $\mathbf{x} \in \mathbf{Q}_k$ satisfies $\mathbf{x} \cdot \mathbf{q} \geq \tau$. We therefore measured their running times and report the average times by using all user vectors[6]. We did not test the space cost of each algorithm, since it is guaranteed that our algorithm and Linear-Scan, which do not require additional data structures, are better than NNS and NNIS, which require $O(n^{1+\epsilon})$ space, where $\epsilon \in (0, 1)$, w.r.t. space cost. The default values of k and τ were respectively 5 and 4.0 (because the scale of ratings in the datasets is 1 to 5).

Impact of k. Figure 1 shows the results of our experiments varying the output size k. As k increases, our algorithm needs longer running time. This is reasonable, as its time complexity is linear to k. Since Linear-Scan needs $O(n)$ time, it has no impact of k. NNS and NNIS are also less sensitive to k.

We see that our algorithm is significantly faster than the other algorithms. When $k = 5$, for example, our algorithm is 318, 182, 72, and 56 times faster than the best of the competitors on Amazon-Kindle, Amazon-Movie, MovieLens, and Netflix, respectively. Furthermore, even when $k = 20$, our algorithm needs less than 0.15 [msec] in average for these four datasets. As noted in Section 2, this less than millisecond order is highly preferable in large-scale systems, clarifying the practical advantage of our algorithm.

[6]Amazon-Kindle, Amazon-Movie, MovieLens, and Netflix have 1,406,890, 2,088,620, 162,541, and 480,189 users, respectively.

Impact of τ. Figure 2 exhibits the results of our experiments varying the threshold τ. Notice that, as τ increases, $|\mathbf{Q}|$ decreases. Therefore, the running time of Linear-Scan decreases as τ increases. On the other hand, the other algorithms need longer running times as τ increases. We found that, as τ increases, the success probability (i.e., the probability of sampling $\mathbf{x} \in \mathbf{Q}$) decreases. (For instance, $|\mathbf{Q}|/j$ of our algorithm decreases for larger values of τ.) This situation requires a larger number of iterations, which leads to the longer running time. Nevertheless, our algorithm keeps outperforming the other algorithms, so our algorithm is still the first choice even when τ is large.

5 CONCLUSION

This paper formulated a new problem, namely the fair and independent inner product search (FI-IPS) problem. This problem can alleviate the drawback of the k-maximum IPS problem (i.e., skewed and static results), because FI-IPS returns k items *randomly* sampled from a set of items satisfying $\mathbf{x} \cdot \mathbf{q} \geq \tau$. This paper overcame the computational challenge of the FI-IPS problem, and we proposed a simple yet efficient algorithm that returns a correct solution in $O(\log n + k)$ expected time. We conducted experiments using real datasets and demonstrated that our algorithm significantly outperforms baselines.

ACKNOWLEDGMENTS

This research is partially supported by AIP Acceleration Research JPMJCR23U2 and JST CREST JPMJCR21F2, Japan.

REFERENCES

[1] Mustafa Abdool, Malay Haldar, Prashant Ramanathan, Tyler Sax, Lanbo Zhang, Aamir Manaswala, Lynn Yang, Bradley Turnbull, Qing Zhang, and Thomas

Legrand. 2020. Managing Diversity in Airbnb Search. In *KDD*. 2952–2960.
[2] Peyman Afshani and Jeff M Phillips. 2019. Independent Range Sampling, Revisited Again. In *SoCG*.
[3] Peyman Afshani and Zhewei Wei. 2017. Independent range sampling, revisited. In *ESA*, Vol. 87. 3.
[4] Daichi Amagata. 2023. Diversity Maximization in the Presence of Outliers. In *AAAI*.
[5] Daichi Amagata and Takahiro Hara. 2016. Diversified set monitoring over distributed data streams. In *DEBS*. 1–12.
[6] Daichi Amagata and Takahiro Hara. 2021. Reverse Maximum Inner Product Search: How to efficiently find users who would like to buy my item?. In *RecSys*. 273–281.
[7] Daichi Amagata and Takahiro Hara. 2023. Reverse Maximum Inner Product Search: Formulation, Algorithms, and Analysis. *ACM Transactions on the Web* (2023).
[8] Ashton Anderson, Lucas Maystre, Ian Anderson, Rishabh Mehrotra, and Mounia Lalmas. 2020. Algorithmic Effects on the Diversity of Consumption on Spotify. In *The Web Conference*. 2155–2165.
[9] Martin Aumuller, Sariel Har-Peled, Sepideh Mahabadi, Rasmus Pagh, and Francesco Silvestri. 2021. Fair near neighbor search via sampling. *ACM SIGMOD Record* 50, 1 (2021), 42–49.
[10] Martin Aumüller, Sariel Har-Peled, Sepideh Mahabadi, Rasmus Pagh, and Francesco Silvestri. 2022. Sampling a Near Neighbor in High Dimensions—Who is the Fairest of Them All? *ACM Transactions on Database Systems* 47, 1 (2022), 1–40.
[11] Martin Aumüller, Rasmus Pagh, and Francesco Silvestri. 2020. Fair near neighbor search: Independent range sampling in high dimensions. In *PODS*. 191–204.
[12] Wei-Sheng Chin, Bo-Wen Yuan, Meng-Yuan Yang, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. 2016. LIBMF: A Library for Parallel Matrix Factorization in Shared-memory Systems. *The Journal of Machine Learning Research* 17, 1 (2016), 2971–2975.
[13] Xinyan Dai, Xiao Yan, Kelvin KW Ng, Jiu Liu, and James Cheng. 2020. Norm-Explicit Quantization: Improving Vector Quantization for Maximum Inner Product Search. In *AAAI*. 51–58.
[14] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-scale Inference with Anisotropic Vector Quantization. In *ICML*. 3887–3896.
[15] Sariel Har-Peled and Sepideh Mahabadi. 2019. Near neighbor: who is the fairest of them all?. In *NeurIPS*. 13176–13187.
[16] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-class Collaborative Filtering. In *World Wide Web*. 507–517.
[17] Kohei Hirata, Daichi Amagata, Sumio Fujita, and Takahiro Hara. 2022. Solving Diversity-Aware Maximum Inner Product Search Efficiently and Effectively. In *RecSys*. 198–207.
[18] Kohei Hirata, Daichi Amagata, Sumio Fujita, and Takahiro Hara. 2023. Categorical Diversity-Aware Inner Product Search. *IEEE Access* 11 (2023), 2586–2596.
[19] Kohei Hirata, Daichi Amagata, and Takahiro Hara. 2022. Cardinality Estimation in Inner Product Space. *IEEE Open Journal of the Computer Society* 3, 01 (2022), 208–216.
[20] Xiaocheng Hu, Miao Qiao, and Yufei Tao. 2014. Independent range sampling. In *PODS*. 246–255.
[21] Zhengbao Jiang, Ji-Rong Wen, Zhicheng Dou, Wayne Xin Zhao, Jian-Yun Nie, and Ming Yue. 2017. Learning to Diversify Search Results via Subtopic Attention. In *SIGIR*. 545–554.
[22] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: fast and exact inner product retrieval in recommender systems. In *SIGMOD*. 835–850.
[23] Stanislav Morozov and Artem Babenko. 2018. Non-metric Similarity Graphs for Maximum Inner Product Search. In *NeurIPS*. 4721–4730.
[24] Hayato Nakama, Daichi Amagata, and Takahiro Hara. 2021. Approximate Top-k Inner Product Join with a Proximity Graph. In *IEEE Big Data*. 4468–4471.
[25] Gourab K. Patro, Lorenzo Porcaro, Laura Mitchell, Qiuyue Zhang, Meike Zehlike, and Nikhil Garg. 2022. Fair ranking: a critical review, challenges, and future directions. In *FAccT*. 1929–1942.
[26] Zhan Su, Zhicheng Dou, Yutao Zhu, Xubo Qin, and Ji-Rong Wen. 2021. Modeling Intent Graph for Search Result Diversification. In *SIGIR*. 736–746.
[27] Shulong Tan, Zhaozhuo Xu, Weijie Zhao, Hongliang Fei, Zhixin Zhou, and Ping Li. 2021. Norm Adjusted Proximity Graph for Fast Inner Product Retrieval. In *KDD*. 1552–1560.
[28] Yufei Tao. 2022. Algorithmic Techniques for Independent Query Sampling. In *PODS*. 129–138.
[29] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. 2015. LEMP: Fast Retrieval of Large Entries in a Matrix Product. In *SIGMOD*. 107–122.
[30] Hanxin Wang, Daichi Amagata, Takuya Makeawa, Takahiro Hara, Niu Hao, Kei Yonekawa, and Mori Kurokawa. 2020. A DNN-Based Cross-Domain Recommender System for Alleviating Cold-Start Problem in E-Commerce. *IEEE Open Journal of the Industrial Electronics Society* 1 (2020), 194–206.
[31] Long Xiang, Bo Tang, and Chuan Yang. 2019. Accelerating exact inner product retrieval by cpu-gpu systems. In *SIGIR*. 1277–1280.
[32] Long Xiang, Xiao Yan, Lan Lu, and Bo Tang. 2021. GAIPS: Accelerating Maximum Inner Product Search with GPU. In *SIGIR*. 1920–1924.
[33] Dong Xie, Jeff M Phillips, Michael Matheny, and Feifei Li. 2021. Spatial Independent Range Sampling. In *PVLDB*. 2023–2035.
[34] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. 2018. Norm-ranging lsh for maximum inner product search. In *NeurIPS*. 2952–2961.
[35] Han Zhang, Hongwei Shen, Yiming Qiu, Yunjiang Jiang, Songlin Wang, Sulong Xu, Yun Xiao, Bo Long, and Wen-Yun Yang. 2021. Joint learning of deep retrieval model and product quantization based embedding index. In *SIGIR*. 1718–1722.
[36] Jin Zhang, Qi Liu, Defu Lian, Zheng Liu, Le Wu, and Enhong Chen. 2022. Anisotropic Additive Quantization for Fast Inner Product Search. In *AAAI*. 4354–4362.