Efficient Approximate Maximum Inner Product Search over Sparse Vectors

Xi Zhao[1], Zhonghan Chen[1], Kai Huang[2], Ruiyuan Zhang[1], Bolong Zheng[3], Xiaofang Zhou[1]
[1]Hong Kong University of Science and Technology, Hong Kong
[2]Macau University of Science and Technology, Macau
[3]Huazhong University of Science and Technology, Wuhan, China
{xzhaoca,zchenhj}@cse.ust.hk,huangkai@must.edu.mo,zry@cse.ust.hk,bolongzheng@hust.edu.cn,zxf@cse.ust.hk

***Abstract*—The maximum inner product search (MIPS) problem in high-dimensional vector spaces has various applications, primarily driven by the success of deep neural network-based embedding models. Existing MIPS methods designed for dense vectors using approximate techniques like locality-sensitive hashing (LSH) have been well studied, but they are not efficient and effective for searching sparse vectors due to the near-orthogonality among the sparse vectors. The solutions to MIPS over sparse vectors rely heavily on inverted lists, resulting in poor query efficiency, particularly when dealing with large-scale sparse datasets. In this paper, we introduce SOSIA, a novel framework specifically tailored to address these limitations. To handle sparsity, we propose the SOS transformation, which converts sparse vectors into a binary space while providing an unbiased estimator of the inner product between any two vectors. Additionally, we develop a minHash-based index to enhance query efficiency. We provide a theoretical analysis on the query quality of SOSIA and present extensive experiments on real-world sparse datasets to validate its effectiveness. The experimental results demonstrate its superior performance in terms of query efficiency and accuracy compared to existing methods.**

***Index Terms*—Maximum Inner Product Search, Sparse Vectors**

I. Introduction

In recent years, there has been a significant surge in the availability of high-dimensional vector data (*i.e.,* vector representations of objects), primarily due to the proliferation of unstructured data and the remarkable success of deep neural network-based embedding models. These models have demonstrated their ability to embed various types of unstructured data, including videos [10], molecular structural information of drugs [8], and documents [29], into vectors. Subsequently, these vectors can be stored in a vector database to facilitate a wide range of applications, including personalized recommendations and information retrieval. As a result, the *Maximum Inner Product Search (MIPS) problem*, which aims to find a vector (*i.e.,* a representation of an object) from a dataset D that has the maximum inner product with a given query q, plays a crucial role in these applications. As many vectors generated by embedding models exhibit sparsity (*e.g.,* only 43 non-zero entries in the vector generated by the SPLADE model [15], [16]), the MIPS problem for sparse vectors has been widely studied [26], [38], [39], [48], [53], [55]. However, the "Dimensionality Curse" phenomenon makes exact MIPS computation in high-dimensional spaces computationally expensive. In this paper, we study the problem of *Approximate Maximum Inner Product Search (AMIPS)* over sparse vectors, which identifies the top-k "approximately" most relevant objects for the query q over the collection D consisting of sparse vectors. Formally, given a query point $q \in \mathbb{R}^d$, a positive integer k, and an approximation ratio $c \in (0, 1)$, the AMIPS over the sparse vectors D retrieves k points $x_i \in D$ $(1 \leq i \leq k)$ such that the inner product $q^\top x_i \geq c \cdot q^\top x_i^*$ where $x_i{}^*$ is the i^{th} exact MIPS point of q over D.

The AMIPS problem with sparse vectors presents a significant challenge: most vectors are near-orthogonal, which makes it difficult for MIPS algorithms designed for dense vectors to effectively operate on sparse vectors. In contrast to methods for the MIPS problem in dense datasets, existing studies [2], [6], [7], [9], [13], [14], [20], [35] tackle the AMIPS problem for sparse datasets by considering only the non-zero indices and values of each sparse data. These studies directly organize this data using inverted index-based methods [6], [7], [13]. Several evaluation strategies [6], [35], [40] have been developed to filter out certain points by estimating the inner product of two vectors, aiming to reduce the number of inner product computations. However, most of these evaluation strategies are not computationally cheaper than computing the exact inner product, and all points that share common non-zero entries with the query must be considered, resulting in a high query cost of $O(n)$ where $n = |D|$. Furthermore, these evaluation strategies often rely on the assumption that the vectors follow a specific distribution, which may not be applicable in the case of dynamic datasets commonly found in modern vector databases. Another limitation is that there is no theoretical guarantee that these methods offer a better query cost than $O(n)$, which hampers their performance on large-scale datasets. As a result, designing an efficient and effective method for the AMIPS problem over large-scale and dynamic sparse datasets, without any underlying constraints or assumptions, remains a significant challenge.

To effectively address the AMIPS problem, we start by considering a simplified version of the problem on a binary-valued dataset in $\{0, 1\}^{d'}$. This simplified version is equivalent to a set overlap search problem, where the overlap of two sets A and B (denoted as $|A \cap B|$) is equivalent to the inner product of their binary representations [46]. In particular, set overlap search is a type of set similarity search problem that is related to other set search problems such as Jaccard similarity search [24], [25] and set containment search [1], [5]. These related problems have mature algorithms for approximate or exact solutions [43], [54], [57], [58], including inverted list-based methods [57] and minHash-based methods [58], which greatly improve the efficiency of set overlap search without any constraints on the dataset and enable sublinear cost solutions. With this in mind, we can explore transforming a real-valued sparse dataset into a binary-valued dataset while preserving the inner product relationships in their original space. Consequently, the AMIPS problem can be completely solved by addressing a set overlap search problem. While some sketch-based methods like BinSketch [40] attempt to transform a sparse vector in $\mathbb{R}^d$ into a binary space for addressing the MIPS problem, they fail to preserve the inner product relationship without introducing significant distortion error.

Motivated by these observations, we propose a novel framework called SOSIA to address the AMIPS problem in a sparse vector space. In particular, we present the SOS (Sparse Vector to Set) transformation to convert the AMIPS problem on $\mathbb{R}^d$ into the AMIPS problem on a binary space $\{0,1\}^{d'}$, which can be solved using minHash-based methods. The core of SOS is a transformation function $\mathcal{T}:\mathbb{R}^d \rightarrow \{0,1\}^{d'}$ that constructs a binary sketch for each vector in the dataset and query set. Theoretical results demonstrate that the inner product between $\mathcal{T}(x_1)$ and $\mathcal{T}(x_2)$ serves as an unbiased estimator of the inner product between x_1 and x_2 for any two points in $\mathbb{R}^d$. This property enables us to tightly control the error introduced by the transformation. Additionally, $\mathcal{T}$ is a data-independent transformation and supports dynamic datasets. Moreover, SOS offers simplicity in deployment, making it compatible with disk-based, parallel, or distributed settings. This flexibility enhances its adaptability within modern vector databases, enabling efficient processing and storage. We further develop a novel minHash-based index, which can efficiently address the AMIPS problem in the transformed space and achieve high query performance in terms of efficiency and accuracy. In summary, we make the following contributions in the paper.

- We propose a novel framework called SOSIA to address the problem of AMIPS over sparse vectors. To address the challenges posed by sparsity, we develop the SOS transformation, which converts sparse vectors into a binary space while providing an unbiased estimator of the inner product between any two vectors.
- We further develop a novel minHash-based index that enhances the efficiency of answering queries in the AMIPS problem. This indexing method ensures a high query quality with a small query cost.
- We conduct extensive experiments on real-world sparse datasets to validate the effectiveness of SOSIA. The experimental results demonstrate that SOSIA outperforms competitors in terms of both query efficiency and accuracy.

The rest of the paper is organized as follows. Section II reviews related work on MIPS problem and set similarity search problem. Section III presents the problem setting and basic concepts. Section IV shows the details of the SOS sketch and SOSIA framework. Section V provides proofs of lemmas and the theoretical guarantee of SOSIA. Section VI presents experimental studies and Section VII concludes the paper.

II. Related Work

A. Maximum Inner Product Search over Dense Datasets

The MIPS problem has been extensively studied in the context of dense datasets. In low-dimensional spaces, exact solutions to the MIPS problem can be achieved using space-partitioning trees, such as the M-tree [27] and cone-tree [41]. But their performance rapidly degrades as the dimensionality d increases due to the curse of dimensionality. In high-dimensional spaces, MIPS problem is mainly solved approximately by locality-sensitive hashing (LSH) methods and graph based method [37], [55]. LSH based methods are the mainstream approximation method for solving the NNS and MIPS problems [3], [12], [19], [26], [39], [45], [55]. Since the inner product is not a metric, asymmetric transformations are applied to convert the MIPS problem into nearest neighbor search (NNS) ahead [3], [45], [47] , which aims to find a vector from a dataset that has the minimal distance with a given query under a metric. Then, the MIPS problem is reduced to an NNS problem. However, these transformations either bring distortion errors or result in an imbalance in the transformed dataset, and thus limit the query performance. FARGO [55] proposes a novel transformation named random XBOX transformation that addresses these two issues. Graph-based [33], [37], [50] and learning-based [11], [17], [44] methods are also proposed for MIPS problem. However, both of these methods incur significantly high training or construction costs, which restrict their applicability to large-scale datasets.

These methods [11], [37], [39], [55] are mainly designed for dense vectors, the case where the entries of each vector are almost surely non-zero. In this case, the data points from a real-world dataset (not a random dataset) are usually well-clustered and the local intrinsic dimensionality (LID) [21], [31], [56] of the dataset are much less than its dimensionality d, which make it possible to either partition the nearby data points into several hash buckets or organize the datasets by proximity graphs. Unfortunately, when the points are sparse with very few non-zero entries, these methods hardly port over successfully because the points are near-orthogonal in a sparse high-dimensional space.

B. Maximum Inner Product Search over Sparse Datasets

Traditional approaches for addressing the MIPS problem in sparse datasets heavily rely on inverted index [6], [7], [13], [32], [35], [36]. The inverted index are previously designed for answering set similarity search, especially set overlap search. An inverted index consists of many posting lists where each posting list represents a specific entry of a vector and stores the point IDs and their non-zero values in this entry. By considering the inner product as a weighted set overlap, the problem can be answered exactly by traversing the posting lists of query's non-zero entries because the inner product is the sum of contributions in all entries. Such a naive approach is simple but inefficient for large-scale datasets since nearly all data points are required to be computed in the worst case. Therefore, many evaluation strategies are designed for filtering some points by quickly computing their inner product upper bounds to the query. These evaluation strategies can be categorized into *Document-at-a-time* (DAAT) strategies and *Term-at-a-time* (TAAT) strategies. DAAT strategies, such as WAND, BMW, and VBMW [6], [13], [35], estimate the upper bound of the inner product of a single point appearing in the posting list of a query term before moving to the next point. The point whose upper bound is less than the current found largest inner product will be filtered out; otherwise, we compute its real inner product to the query and update the current largest inner product. The efficiency of these methods depends on how tight an upper bound is. WAND [6] adopts the maximal nonzero values in each posting list as an upper bound of this posting list. Then, the inner product upper bound of a point is estimated by accumulating the sum of upper bounds in the common posting lists of this point and query point. To obtain a tighter upper bound, BMW [13] and VBMW [35] organize the terms in a posting list by several smaller-size blocks and define an upper bound for each block rather than the entire posting list, which greatly reduces the difference between the upper bound and the exact values and thus make it possible to filter out more points. In addition, some methods represent sparse vectors by the sketches such that the inner product of sketches approximates the inner product of original vectors [2], [9], [20]. Then, we estimate the upper bounds of the inner product quickly by computing the inner product between two sketches. TAAT strategies, such as SINNAMON [7], process query terms one by one and accumulate a partial inner product as the contribution of each query term is computed. To obtain a tight estimate of the exact inner product and facilitate filtering some points, SINNAMON accesses the posting lists with the decreasing order of the query term's value since a large value has a higher impact on the inner product. When accessing the posting lists, SINNAMON also adopts a low-dimensional sketch like that in BinSketch [40] for estimating the inner product of two points by accessing only a part of posting lists. Although these methods effectively reduce the number of full computations, the computational cost associated with computing these upper bounds remains substantial, often comparable to that of computing the exact inner product. Moreover, in most cases, it is necessary to compute the upper bounds for all points that share common non-zero entries with the query, resulting in a query cost that is not significantly better than $O(n)$.

III. Preliminaries

In this section, we introduce the AMIPS problem and preliminaries. Frequently used notations are in Table I.

TABLE I: List of Key Notations.

Notation	Description
$\mathbb{R}^d$	d-dimensional vector space
$\mathcal{D}$	The dataset
n	The cardinality of dataset
x	A data point
l	The base of SOS transformation
$x.ind$	x's non-zero indices
$x.val$	x's non-zero values
q	A query point
$q^\top x$	The inner product between q and x
$\mathcal{N}_I(q)$	The point that has the largest inner product to q in $\mathcal{D}$
$\lVert x\rVert$	The L2 norm of a point x in $\mathbb{R}^d$
$\lvert x\rvert, \lvert x_b\rvert$	The L1 norm of a point x in $\mathbb{R}^d$ or a point x_b in $\{0,1\}^d$
A, B	A set

A. Problem Definition

Definition 1 (Maximum Inner Product Search)**.** *Given a dataset $\mathcal{D} = \{x_1, x_2, \cdots, x_n\} \subseteq \mathbb{R}^d$ with $|\mathcal{D}| = n$ and a query point $q \in \mathbb{R}^d$, the maximum inner product search (MIPS) returns a point $\mathcal{N}_I(q) = x^* \in \mathcal{D}$ such that*

$$\mathcal{N}_I(q) = \arg\max_{x\in\mathcal{D}} q^\top x. \tag{1}$$

Definition 2 (Approximate Maximum Inner Product Search)**.** *Given a dataset $\mathcal{D} \subseteq \mathbb{R}^d$ with $|\mathcal{D}| = n$, a query point q, an approximation ratio $c \in (0,1)$ and a positive integer k, the approximate maximum inner product search ((c,k)-AMIPS) returns k points $x_1, \ldots, x_k$ that are sorted in descending order w.r.t. their inner products to q. If x_i^* is the i-th maximum inner product of q in $\mathcal{D}$, it satisfies $q^\top x_i \geq c \cdot q^\top x_i^*$.*

Remark 1. *We denote c-AMIPS problem as the (c,k)-AMIPS problem at $k = 1$.*

For a sparse vector $x \in \mathbb{R}^d$, we only record its non-zero information as a tuple $x = \langle x.ind, x.val\rangle$ where $x.ind = \{j \mid x[j] \neq 0\}$ is the non-zero indices and $x.val = \{x[j] \mid x[j] \neq 0\}$. For convenience, all indices begin at 0 rather than 1. We use $x.nnz$ to denote the size of $x.ind$ and $x.val$ and use $x.ind[j]$ and $x.val[j]$ to denote the j-th non-zero entry in x, *i.e.*, $x[x.ind[j]] = x.val[j]$. When x is a binary-valued data, it can be represented by only $x.ind$. Since $x.ind$ is a set, we call the vector x as the binary representation of the set $x.ind$ and call $x.ind$ as the sketch of x. The inner product of two binary vectors is related to the set overlap.

Definition 3 (Set Overlap)**.** *The overlap of two sets A and B is the size of their intersection,* i.e., $s_O(A,B) = |A \cap B|$.

We employ the following fact to establish the connection between inner product and set overlap.

Fact 1. *For two sets $A, B \subset U = \{e_1, e_2, \ldots, e_d\}$, the binary representation of A and B are two vectors $a, b \in \{0,1\}^d$ and $a^\top b = |A \cap B|$.*

To further illustrate the above conceptions, let us consider the example 1.

Example 1. *Consider the dataset D shown in Fig. 1 where $D = \{x_0 = (0,0,0.7,0,0)$, $x_1 = (0,0.2,0,0,0.3)$, $x_2 =$*

$(0, 0.5, 0, 0, 0), x_3 = (0.6, 0, 0.1, 0, 0.3)\} \subset \mathcal{R}^5$. x_2 can be represented as $x_1 = \langle[1, 4], [0.2, 0.3]\rangle$, where $x_1.ind = [1, 4]$ and $x_1.val = [0.2, 0.3]$. Similarly, a query point $q = (0, 0.2, 0, 0, 0.5)$ can be represented as $q = \langle[1, 4], [0.2, 0.5]\rangle$. The exact MIPS result for this query is x_1 ($q^\top x_1 = 0.19$). When $c = 0.5$ and $k = 2$, the (c, k)-AMIPS returns any two points in x_1, x_2, x_3 as correct results. Then, given a binary vector $x_b = \{0, 1, 0, 0, 1\}$, we can represent it as a set $\{[1, 4]\}$.

Since a binary vector x is totally equivalent to the set $A = x.ind$. We do not distinguish between a binary vector and a set longer in this paper.

B. Jaccard Similarity and MinHash

Definition 4 (Jaccard Similarity)**.** *The Jaccord Similarity of two sets A and B is defined as $s_J(A, B) = |A \cap B|/|A \cup B|$.*

A basic observation between Jaccard similarity and set overlap is

$$s_J(A, B) = \frac{s_O(A, B)}{|A| + |B| - s_O(A, B)}, \tag{2}$$

Therefore, given the Jaccard similarity between two sets, we can efficiently estimate their overlap. Since Jaccard similarity is normalized to the range [0, 1], it is easier to estimate using techniques like MinHash, as compared to directly estimating set overlap.

Definition 5 (MinHash)**.** *Given a universal collection of sets U, a family of minHash functions $\mathcal{H} = \{h : U \to Z\}$ maps the members of U into integers such that for any two sets $A, B \in U$, the collision probability of A and B equals to their Jaccard similarity,* i.e.,

$$\Pr[h(A) = h(B)] = s_J(A, B). \tag{3}$$

The goal of minHash is to estimate the Jaccard similarity between a query set and the set in a collection $\mathcal{D} \subset U$ quickly without explicitly computing the Jarccard. By adopting multiple minHash functions $h_0, h_1, \cdots, h_{m-1}$ from H, we can compute an unbiased estimation of the Jaccard similarity of two sets by counting the number of their collisions among m minHash functions.

C. Locality Sensitive Hashing index

When answering set Jaccard similarity search problem via a number of MinHash functions, a common strategy is to build Locality Sensitive Hashing (LSH) index. LSH function is a kind of hash functions based on specific metric, which is defined as:

Definition 6 (Locality Sensitive Hashing (LSH) [19], [52])**.** *Given a metric space $\mathcal{M} = (X, dist)$ where X is a collection of points and $dist$ is a function $dist : X \times X \to \mathbb{R}$, a distance r and an approximation ratio $c > 1$, a family of hash functions $\mathcal{H} = \{h : X \to Z\}$ is called (r, cr, p_1, p_2)-locality-sensitive, if for $\forall o_1, o_2 \in X$, it satisfies both conditions below:*

- *If $dist(o_1, o_2) \le r, \Pr[h(o_1) = h(o_2)] \ge p_1$;*
- *If $dist(o_1, o_2) > cr, \Pr[h(o_1) = h(o_2)] \le p_2$,*

where $h \in \mathcal{H}$ is chosen at random, p_1, p_2 are collision probabilities and $p_1 > p_2$.

MinHash functions are locality-sensitive based on the Jaccard distance, which is defined as:

Definition 7 (Jaccard Distance [28])**.** *The Jaccord distance of two sets A and B is $\delta_J(A, B) = 1 - s_J(A, B)$.*

Jaccard distance is a metric that satisfies the triangle inequality [28]. In the metric space $\mathcal{M} = (U, \delta_J)$, a minHash function is $(r, cr, 1 - r, 1 - cr)$-locality-sensitive for any $0 < r < 1/c$. We can adopt LSH index to answer an approximate Jaccard similarity search, like other nearest neighbor search (NNS), such as in Euclidean space. An LSH function is insufficient to answer NNS well since p_2 is too large in an LSH function such that there are many false negative points, *i.e.,* the points are too far away from the query, but still have the same hash value. (K, L)-LSH index is a common approach to address this problem [19], [52]. In the (K, L)-LSH index, to reduce the false negative ratio p_2, we usually concatenate K LSH functions $h_0, h_1, \cdots, h_{K-1} \in \mathcal{H}$ as

$$g(o) =< h_0(o), h_1(o), \cdots, h_{K-1}(o) >$$

and the points that agree on all K LSH functions are regarded as collisions and mapped into the same hash bucket. In the query phase, we only check the points that collide with the query point q. In this strategy, the false negative ratio are decreased as p_2^K, which greatly reduce the number of points to be checked. On the other hands, the true positive ratio are decreased from p_1 into p_1^K, which will also reduce the probability that a real nearest neighbor are found and degrade the query accuracy. A practical approach to address this issue is to repeat such a procedure L times independently to obtain enough candidate points in total L hash buckets. By properly choosing the values of K and L, an approximate nearest neighbor search can be solved in sublinear query cost with quality guarantee.

IV. The SOSIA framework

In this section, we present our framework called SOSIA to address the (c, k)-AMIPS problem in a sparse vector space, as depicted in Fig 1. SOSIA involves several key steps. Firstly, SOSIA employs the SOS transformation (refer to Section IV-A) to convert points in the sparse dataset into binary vectors. These binary vectors are considered as sets, which are mapped into a hash bucket in each of m hash tables using m independent MinHash functions (refer to Section IV-B). When a query q is received, it is also transformed into a binary vector q_b using the SOS transformation. Subsequently, it performs a nearest neighbor search based on Jaccard distance by utilizing the hash tables to obtain T candidates for the query q (refer to Section IV-C). During the query processing, we notice that the existing minHash LSH indexes cannot handle the issue that the collision probability varies greatly with the query points so that some queries obtain too many candidates while other queries collide with a few points. To address this

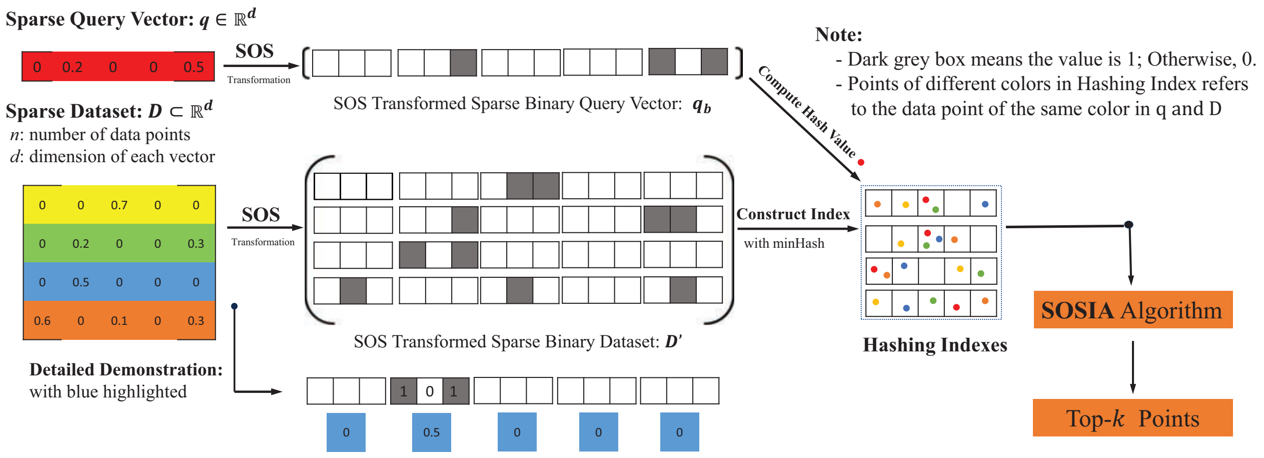

Sparse Query Vector: $q \in \mathbb{R}^d$
0 0.2 0 0 0.5
SOS
Transformation
SOS Transformed Sparse Binary Query Vector: q_b
Compute Hash Value
Note:
- Dark grey box means the value is 1; Otherwise, 0.
- Points of different colors in Hashing Index refers to the data point of the same color in q and D
Sparse Dataset: $D \subset \mathbb{R}^d$
n: number of data points
d: dimension of each vector
0 0 0.7 0 0
0 0.2 0 0 0.3
0 0.5 0 0 0
0.6 0 0.1 0 0.3
SOS
Transformation
Construct Index
with minHash
SOS Transformed Sparse Binary Dataset: D'
Hashing Indexes
SOSIA Algorithm
Top-k Points
Detailed Demonstration:
with blue highlighted
1 0 1
0 0.5 0 0 0

Fig. 1: SOSIA Framework

problem, a novel counting and refinement strategy is designed to find adaptively no more than T candidates for different query points efficiently.

A. SOS Transformation

Given a base l, SOS transformation converts a point x in the dataset $\mathcal{D} \subset \mathbb{R}^d$ into a binary vector x_b in $\{0,1\}^{d'}$ where $d' = d \cdot l$, *i.e.*,SOS is a transform $\mathcal{T} : \mathbb{R}^{+d} \to \{0,1\}^{d'}$. Each entry $x[i]$ in $\mathbb{R}^d$ is mapped to l entries at the range $x_b[il : il + l - 1]$ in $\{0,1\}^{d'}$ independently. The larger $x[i]$ is, the more likely the entries of $x_b[il : il + l - 1]$ are to be 1. To be simplified, we assume the maximal entries in $\mathcal{D}$ is 1; otherwise, we normalized $\mathcal{D}$ to make its maximal entries

$$M = \max_{x \in \mathcal{D}} \max_{0 \le i < d} x[i]$$

to be 1, which does not effect the MIPS result of any query point. The detail of SOS is shown in Alg. 1. For a non-zero entry $x[x.ind[j]] = x.val[j]$, each of l entries in $x_b[il : il + l - 1]$ will be set as 1 with the probability $x.val[j]$ (Line 5). The output of Alg. 1 is a binary vector in $x_b \in \{0,1\}^{d \cdot l}$. We take the datasets shown in Fig. 1 as an example to illustrate the details of the SOS transformation.

Algorithm 1: SOS Transformation

```
Input: A normalized sparse point $x \in \mathbb{R}^d$ and a
       positive integer $l$
Output: A binary vector $x_b \in \{0,1\}^{d \cdot l}$
1 $S \leftarrow \varnothing$;
2 for $j \leftarrow 0$ to $x.nnz$ do
3     for $j \leftarrow x.ind[j] \cdot l$ to $x.ind[j] \cdot l + l$ do
4         Genarate a random variable $r$ from the uniform
          distribution $U(0,1)$;
5         if $r < x.val[j]$ then
6             $S \leftarrow S \cup \{j\}$;
7 return $S$;
```

Example 2. *Let us consider the SOS transformation with the base $l = 3$ for the points shown in Fig. 1. We transform $x_0 = (0, 0, 0.7, 0, 0) \in \mathcal{D}$ into $\mathcal{T}(x_0) \in \{0,1\}^{15}$. Specifically, we loop only through non-zero entries ($x_0.ind$) of x_0, which is* 2 *in this simple case. In index* 2 *of x, for each of its three corresponding entries* $6, 7$ *and* 8 *in $\mathcal{T}(x_0)$, we generate a random variable r following an uniform distribution $U(0,1)$. If the r is less than its entry value* (0.7 *in this case), we add the index to S, which is the output set containing all non-zero index for $\mathcal{T}(x)$. For example, if we generate three random numbers* 0.8, 0.3 *and* 0.4 *for these three indices* $6, 7, 8$, *respectively.* 7 *and* 8 *will be put into S since $0.8 > 0.7$, $0.3 < 0.7$ and $0.4 < 0.7$. Finally, the output of SOS is $\mathcal{T}(x_0) = \{7, 8\}$, which is equivalent to the vector* $(0,0,0,0,0,0,0,1,1,0,0,0,0,0,0)$. *Similarly, the point $x_1 = (0, 0.2, 0, 0, 0.3)$ can be transformed into $\mathcal{T}(x_1) = \{5, 12, 13\}$.*

By adopting SOS to process the query q and each point x in $\mathcal{D}$, we can answer the AMIPS problem in $\mathcal{D}$ by considering the AMIPS of $\mathcal{T}(q)$ in a binary-valued dataset $\mathcal{D}' = \{\mathcal{T}(x) | x \in \mathcal{D}\}$ since $\mathcal{T}(q)^\top \mathcal{T}(x)$ satisfies the following lemma:

Lemma 1. *Given a query point q and a data $x \in \mathcal{D}$ where $q_b = \mathcal{T}(q)$ and $x_b = \mathcal{T}(x)$, then $q_b^\top x_b / l$ is an unbiased estimation of $q^\top x$.*

The proof of this lemma is provided in Section V-A. This lemma guarantees that SOS will not incur any distortion error in terms of the inner product relationship of points in the original space, based on which we can estimate the value of $q^\top x$ as tight as desired, *i.e.*, make $q_b^\top x_b / l$ be as close to $q^\top x$ as possible. Intuitively, enlarging l forces $q_b^\top x_b / l$ to be closer to $q^\top x$. We formulate this conclusion by computing the variance of $q_b^\top x_b / l$.

Lemma 2. *The variance of $q_b^\top x_b / l$ is*

$$\mathrm{Var}[q_b^\top x_b / l] = \frac{1}{l} \sum_{j=0}^{d-1} q[j]x[j](1 - q[j]x[j]).$$

The proof of this lemma is provided in Section V-A. Since the value of $\sum_{j=0}^{d-1} q[j]x[j](1-q[j]x[j])$ depends only on the q and x themselves, the variance of $q_b^\top x_b/l$ can be reduced by enlarging l, which allows us to estimate the value of $q^\top x$ more tightly by choosing a larger base l.

Algorithm 2: Indexing in SOSIA

```
Input: A normalized sparse dataset $\mathcal{D}$, a query point $q$
       and an approximate ratio $c$
Output: Hash Tables
1  Determine the value of $m$ based on Eq. 4;
2  Generate $m$ minHash functions $h_1, h_2, \dots, h_m$;
3  $\mathcal{B}[1:m] \leftarrow \varnothing$;
4  for $x \in \mathcal{D}$ do
5      $x_b = \mathcal{T}(x)$;
6      $id \leftarrow x_b$'s point id;
7      for $j \leftarrow 0$ to $m-1$ do
8          Insert $\langle h_j(x_b), \langle id, |x_b|\rangle\rangle$ into $\mathcal{B}[j]$;
9  for $j \leftarrow 0$ to $m-1$ do
10     for a hash bucket $B$ in $\mathcal{B}[j]$ do
11         Sort the tuples $\langle id, |x_b|\rangle$s in $B$ by the
           descending order of $|x_b|$;
12 return $\mathcal{B}$;
```

B. Indexing in SOSIA

After obtaining transformed dataset $\mathcal{D}' = \{x_b = \mathcal{T}(x) | x \in \mathcal{D}\}$, we choose m minHash functions and compute the hash values for each points x_b in $\mathcal{D}$. Then, m independent minHash indexes are built for the subsequent search in $\mathcal{D}'$. Unlike (K, L)-LSH indexes that build L numbers of K-dimensional hash tables, SOSIA builds m one-dimensional hash tables to facilitate determining the collision conditions based on different query points adaptively and dynamically. Algorithm 2 describes the indexing phase of SOSIA. First, we need to determine the number of hash functions, m, for answering the (c, k)-AMIPS problem. A larger m enables us to find higher-quality candidates but brings a higher query cost and space consumption. Based on our query strategy, we choose the smallest m that guarantees a correct (c, k)-AMIPS result with high probability, m is computed by

$$\begin{cases} t = \left(\dfrac{\sqrt{c}+1}{2}\right)^2 \\ m = \dfrac{3c(1-c\gamma)(\sqrt{t/c}-t\gamma)^2}{\gamma(t-\sqrt{tc})^2} \cdot \ln \dfrac{2n}{T}, \end{cases} \tag{4}$$

where $\gamma \in (0, 1)$ is a parameter to control the minimal inner product to be searched in our index and T is the maximal number of points to be checked. Such an m guarantees that SOSIA index returns a correct c-AMIPS with a constant probability $1/2 - 1/e$ when $q^\top x^* \geq \gamma(|q| + |x|)$ where $x^* = \mathcal{N}_I(q)$ is the maximal inner product result of q in $\mathcal{D}$, as analyzed in Section V-B. The reason why introducing a γ is as following: Let $s_I^* = q^\top \mathcal{N}_I(q)$ be the maximal inner product, when s_I^* is too small compared to the norm of q, any point in the dataset will not be similar to the query point q in terms of the inner product, making it impossible to distinguish the points in the dataset via minHash. So, we only guarantee a correct (c, k)-AMIPS result when Then, m minHash LSH indexes are adopted for indexing the points x_b transformed by SOS. For a term $\langle key, value\rangle$ stored in a hash table, the *key* is the hash value of the point x_b and the *value* is a tuple consisting of the x_b's identity and norm. Storing the norm $|x_b|$ is for restoring the value of $s_O(q_b, x_b)$ by $s_J(q_b, x_b)$ since minHash is designed for estimating $s_J(q_b, x_b)$. As such,

$$s_O(q_b, x_b) = \frac{|q_b| + |x_b|}{1 + 1/s_J(q_b, x_b)}. \tag{5}$$

This equation also indicates that a larger $|x_b|$ are more likely to yield a larger $s_O(q_b, x_b)$. So, in each hash bucket *i.e.,* the hash terms with the same keys, we will sort the points by the descending order of their norms to quickly find the points with a large norm at first in the query phase, as shown in Line 11.

C. Search in SOSIA

When a query q comes, we also transform it into the binary vector q_b via SOS and then compute its m hash values. To find the candidate points with a large set overlap to q_b, a novel search algorithm based on collision counting is designed to quickly find enough number of high-quality candidates with a small size of indexes. The search strategy in SOSIA is quite different compared to the (K, L)-LSH indexes. In (K, L)-LSH indexes, there are L numbers of K-dimensional hash tables and the points that collide with the query point in at least one of L hash tables will be considered as a candidate and will be checked. For each data point, the collision condition in each K-dimensional hash tables is that its K hash functions are all evaluated to be the same with the hash value of query points. While such a strategy is efficient to answer an approximate nearest neighbor search problem in Euclidean space, it is inefficient and infeasible to answer an approximate Jaccard similarity problem due to the difference between the minHash function and the LSH functions under Euclidean space.

A typical LSH defined in the Euclidean space $\mathbb{R}^d$ is as follows [12]:

$$h^{L2}(x) = \left\lfloor \frac{a^\top x + b}{w} \right\rfloor, \tag{6}$$

where a is a d-dimensional vector whose entries are chosen independently from the standard normal distribution, w is a pre-defined integer and b is a number chosen uniformly from $[0, w)$. The flexibility of such a hash function is that we can tune the collision probability for any two points by varying the value of w. A larger w enables a greater number of points agreeing on a hash value and enlarging the collision probability. Moreover, regardless of the distance between two points, they can be mapped into the same hash bucket by selecting a sufficiently large value for the parameter w. This enables LSH functions in Euclidean space to solve the approximate nearest neighbor search problem with varied distances between query points and their exact nearest neighbor.

```
Algorithm 3: Search in SOSIA
Input: A dataset $\mathcal{D}$, the SOSIA index $\mathcal{B}$, $T$, $c$ and $k$
Output: At most $k$ points in $\mathcal{D}$
1  $q \leftarrow \frac{q}{\max q[i]}$, $q_b \leftarrow \mathcal{T}(q)$;
2  $l \leftarrow$ the base of SOS; (see Alg. 1)
3  Determine the value of $t$ based on Eq. 4;
4  $I_0 \leftarrow \sqrt{\|q\|}$, $I_t \leftarrow 0$,$I \leftarrow I_0$;
5  Compute $q_b$'s minHash values $h_1(q_b), \dots, h^*_m(q_b)$;
6  $H \leftarrow \varnothing$,$R \leftarrow \varnothing$,$cost \leftarrow 0$;
7  while $I_t < cI$ and $cost < T + k$ do
8      $x_b \leftarrow$ the first unseen point with largest norm in
       $\mathcal{B}[j].at(h_j(q_b)), (1 \leq j \leq m)$;
9      if no points is unseen then
10         Break;
11     $\alpha \leftarrow |\{j | h_j(q_b) = h_j(x_b)\}|$;
12     $s_O \leftarrow \dfrac{|q_b| + |x_b|}{1 + m/\alpha}$;
13     Update$(x, s_O)$;
14 while $I_t < cI$ and $cost < T + k$ do
15     if $H$ is empty then
16         Break;
17     $\langle x, s_O \rangle \leftarrow H.top()$;
18     if $s_O/l < tI$ then
19         $I \leftarrow cI$, continue;
20     $H.pop()$;
21     $x \leftarrow$ the point reprsented by $x_b$ in $\mathcal{D}$;
22     $I_x = q^\top x$;
23     $cost \leftarrow cost + 1$;
24     $e \leftarrow \{x, I_x\}$;
25     $R \leftarrow R \cup \{e\}$;
26     if $|R| = k + 1$ then
27         Remove the element with the smallest $I_x$ in $R$;
28         $I_t \leftarrow \min\{e.I_x | e \in R\}$;
29 return $R$;
```

```
Algorithm 4: Update$(x, s_O)$
Input: $x_b$, $q$, $l$, $t$, $I$, $cost$, $c$, $k$, $H$, and $R$
Output: $k$ points in $\mathcal{D}$
1  if $s_O/l \geq tI$ then
2      $x \leftarrow$ the point reprsented by $x_b$ in $\mathcal{D}$;
3      $I_x = q^\top x$;
4      $cost \leftarrow cost + 1$;
5      $e \leftarrow \{x, I_x\}$;
6      $R \leftarrow R \cup \{e\}$;
7  else
8      $x \leftarrow$ the point reprsented by $x_b$ in $\mathcal{D}$;
9      $H \leftarrow H \cup \{\langle x, s_O \rangle\}$;
10 if $|R| = k + 1$ then
11     Remove the element with the smallest $I_x$ in $R$;
12     $I_t \leftarrow \min\{e.I_x | e \in R\}$;
```

However, these conditions are not satisfied by a minHash function. A minHash function h, defined for the a collections of sets $\mathcal{D}'$, is usually based on a random permutation of the elements of the universal set $U = \cup_{A \in \mathcal{D}'} A$. Let $P = \{e_1, e_2, \dots, e_d\}$ be a random permutation of U, $h(A)$ is the minimal position of A's elements in P, *i.e.*, $h(A) = \min\{j \mid P[j] \in A\}$. Since no hyper-parameter like w in h^{L2} is used for tuning the collision probability of two points, we are unable to enlarge the collision probability of two sets when their Jaccard similarity is fixed. So, it becomes extremely difficult to choose a static K and L for varied query points. Specifically, denote $\mathcal{N}_J(q)$ as the set that has the maximal Jaccard similarity to the query q in the dataset $\mathcal{D}$, a large K enables us to find the good results for those queries q having a large $s_J(q, \mathcal{N}_J(q))$ but make the queries q having a small $s_J(q, \mathcal{N}_J(q))$ hard to find any candidate. While a small K enables us to find the good results for those queries q having a small $s_J(q, \mathcal{N}_J(q))$ but make the queries q having a small $s_J(q, \mathcal{N}_J(q))$ find too many candidates and increase the query cost. To address this dilemma, we adopt a dynamic counting based strategy for adaptively addressing varied query point.

Alg. 3 describes the search processing in SOSIA. Although the LSH method cannot solve a c-AMIPS problem directly, it can solve the problem by resolving a sequence of threshold-based (I, c)-MIPS problem by decreasing the inner product threshold I [23], [55]. Therefore, to answer a c-AMIPS problem, we execute a sequence of (I, c)-MIPS problems as defined below.

Definition 8 (((I, c)-MIPS [30]). *Given a threshold I and an approximation factor $c < 1$, the (I, c)-MIPS for a query point q over dataset $\mathcal{D}$ returns the following result:*

1) *If at least one point x exists in $\mathcal{D}$ such that $q^\top x > I$, it returns such a point x' in $\mathcal{D}$ that $q^\top x' > c \cdot I$;*
2) *If no such point x exists in $\mathcal{D}$, it returns nothing.*

So, we execute a sequence of (I, c)-MIPS problems by beginning with $I = \sqrt{\|q\|}$ (Line 4), which is the maximal value of $q^\top \mathcal{N}_I(q)$ since all entries have been normalized into $[0, 1]$ in $\mathcal{D}$. I_t is used for recording the current found k-th largest inner produce value. We initialize a max-heap H to store the all counted points and a min-heap R with size no more than k to store the found best k results so far. We divide the remaining search processing into two stages – the *counting stage* and *refinement stage*.

Counting stage (Lines 7-13). At this stage, we count the number of collisions between $q_b = \mathcal{T}(q)$ and points that lie in the same hash buckets as q_b. And in each hash bucket, we sort points in decreasing order by theirs' norm (Line 7). Since the points in a hash bucket has been sorted based on their norms, it only takes $O(\log n)$ time to obtain the first unseen point x_b that has the largest norm in q_b's m hash buckets by maintaining a max-heap that stores the first unseen point with largest norm in q_b's each hash bucket. For such a point x_b that collides α times with q_b, its estimated Jaccard similarity to q_b

is α/m and we will compute its set overlap s_O to q_b based on Eq. 5. Next, we updating the results by inserting x_b into either H or R based on the condition $s_O/l > tI$ where t is a threshold computed by Eq. 4. As shown in Lemma 3 defined below, s_O/l is an unbiased estimator of $q^\top x$. The proof of Lemma 3 is provided in Section V-B.

Lemma 3. *Given a set space X and a family of MinHash functions $\mathcal{H} = \{h : X \to Z\}$, $h_0, h_1, \ldots, h_{m-1}$ are m minHash functions that are drawn independently from $\mathcal{H}$. For two points $q, x \in \mathbb{R}^d$, denote $\alpha = |\{j|h_j(\mathcal{T}(q)) = h_j(\mathcal{T}(x))\}|$. If $\alpha > 0$, then*

$$s_O/l = \frac{|\mathcal{T}(q)| + |\mathcal{T}(x)|}{(1 + m/\alpha)l}, \tag{7}$$

is an unbiased estimator of $q^\top x$ where $\mathcal{T}$ is the SOS transformation with the base l.

Alg. 4 describes the details of updating the results: When $s_O/l > tI$, $q^\top x$ is believed to be larger than cI with a high probability. We then compute the value of $q^\top x$ and put it in the result set R. Otherwise, $q^\top x$ is believed to be smaller than cI with a high probability and we insert it into H without computing $q^\top x$ in this stage. We keep record the k-th largest found result so far as I_t when R has more than k points. The counting stage terminates when one of the following three conditions satisfies:

T1. All points, which lie in the same hash bucket (in each of m hash tables) as q_b, are counted (Line 9).
T2. I_t is no smaller than cI (Line 7).
T3. There are $T + k$ points x that $q^\top x$ is computed (Line 7).

When **T2** is satisfied, the algorithm successfully returns k correct (I, c)-MIPS results. Since the current $I = \sqrt{\|q\|}$ is the maximal possible inner product between q and any point in $\mathcal{D}$, the found results are correct (c, k)-AMIPS results. When $T3$ is satisfied, the algorithm has checked $T + k$ points but less than k satisfied (I, c)-MIPS result are found. In this case, let I_k^* be the k-th exact largest inner product between q and the point in $\mathcal{D}$, I_k^* is believed to be less than I_t/c^2 with high probability, *i.e.,* the algorithm has found correct (c^2, k)-AMIPS results. We will prove this statement in Section V-B. When **T1** is satisfied, the algorithm neither checks enough number of candidates, *i.e.,* $T + k$ candidates, nor has found k correct (I, c)-MIPS result. In this case, we requires to conduct another (cI, c)-MIPS result with decreasing the threshold from I to cI in the next *refinement stage*.

Refinement stage (Lines 14-28). In this stage, we attempt to extract more candidates x from H whose estimated inner product to q, s_O/l, exceed tI. If no such point x exists but the termination conditions **T2** and **T3** are not satisfied, we keep decreasing the inner product threshold from I to cI. The counting stage terminates when **T2**, **T3** and **T4** satisfies. **T2** and **T3** are same as that in the counting stage and **T4** is:

T4. The max-heap H becomes empty (Line 15).

When the *refinement stage* is terminated by **T4**, it indicates we fail to find enough number of candidates from m minHash indexes. In this case, it is uncertainty whether we have found correct (c^2, k)-AMIPS results. But by setting m based on Eq. 4, we ensure this case will happen only when $I_k^* < \gamma\sqrt{\|I_0\|}$ where γ is a parameter defined in Eq. 4 with a high probability. In other word, when I_k^* is larger than this threshold, we would have found it with a high probability in one of two stages. When the *refinement stage* is terminated by **T2** or **T3**, we guarantee a correct (c^2, k)-AMIPS result as analyzed in the *counting stage*. The correctness of these statements will be proven in Section V-B.

Example 3. *We take the dataset $\mathcal{D}$ shown in Fig. 1 as an example to illustrate the indexing of SOSIA and how to answer a $(2, 0.5)$-AMIPS in SOSIA. There are $m = 4$ minHash functions $h_0, \cdots, h_3$ that are designed for the transformed dataset $\mathcal{D}' = \{x_b = \mathcal{T}(x)|x \in \mathcal{D}\}$ and 4 hash tables are built based on their hash values. When the query q comes, we compute its hash values and find the points that collide with it among these 4 hash tables for answering an (I, c)-MIPS problem with $I = \sqrt{\|q\|} = 0.54$. In the counting stage, the first point to be checked is x_{b3} since it has the largest norm among all unseen points. To update the results in H and R, we find the collision number between q_b and x_{b3} is $\alpha = 1$ and the computed $s_O = 1.2$. Assume $t = 0.8$, $s_O/l = 0.4 \geq tI = 0.43$ is not satisfied and thus we store x_3 into H. Similarly, for the second point x_{b2}, $\alpha = 1$ and the computed s_O is 1.0. So, we store x_2 into H. For the third checked point is x_{b1} with $\alpha = 2$. The computed $s_O = 2.0$ and $s_O/l \geq tI$ is satisfied. We store x_1 into R. At this moment, all points that collide with q are counted and no enough point is found. Therefore, the query processing continues in the refinement stage with decreasing I from $I_0 = 0.54$ to $cI_0 = 0.27$. Then, x_3 are popped from H and we update it into $R = \{x_1\}$. Then, $|R| = 2$ and $I_t = \min\{q^\top x_1, q^\top x_3\} = 0.15$. Hence, $I_t < cI$ is violated and the query terminates based on the condition* **T2**. *The returned points are x_1 and x_3, which are the correct $(2, 0.5)$-AMIPS for q.*

V. Theoretical Analysis

In this section, we first provide the proofs of lemmas in Section IV, which builds the bridge between the inner product of two points and their overlap is the transformed space. Then, we demonstrate that SOSIA returns a correct (c^2, k)-AMIPS result with high probability in Section V-B, which ensures the correctness of our algorithm. Finally, we analyze the space and time complexity of our algorithm in SectionV-C.

A. Some Proofs

We first prove Lemma 1 that reveals $q_b^\top x_b/l$ is an unbiased estimator of $q^\top x$.

Proof. The expectation of $q_b^\top x_b/l$ is

$$\begin{aligned}\mathrm{E}[q_b^\top x_b/l] &= 1/l \cdot \mathrm{E}\left[\sum_{j=0}^{dl-1} q_b[j]x_b[j]\right] \\ &= 1/l \cdot \mathrm{E}\left[\sum_{j=0}^{d-1}\sum_{m=0}^{l-1} q_b[jl+m]x_b[jl+m]\right].\end{aligned}$$

Since all the entries of q_b and x_b are set as 1 independently, we have:

$$\mathrm{E}[q_b^\top x_b/l] = 1/l \cdot \sum_{j=0}^{d-1}\sum_{m=0}^{l-1} \mathrm{E}[q_b[jl+m]]\,\mathrm{E}[x_b[jl+m]].$$

$q_b[jl+m] = 1$ incurs when the random number $r \sim U(0,1)$ is less than $q[j]$. Since $0 \le q[j] \le 1$, $\mathrm{E}[q_b[jl+m]] = \Pr[q_b[jl+m] = 1] = q[j]$. Similarly, $\mathrm{E}[x_b[jl+m]] = x[j]$. So,

$$\mathrm{E}[q_b^\top x_b/l] = 1/l \cdot \sum_{j=0}^{d-1}\sum_{m=0}^{l-1} q[j]x[j] = \sum_{j=0}^{d-1} q[j]x[j] = q^\top x,$$

which indicates $q_b^\top x_b/l$ is an unbiased estimation of $q^\top x$. □

Then, we prove Lemma 2.

Proof. The variance of $q_b^\top x_b/l$ is

$$\begin{aligned}\mathrm{Var}[q_b^\top x_b/l] =& 1/l^2 \cdot \mathrm{Var}\left[\sum_{j=0}^{dl-1} q_b[j]x_b[j]\right] \\ =& 1/l^2 \cdot \sum_{j=0}^{d-1}\sum_{m=0}^{l-1} \mathrm{Var}\left[q_b[jl+m]x_b[jl+m]\right].\end{aligned}$$

The variable $q_b[jl+m]x_b[jl+m]$ follows a binomial distribution and $\Pr[q_b[jl+m]x_b[jl+m] = 1] = \Pr[q_b[jl+m] = 1] \cdot \Pr[x_b[jl+m] = 1] = q[j]x[j]$. So, $\mathrm{Var}\left[q_b[jl+m]x_b[jl+m]\right] = q[j]x[j](1-q[j]x[j])$ and

$$\begin{aligned}\mathrm{Var}[q_b^\top x_b/l] =& 1/l^2 \cdot \sum_{j=0}^{d-1}\sum_{m=0}^{l-1} q[j]x[j](1-q[j]x[j]) \\ =& 1/l \cdot \sum_{j=0}^{d-1} q[j]x[j](1-q[j]x[j]).\end{aligned}$$

□

Next, we prove Lemma 3 to demonstrate that we can obtain an unbiased estimator of $q^\top x$ by using minHash.

Proof. Let ν_j be a 0-1 random variable and $\nu_i = 1$ indicates $h_j(\mathcal{T}(q)) = h_j(\mathcal{T}(x))$. Then, we have $\mathrm{E}\,\nu_j = \Pr[\nu_j = 1] = \Pr[h_j(\mathcal{T}(q)) = h_j(\mathcal{T}(x))] = s_J(\mathcal{T}(q), \mathcal{T}(x))$. Since $\alpha = \sum_{j=0}^{m-1} \nu_j$ and $\nu_1, \cdots, \nu_j$ are independent, $\mathrm{E}\,\alpha = \sum_{j=0}^{m-1} E\nu_j = m \cdot s_J(\mathcal{T}(q), \mathcal{T}(x))$. And $\mathrm{E}\left[\frac{1}{(1+m/\alpha)l}\right] = \frac{1}{(1+m/\mathrm{E}\,\alpha)l} = (1 + 1/s_J(\mathcal{T}(q), \mathcal{T}(x)))/l$. Hence,

$$\begin{aligned}\mathrm{E}[s_O/l] =& \mathrm{E}\left[\frac{|\mathcal{T}(q)| + |\mathcal{T}(x)|}{\mathrm{E}[(1+m/\alpha)l]}\right] \\ =& 1/l \cdot \mathrm{E}\left[\frac{|\mathcal{T}(q)| + |\mathcal{T}(x)|}{1 + 1/s_J(\mathcal{T}(q), \mathcal{T}(x))}\right] \\ =& \mathrm{E}\left[\frac{s_O(\mathcal{T}(q), \mathcal{T}(x))}{l}\right],\end{aligned}$$

which equals to $q^\top x$ based on Lemma 1. □

B. Theoretical Guarantee of SOSIA

Next, we guarantee the correctness of SOSIA by showing it ensures a correct c^2-AMIPS result with certain conditions. First, we demonstrate the estimated s_O is proper for generating the candidates (Line 1 in Alg. 4) by the following lemma:

Lemma 4. *Given a query q, a point x, a threshold $0 < \gamma < 1$, m minHash functions, an inner product threshold I, an approximation ratio $0 < c < 1$, and a parameter $c < t < 1$, let $q_b = \mathcal{T}(q)$ and $x_b = \mathcal{T}(x)$. We define $s_O(q_b, x_b) = \frac{|q_b|+|x_b|}{1+m/\alpha}$, where α is the counting numbers between q_b and x_b in m minHash functions. For the following two events:*

- ***E1:*** *If $q^\top x \ge I$, then $s_O(q_b, x_b) > ltI$,*
- ***E2:*** *The number of point x, where $s_O(q_b, x_b) > ltI$ but $q^\top x < cI$, is less than T*

by setting t and w based on Eq. 4 and setting $l = \frac{4\cdot\max\{3c,2\}}{(1-\sqrt{c})^2} \cdot \ln n$, the probability that E1 occurs is at least $1-1/e$, and the probability that E2 occurs is at least $1/2$ when $q^\top \mathcal{N}_I(q) \ge \gamma(|q| + \mathcal{N}_I(q))$.

Proof. (Sketch.) We prove that $\Pr[s_O(q_b, x_b) > ltI \mid q^\top x < cI] < \frac{T}{2n}$ according to the Chernoff bound [1]. Then the expected number of points x , which satisfies $s_O(q_b, x_b) > ltI$ but $q^\top x < cI$, is $N_f \le n \cdot \frac{T}{2n} = T/2$. Based on the Markov's inequality, the probability that $E2$ occurs is at most $1 - T/N_f = 1/2$. Similarly, the probability that E1 occurs can be directly bounded. Due to the space limitation, we provide the detailed proof in our technical report [42]. □

Theorem 1. *When $q^\top \mathcal{N}_I(q) \ge \gamma(|q| + |\mathcal{N}_I(q)|)$, by setting parameters as that in Lemma 4, Alg. 3 returns a correct c^2-AMIPS result with a constant probability of at least $1/2 - 1/e$.*

Proof. We prove the theorem by assuming E1 in Lemma 4 holds, whose probability is at least $1-1/e$. When $q^\top \mathcal{N}_I(q) \ge \gamma(|q| + \mathcal{N}_I(q))$, the probability that $\mathcal{N}_I(q)$ does not collide with q in m hash tables is at most $1 - 1/e$, based on E1 in Lemma 4. In this case, Alg. 3 will not terminate by **T4**. If Alg. 3 terminates by **T2** ($I_t \ge cI$) when conducting (I_1, c)-MIPS search, it indicates the last round of (I, c)-MIPS with $I = I_1/c$ does not find correct results. So, $q^\top \mathcal{N}_I(q) < I_1/c$ based on E1 in Lemma 4; otherwise, $\mathcal{N}_I(q)$ must have been found when conducting (I_1, c)-MIPS search. In addition, since $I_t \ge cI_1 \ge c^2 \mathcal{N}_I(q)$, a correct c^2-AMIPS result have been found. If Alg. 3 terminates by **T3** when conducting (I_1, c)-MIPS search, it also indicates $q^\top \mathcal{N}_I(q) < I_1/c$, based on E1 in Lemma 4. In addition, $T + k$ number of points x fulfill $s_O(q_b, x_b) > lt \cdot I_1$ but none of them satisfies $q^\top x > cI$, which conflicts with E2. Since $\Pr[E2] \ge 1/2$, the probability that Alg. 3 terminates by **T3** is at most $1/2$.

□

C. Query Time and Space Consumption

We analyze the space and time complexity of SOSIA in this part. As shown in Eq. 4 and Lemma 4, $m = O(\log n)$ and $l = O(\log n)$. So the extra space consumption for storing the

[1]https://en.wikipedia.org/wiki/Chernoff_bound

TABLE II: Summary of Datasets

Datasets	n	φ_d	φ_q	d	Size (GB)
SPLADE1M*	1,000,000	126.3	49.1	30,109	0.97
SPLADE-F*	8,841,823	126.8	49.1	30,109	8.42
Yelp1M*	1,000,000	113.9	26.5	30,522	0.88
Yelp-F*	6,990,280	113.9	26.5	30,522	5.98
Rand1M†	1,000,000	127.3	49.0	30,000	0.97
Rand10M†	10,000,000	127.8	49.0	30,000	9.52

*Real-world Datasets; †Synthetic Datasets

minHash indexes are $O(n \log n)$. For the query time, the inner product of at most $T + k$ points are computed with respect to q, where T is a constant. Each inner product computation cost $O(q.nnz)$ time, so the computing cost is $O(q.nnz)$. In the worst case, all points are counted $O(m)$ times, thus the counting cost is $O(nm) = O(n \log n)$. Therefore, the total query time is $O(n \log n + q.nnz)$.

Remark 2. *Although the query time of SOSIA is $O(n \log n + q.nnz)$, which seems to be worse than a brute force method. However, such a cost is the bound of the worst query complexity and the query time is still dominated by the time of computing the real inner products,* i.e., *the computation cost. Fortunately, the computation cost is bounded by $O(q.nnz)$, much better than $O(n)$. Many LSH methods, under Euclidean space, have $O(n)$ or even $O(n \log n)$ query time, such as C2-LSH [18], QA-LSH [22], SRS [49], but still outperform many other LSH methods that have a sublinear query cost, such as LSB-Forest [51]. We will further demonstrate the efficiency of SOSIA in the next section by experiments.*

VI. Experimental Study

We conduct extensive experiments on real world data and synthesized datasets for a comprehensive evaluation and analysis on SOSIA. We implement the SOSIA [2] and competitors in C++ and compiled with g++ using -Ofast optimization. All experiments are conducted on a Ubuntu server with 4 Intel(R) Xeon(R) Gold 6218 CPUs (160 threads) and 1.5 TB RAM.

A. Experimental Settings

Datasets. Six sparse datasets are used in our experiments, as shown in Table II. The former two datasets are obtained by embedding MS Marco Passage Retrieval v1 dataset [4] via the SPLADE model [15], [16], since the SPLADE model has been proven to be the most promising sparse embedding model for document retrieval. SPLADE1M contains 1M points chosen randomly from SPLADE-F. Yelp-1M and Yelp-F are embedded from Yelp dataset[3] via SPLADE model achieved in *pyserini* toolkit[4]. Rand1M and Rand10M are randomly generated based on the datasets SPLADE-F dataset. In Table II, φ_d and φ_q are the average number of non-zero values in the dataset and query set, respectively.

[2]https://github.com/Jacyhust/SOSIA
[3]https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset
[4]https://pypi.org/project/pyserini/

Competitors. To demonstrate the indexing and query performance of our SOSIA framework, we compare it with WAND [6], SINNAMON [7] and HNSW [34]. WAND is the most representative inverted index based method using DAAT (Document-at-a-time) query strategy. SINNAMON [7], proposed by Pinecone, is a good representative for TAAT (Term-at-a-time) strategy-based method. HNSW is a great method for vector search over dense vectors. Since it is highly optimized for dense vectors, we convert the sparse vectors into dense vectors and then adopt HNSW to answer AMIPS.

Parameter Settings. We consider the (c, k)-AMIPS queries with $c = 0.5$ and $k = 50$ for all algorithms as the default settings. Following settings in the paper or source code of our competitors, we adopt the fixed parameters on each algorithm to facilitate using them. The parameter settings of all algorithms are as follows: For SOSIA, the base of SOS, $l = 40$, the number of minHash functions $m = 150$, the maximal number of points to be checked, $T = 10000$ for small datasets (SPLADE1M, Rand1M), and $T = 20000$ for large datasets (SPLADE-Full and Rand10M). For SINNAMON, the sketch dimensionality $m = 0.25\varphi_d$, the number of mappings $h = 2$, and the time budget T is set as twice of the running time of SOSIA. For WAND, we set the factor $F = 2.0$. For HNSW, we set $M = 24$ and $ef = 80$.

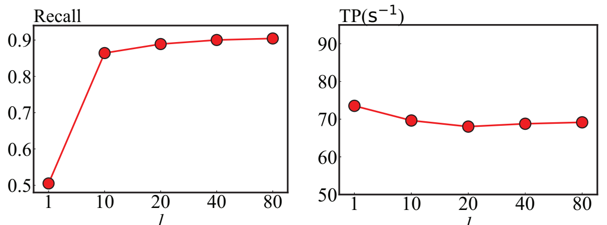
Evaluation Metrics. We evaluate the performance of our SOSIA framework and the competitors with respect to index size, indexing time, recall and throughput. As an approximate requires balance the query efficiency and accuracy. We use throughput to measure the query efficiency, which refers to the number of queries answered at a given time. For an algorithm, given a query q, we denote the result of a (c, k)-AMIPS as $R = \{x_1, x_2, \cdots, x_k\}$. Given a query q, denote $R^* = \{x_1^*, x_2^*, \cdots, x_k^*\}$ as the result of exact k MIPS. The recall are computed as following:

$$Recall = \frac{|R \cap R^*|}{|R^*|} \tag{8}$$

B. Self Evaluation

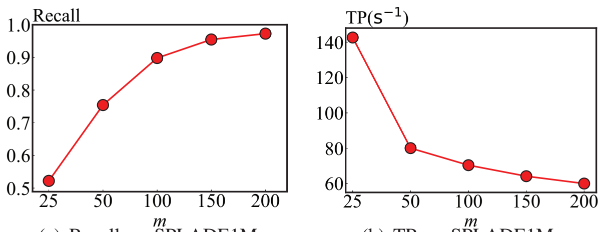
In this subsection, we first demonstrate the effects of the base of SOS, l, and the number of minHash functions, m, on the query performance of our framework SOSIA. Then, we study the effect of the number of threads on the indexing time and throughput.

1) ***Parameter study on*** *l:* To further study the SOSIA framework, we evaluate the impact of l, the base of SOS transformation, on the query performance of SOSIA by setting l in the range of $\{1, 10, 20, 40, 80\}$. As shown in Fig. 2, the throughput of SOSIA remains nearly unchanged as l increases. The reason is that l does not affect the collision probability of two transformed vectors and thus does not affect the query efficiency. As for the recall, it keeps increasing with l, since a larger recall brings us a better estimation for the exact inner product based on Lemma 2 and thus enhance the query accuracy. While the marginal increase of recall is small after $l = 40$, we choose $l = 40$ as the default value.



(a) Recall on SPLADE1M (b) TP on SPLADE1M

Fig. 2: Query Performance of SOSIA when varying l



(a) Recall on SPLADE1M (b) TP on SPLADE1M

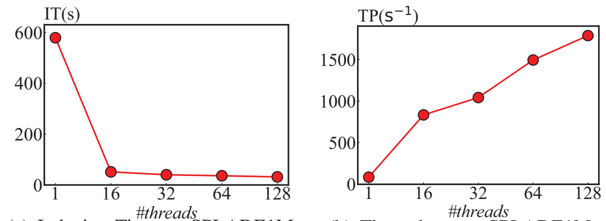
Fig. 3: Query Performance of SOSIA when varying m

2) ***Parameter study on*** m: In this experiment, we evaluate the effect of the m, the number of hash functions, on the query performance of SOSIA by setting m in the range of $\{25, 50, 100, 150, 200\}$ on the dataset SPLADE1M. As shown in Figure 3, the throughput continuously increases when we increase m on both datasets. This is because a larger m requires a longer time for counting the collision numbers. Also, the recall increases as m increases, as more minHash functions yield a more accurate estimation on Jaccard similarity between two sets and thus yield a more accurate estimation on inner product. The marginal increase of the recall is small after $m = 150$, while the throughput would decrease significantly, we choose $m = 150$ as the default value.

3) ***Effect of the number of threads***: In this set of experiments, we study the performance of SOSIA concerning the number of threads by setting the number of threads in the range of $\{1, 16, 32, 64, 128\}$ on the dataset SPLADE1M. As shown in Fig. 4, a larger number of threads yield a smaller indexing time and a larger throughput. The reason is as follows: In the indexing phase, each point can be inserted independently without affecting the other points. In the query processing, different queries can be performed independently without involving inter-query data dependencies. Therefore, a larger number of threads always bring a higher indexing efficiency and query efficiency.

C. Performance Overview

In this experiment, we provide an overview of the average throughput (TP), recall, index size (IS) and indexing time (IT) with default parameter settings on all datasets, as shown in Table III. When comparing the query performance of the different algorithms, we observe that SOSIA consistently achieves



(a) Indexing Time on SPLADE1M (b) Throughput on SPLADE1M

Fig. 4: Efficiency of SOSIA when varying number of threads

TABLE III: Performance Overview

		SOSIA	SINNAMON	WAND	HNSW
SPLADE1M	TP (s^{-1})	**64.18**	30.05	10.91	41.15
	Recall	**0.9548**	0.9526	0.9498	0.8764
	IS (GB)	**1.1**	1.2	0.94	0.52
	IT (s)	**31.1**	3.32	2.34	495.7
SPLADE-F	TP (s^{-1})	**12.13**	30.05	10.91	4.86
	Recall	**0.9548**	0.9526	0.9498	0.802
	IS (GB)	**9.9**	10.3	8.4	4.63
	IT (s)	**255.8**	28.68	19.93	3561.4
Yelp1M	TP (s^{-1})	**28.89**	12.82	6.93	26.39
	Recall	**0.8468**	0.8426	0.8222	0.459
	IS (GB)	**1.1**	1.2	0.94	0.532
	IT (s)	**17.8**	2.32	1.38	690.3
Yelp-F	TP (s^{-1})	**8.16**	3.88	0.89	5.11
	Recall	**0.6754**	0.281	0.937	0.4242
	IS (GB)	**8.2**	8.4	6.8	3.49
	IT (s)	**139.4**	26.49	19.82	6997.4
Rand1M	TP (s^{-1})	**46.79**	1.08	10.91	29.50
	Recall	**0.9548**	0.9526	0.9498	0.89
	IS (GB)	**1.1**	1.2	0.95	0.38
	IT (s)	**54.1**	3.45	2.46	545.7
Rand10M	TP (s^{-1})	**7.28**	10.87	10.91	6.89
	Recall	**0.9548**	0.9526	0.9498	0.9074
	IS (GB)	**11.2**	11.8	9.5	5.8
	IT (s)	**342.9**	32.10	22.46	7096.1

the highest recall on most of datasets, except for SPLADE-Full where it is slightly lower than WAND. HNSW has a low recall compared to the other algorithms, indicating the graph-based methods cannot directly answer the sparse MIPS problem well. Additionally, SOSIA exhibits significantly higher throughputs compared to its competitors. The reason is that SOSIA only considers points with a large collision number to the query, which greatly reduces the cost of exactly computing inner products. When comparing the index size of the different algorithms, we found that all algorithms but HNSW have the close index size and SINNAMON always has the largest one. This is because SINNAMON requires storing an inverted list and n sketches, where the length of the sketch for each point is $m \approx 30$. Compared to WAND that only stores an inverted list, SINNAMON has a larger index size. The index size of SOSIA depends on the number of hash tables m. Each hash table only stores a point once and the total index size is close to the size of dataset itself. HNSW stores $24 - 48$ edges for each vertex, which yields the smallest index size. However, HNSW has a much higher indexing time compared

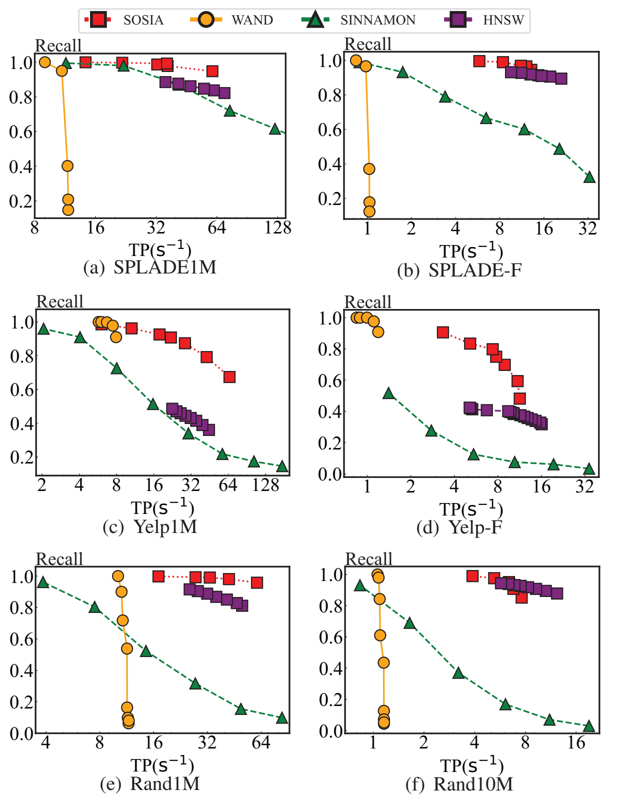


Fig. 5: Recall-Throughput (TP) Curves

to the other three algorithms. Because it requires finding the neighbors for each point, which is time-consuming. WAND and SINNAMON have small indexing times since they only build simple inverted indexes.

D. Evaluation of Query Performance

In these experiments, we analyze the trade-off between recall and throughput (TP) via the Recall-TP curve for all algorithms. The algorithm with the higher throughput for a given target recall is considered to have better query performance. The results on all datasets are shown in Figure 5. From the figures, we make the following observations:

(1) Among all algorithms, SOSIA has the highest throughput to reach the same recall, indicating that it achieves the best trade-off between query quality and efficiency. The reason is that by using SOS transformation and minHash indexes, SOSIA has a better estimator than SINNAMON and WAND. For SINNAMON, the cost of completely scoring a vector x is even higher than directly computing $q^\top x$. Although the scoring phase can be terminated if the time budget T is exhausted, the query accuracy decreases significantly while using the partial scores for ranking and it is difficult to find a proper T. For WAND, the cost of computing an upper bound is also very high. The efficiency of WAND depends on the number of pruned points. Since WAND adopts a global upper bound for each posting list, the estimated upper bound of inner product between a point and the query q is unbounded, which makes the pruning ratio of WAND very low. So WAND has a small throughput. HNSW achieves a high throughput, but its recall is limited. Notably, on the SPLADE-F and Rand10M datasets, HNSW has a throughput comparable to that of SOSIA. This can be attributed to HNSW's query processing, which terminates at a local minimum, effectively preventing excessive query costs. While HNSW excels in throughput, its recall may be a trade-off to consider. As HNSW adopts a heuristic approach for search in the graph index, its query accuracy is unbounded. Also, it is difficult to tune its query accuracy by setting the parameters of the algorithm.

(2) As the throughput decreases, all algorithms achieve higher recall, which aligns with the principle of AMIPS methods, where query accuracy is traded for query efficiency. However, the increasing ratio of recall decreases greatly as the throughput increases, especially when the recall is close to 1. This reveals the difficulty and computationally expensive nature of finding exact MIPS results.

(3) When comparing the recall-TP curves among these four datasets, we find the difference of query performance between SOSIA and the other two algorithms varies greatly. On SPLADE1M and SPLADE-F datasets, SOSIA achieves about 2-4$\times$ speedup of throughput to reach the same recall compared to the second best algorithm. While on Rand1M and Rand10M datasets, the speedup reaches 5-10$\times$, indicating the competitors performs poorly on the random datasets. The reason is that the estimating accuracy and pruning efficiency of SINNAMON and WAND is affected by the data distribution. Moreover, the speedup of SOSIA to its competitors on large dataset such as SPLADE-F and Rand10M is much larger than that on the small dataset, enhancing the adaptability of SOSIA on large-scale datasets.

VII. Conclusion

In this paper, we propose a novel transformation called SOS, which converts sparse vectors into binary vectors while providing an unbiased estimator of their inner product. Building upon SOS, we design SOSIA that utilizes minHash to address the MIPS problem over sparse vectors. SOSIA incorporates a counting-based query strategy, enabling the efficient and adaptive generation of candidates for different query points. This approach allows us to achieve high query accuracy while greatly reducing computational costs. Theoretical analysis demonstrates that SOSIA guarantees a correct c-AMIPS result with constant probability by verifying only a limited number of points. Extensive experiments reveal that SOSIA outperforms its strongest competitor by achieving a query time speedup of 2-10$\times$ while maintaining the same recall.

Acknowledgment

The research work described in this paper was supported by China Unicom (grant# CUNICOMHK23EG01-A). It was partially conducted in JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust.

References

[1] P. Agrawal, A. Arasu, and R. Kaushik. On indexing error-tolerant set containment. In *SIGMOD Conference*, pages 927–938. ACM, 2010.

[2] N. Asadi and J. Lin. Fast candidate generation for real-time tweet search with bloom filter chains. *ACM Trans. Inf. Syst.*, 31(3):13, 2013.

[3] Y. Bachrach, Y. Finkelstein, R. Gilad-Bachrach, L. Katzir, N. Koenigstein, N. Nice, and U. Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *RecSys*, pages 257–264, 2014.

[4] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, et al. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

[5] A. Z. Broder. On the resemblance and containment of documents. In *SEQUENCES*, pages 21–29. IEEE, 1997.

[6] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Y. Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM*, pages 426–434. ACM, 2003.

[7] S. Bruch, F. M. Nardini, A. Ingber, and E. Liberty. An approximate algorithm for maximum inner product search over streaming sparse vectors. *CoRR*, abs/2301.10622, 2023.

[8] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke. The rise of deep learning in drug discovery. *Drug discovery today*, 23(6):1241–1250, 2018.

[9] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[10] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[11] X. Dai, X. Yan, K. K. W. Ng, J. Liu, and J. Cheng. Norm-explicit quantization: Improving vector quantization for maximum inner product search. In *AAAI*, pages 51–58, 2020.

[12] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, pages 253–262, 2004.

[13] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. Optimizing top-k document retrieval strategies for block-max indexes. In *WSDM*, pages 113–122. ACM, 2013.

[14] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *SIGIR*, pages 993–1002. ACM, 2011.

[15] T. Formal, C. Lassance, B. Piwowarski, and S. Clinchant. SPLADE v2: Sparse lexical and expansion model for information retrieval. *CoRR*, abs/2109.10086, 2021.

[16] T. Formal, B. Piwowarski, and S. Clinchant. SPLADE: sparse lexical and expansion model for first stage ranking. In *SIGIR*, pages 2288–2292. ACM, 2021.

[17] M. Fraccaro, U. Paquet, and O. Winther. Indexable probabilistic matrix factorization for maximum inner product search. In *AAAI*, pages 1554–1560, 2016.

[18] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD Conference*, pages 541–552. ACM, 2012.

[19] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[20] B. Goodwin, M. Hopcroft, D. L. anwod Alex Clemmer, M. Curmei, S. Elnikety, and Y. He. Bitfunnel: Revisiting signatures for search. In *SIGIR*, pages 605–614. ACM, 2017.

[21] J. He, S. Kumar, and S. Chang. On the difficulty of nearest neighbor search. In *ICML*, 2012.

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

[23] Q. Huang, G. Ma, J. Feng, Q. Fang, and A. K. H. Tung. Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search. In *KDD*, pages 1561–1570, 2018.

[24] P. Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[25] J. Ji, J. Li, S. Yan, Q. Tian, and B. Zhang. Min-max hash for jaccard similarity. In *ICDM*, pages 301–309. IEEE Computer Society, 2013.

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

[27] N. Koenigstein, P. Ram, and Y. Shavitt. Efficient retrieval of recommendations in a matrix factorization framework. In *CIKM*, pages 535–544, 2012.

[28] S. Kosub. A note on the triangle inequality for the jaccard distance. *Pattern Recognition Letters*, 120:36–38, 2019.

[29] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[30] J. Lemiesz. Efficient framework for operating on data sketches. *Proc. VLDB Endow.*, 16(8):1967–1978, 2023.

[31] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement. *IEEE Trans. Knowl. Data Eng.*, 32(8):1475–1488, 2020.

[32] J. Lin and A. Trotman. Anytime ranking for impact-ordered indexes. In *ICTIR*, pages 301–304. ACM, 2015.

[33] J. Liu, X. Yan, X. Dai, Z. Li, J. Cheng, and M. Yang. Understanding and improving proximity graph based maximum inner product search. In *AAAI*, pages 139–146, 2020.

[34] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, 2020.

[35] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. Faster blockmax WAND with variable-sized blocks. In *SIGIR*, pages 625–634. ACM, 2017.

[36] A. Mallia and E. Porciani. Faster blockmax WAND with longer skipping. In *ECIR (1)*, volume 11437 of *Lecture Notes in Computer Science*, pages 771–778. Springer, 2019.

[37] S. Morozov and A. Babenko. Non-metric similarity graphs for maximum inner product search. In *NeurIPS*, pages 4726–4735, 2018.

[38] B. Neyshabur and N. Srebro. On symmetric and asymmetric lshs for inner product search. In *ICML*, volume 37, pages 1926–1934, 2015.

[39] N. Pham. Simple yet efficient algorithms for maximum inner product search via extreme order statistics. In *KDD*, pages 1339–1347. ACM, 2021.

[40] R. Pratap, D. Bera, and K. Revanuru. Efficient sketching algorithm for sparse binary data. In J. Wang, K. Shim, and X. Wu, editors, *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, pages 508–517. IEEE, 2019.

[41] P. Ram and A. G. Gray. Maximum inner-product search using cone trees. In *KDD*, pages 931–939, 2012.

[42] T. Report. https://github.com/jacyhust/sosia/blob/main/report. 2023.

[43] A. S. R. Santos, A. Bessa, C. Musco, and J. Freire. A sketch-based index for correlated dataset search. In *ICDE*, pages 2928–2941. IEEE, 2022.

[44] F. Shen, W. Liu, S. Zhang, Y. Yang, and H. T. Shen. Learning binary codes for maximum inner product search. In *ICCV*, pages 4148–4156, 2015.

[45] A. Shrivastava and P. Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *NIPS*, pages 2321–2329, 2014.

[46] A. Shrivastava and P. Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *WWW*, pages 981–991. ACM, 2015.

[47] A. Shrivastava and P. Li. Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS). In *UAI*, pages 812–821, 2015.

[48] Y. Song, Y. Gu, R. Zhang, and G. Yu. Promips: Efficient high-dimensional c-approximate maximum inner product search with a lightweight index. In *ICDE*, pages 1619–1630. IEEE, 2021.

[49] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proc. VLDB Endow.*, 8(1):1–12, 2014.

[50] S. Tan, Z. Zhou, Z. Xu, and P. Li. On efficient retrieval of top similarity vectors. In *EMNLP/IJCNLP (1)*, pages 5235–5245, 2019.

[51] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009.

[52] Y. Tian, X. Zhao, and X. Zhou. DB-LSH: locality-sensitive hashing with query-based dynamic bucketing. In *ICDE*, pages 2250–2262, 2022.

[53] X. Yan, J. Li, X. Dai, H. Chen, and J. Cheng. Norm-ranging LSH for maximum inner product search. In *NeurIPS*, pages 2956–2965, 2018.

[54] Y. Zhang and Z. G. Ives. Finding related tables in data lakes for interactive data science. In *SIGMOD Conference*, pages 1951–1966. ACM, 2020.

[55] X. Zhao, B. Zheng, X. Yi, X. Luan, C. Xie, X. Zhou, and C. S. Jensen. FARGO: fast maximum inner product search via global multi-probing. *Proc. VLDB Endow.*, 16(5):1100–1112, 2023.

[56] B. Zheng, X. Zhao, L. Weng, N. Q. V. Hung, H. Liu, and C. S. Jensen. PM-LSH: A fast and accurate LSH framework for high-dimensional approximate NN search. *PVLDB*, 13(5):643–655, 2020.

[57] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller. JOSIE: overlap set similarity search for finding joinable tables in data lakes. In *SIGMOD Conference*, pages 847–864. ACM, 2019.

[58] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller. LSH ensemble: Internet-scale domain search. *Proc. VLDB Endow.*, 9(12):1185–1196, 2016.