Reconsidering Tree based Methods for k-Maximum Inner-Product Search: The LRUS-CoverTree

1st Hengzhao Ma, 2nd Jianzhong Li, 3rd Yong Zhang
Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences
Shenzhen, China
{hz.ma,lijzh,zhangyong}@siat.ac.cn

***Abstract*—Existing literature on k-Maximum Inner-Product Search has made it a common belief that tree based methods are less effective in terms of index construction time and query performance compared to locality sensitive hashing based, similarity graph based and quantization based methods. However, in this paper we partially roll over the existing assessments about tree based k-Maximum Inner-Product Search methods by our newly proposed tree structure named LRUS-CoverTree. The experimental results show that the new k-Maximum Inner-Product Search algorithm based on LRUS-CoverTree outperforms the state-of-the-art locality sensitive hashing based methods, and achieves comparable performance with similarity graph based and quantization based methods in terms of query time and accuracy. What's more important, the desirable query performance is attained with significantly lower index construction time compared to all the other methods. Besides the experimental evaluations, substantial theoretical results about the LRUS-CoverTree and the new k-Maximum Inner-Product Search algorithm are provided, including construction time and search time complexity, size and height of the tree, and so on. Furthermore, several new effective upper bounds on the inner-product value are provided to support the efficient branch-and-bound algorithm on LRUS-CoverTree. In summary, our novel tree structure and new algorithm significantly improve upon existing tree based methods, and it is hoped that this contribution can lead to a reconsideration of tree based k-Maximum Inner-Product Search methods.**

***Index Terms*—k-Maximum Inner Produce Search, Tree index structure, Cover Tree.**

I. Introduction

The k-Maximum Inner-Product Search (k-MIPS) is a well known similarity search problem with a wide range of applications. Given an input point set $P \subset \mathbb{R}^d$ and a query point $q \in \mathbb{R}^d$, k-MIPS aims to find the k points in P with maximum inner-product value. k-MIPS has witnessed a lot of applications in many research areas, such as image retrieval [1], natural language processing [2], [3], recommendation system [4], [5], and vector database [6], to mention just a few. With the advent of Large Scale Artificial Intelligence, maximum inner-product search has also found its importance as one of the supporting technologies of modern AI [7].

As a result of many years of research efforts, the existing methods for k-MIPS can be categorized into four classes including tree based, locality sensitive hashing based, similarity graph based, and quantization based methods. This paper focuses on tree based methods, and the related works of the other three classes will be briefly surveyed in Section II.

Tree based methods organize the input data into a tree structure to support the maximum inner-product search. Examples of such methods include Random Partition Tree [8], [9], Cone Tree [10], and Cover Tree [11], [12]. It is commonly believed that tree based methods are only suitable for low-dimensional data, and the tree construction time and search time grow exponentially with dimension [13], [14]. What's more, there is another obstacle for tree based k-MIPS methods, that is, the inner-product is not a distance metric and does not satisfy the triangular inequality. As a result, the usually used branch-and-bound techniques on space-partitioning trees are not readily applicable to k-MIPS. Consequently, the performance of current tree based methods suffers from high index construction time and high search time, which makes it inferior to the other kinds of methods for k-MIPS.

Despite the common belief that tree based methods exhibit poor performance for the k-MIPS problem, we believe, to the contrary, that it is only because the appropriate tree structure for k-MIPS has not been discovered. In this paper, a novel tree structure for k-MIPS named LRUS-CoverTree is proposed, which significantly outperforms the existing tree based methods in tree construction time and query performance. Compared with other kinds of k-MIPS methods, LRUS-CoverTree outperforms the state-of-the-art LSH based method in both index construction time and query performance. For similarity graph and quantization based methods, the query performance of LRUS-CoverTree is no better but comparable, but the index construction time of LRUS-CoverTree is more than 1 order of magnitude lower. In other words, LRUS-CoverTree achieves comparable or even better query performance to the state-of-the-art with far lower preprocessing costs.

The concrete contributions of this paper are listed below.

1. A new tree structure for k-MIPS is proposed which is named the Long Root Unit Sphere Cover Tree (LRUS-CoverTree). The important properties of LRUS-CoverTree including children size, maximum height, etc., as well as the construction time complexity and space complexity are formally analyzed.

2. A new exact k-MIPS algorithm based on the LRUS-CoverTree is proposed. The central contribution of the algorithm is several new upper bounds on the inner-product value based on the LRUS-CoverTree data structure, which support

This work was supported by the National Natural Science Foundation of China (Grant number 61832003, 62273322, 61972110), and National Key Research and Development Program of China (Grant number 2021YFF1200100, 2021YFF1200104).

pruning nodes efficiently during the k-MIPS search procedure.

3. A new approximate k-MIPS algorithm is proposed, which can return approximate results for a given approximation bound ϵ in $O\left((\frac{4}{(1-\epsilon)\epsilon})^d\right)$ time. Note that this complexity is independent with the input size n.

4. Substantial experimental results are presented to evaluate the performance of our new k-MIPS algorithms. By comparing with existing methods, it is shown that our new k-MIPS algorithms based on LRUS-CoverTree outperform the state-of-the-art tree based and LSH based method, and achieve comparable query performance with similarity graph based and quantization based methods. The plausible query performance is achieved with significantly lower index construction cost. It is shown that the construction time of LRUS-CoverTree is more than 1 order of magnitude lower than all the compared methods on all datasets tested. The experimental performance of our LURS-CoverTree based algorithm is a great surprise against the existing beliefs about tree based k-MIPS methods.

The rest of this paper is organized as follows. Section II overviews the related works. Section III introduces the preliminaries, including the theoretical basement of our new upper bound on inner-product value. Section IV proposes the LRUS-CoverTree along with the construction algorithm, and analyzes its important properties such as children bound, height bound, and many others. Section V describes the new exact and approximate k-MIPS algorithms, and proves their correctness and complexity respectively. Section VI presents the experimental results. Section VII concludes this paper.

II. Related Works

Locality sensitive hashing (LSH) [15]–[17] is a famous method for nearest neighbor search but can also be used for k-MIPS [18]–[22], which requires applying asymmetric transformation to data and query. There are several existing kinds of transformation that convert k-MIPS to nearest neighbor search, such as L2-Transformation [23], Correlation-Transformation [24], and XBOX Transformation [25]. Based on these transformations and the existing locality sensitive hashing functions, the LSH based methods achieve competitive performance for k-MIPS, e.g., the FARGO method [26] and ProMIPS method [27]. The shortcoming of LSH based methods is that applying the asymmetric transformation to data and query incurs additional overhead and may reduce accuracy.

Similarity graph is a kind of graph where the nodes represent the input data, and an edge between two nodes exists only when the two nodes satisfy a specific similarity criterion. There are several kinds of similarity graphs including Relative Neighborhood Graph [28], Navigable Small World Graph (NSW) [29], [30], Delaunay Graph [31], and so on. Among these various similarity graphs, the NSW graph leads to the state-of-the-art method for k-MIPS, e.g., the ip-nsw [32], ip-nsw+ [33] and norm-adjusted proximity graph (NAPG) [34] methods. The disadvantage of similarity graph based methods lies in the expensive cost for constructing the graph.

Quantization [35], [36] is a kind of method that uses a small codebook to encode a large dataset. The encoded data is usually much smaller than the original data and thus permits fast inner-product computation, although sacrificing the accuracy. When the quantization method is well designed for k-MIPS, the resulted algorithm can achieve good balance of search speed and accuracy. The state-of-the-art quantization based k-MIPS methods include ScaNN [36] and Anisotropic Additive Quantization [37]. The quantization method is close to machine learning techniques where the construction of codebook requires a loss function minimization process. As a result, there is no theoretical guarantee on the accuracy for the results returned by the quantization based search procedure.

Another problem of the above three methods is that they can only find approximate results for k-MIPS, while tree based methods can apply to both exact and approximate k-MIPS.

There exist some methods outside the above three classes. For example, there are several works that solve the k-MIPS problem by a multi-arm bandit process [38], [39]. Some other works consider variations of MIPS problem. For example, Yu et. al. [40] considers the budgeted MIPS problem and tackle it by a greedy approach. Ballard et. al. [41] use a diamond sampling method to solve the maximum all-pairs inner-product search problem. The methods mentioned here may not be comprehensive due to the vast literature of the MIPS problem.

TABLE I: List of symbols

notation	explanation
$\|p\|$	the (L_2) norm
$\langle x, y\rangle$	the inner-product
$D(x, y)$	the Euclidean distance
d	the number of dimension
$angle(x, y)$	the angle formed by vectors x, y
$\cos(x, y)$	shorthand of $\cos(angle(x, y))$
$\sin(x, y)$	shorthand of $\sin(angle(x, y))$
q	the query point
P	the set of points to query against
$MIP(k, q, P)$	the point set containing exact k-MIPS results
$MIP_\epsilon(k, q, P)$	the point set containing approximate k-MIPS results
$\mathcal{N}_\cdot$	LRUS-CoverTree node and information. See Table II

III. Preliminaries

Metric space and inner-product. Throughout this paper we use P to denote the input point set, which is extracted form the metric space $\mathbb{R}^d$. Each point $p \in P$ is a d-dimensional point in the form $p = (p_{(1)}, p_{(2)}, \cdots, p_{(d)})$, where $p_{(i)}$ is the i-th coordinate. Since each point in $\mathbb{R}^d$ can be regarded as the end point of the vector started from the origin point $(0, 0, \cdots, 0)$, points and vectors will be used interchangeably from now on.

Given two points x, y, the inner-product of them, denoted as $\langle x, y\rangle$, is $\sum_{i=1}^{d}(x_{(i)} \times y_{(i)})$. The norm of a vector p, denoted as $\|p\|$, is $\sqrt{\sum_{i=1}^{d} p_{(i)}^2}$. A vector u is called a unit vector if $\|u\| = 1$. It is easy to see $\frac{p}{\|p\|}$ is a unit vector for any point p with non-zero norm. The Euclidean distance between two points x, y is $D(x, y) = \sqrt{\sum_{i=1}^{d}(x_{(i)} - y_{(i)})^2}$.

To avoid ambiguity, the symbol '$\times$' is used to denote the multiplication of two real numbers. The capital D is used to denote the Euclidean distance and d is used to denote the number of dimension.

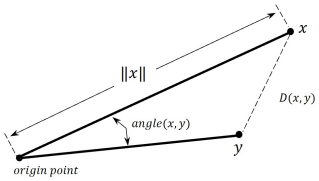


Fig. 1: Demonstration of vectors, angle and norm

Trigonometric functions. Given two vectors x, y, let $angle(x, y)$ denote the angle formed by the two vectors. Let $\cos(x, y)$ be a shorthand of $\cos(angle(x, y))$, $\sin(x, y)$ be a shorthand of $\sin(angle(x, y))$, which are the cosine and sine value of $angle(x, y)$, respectively.

It is well known that $\langle x, y\rangle = \|x\|\|y\|\cos(x, y)$. Besides, by the well known cosine formula, it holds that $\cos(x, y) = \frac{\|x\|^2+\|y\|^2-D(x,y)^2}{2\|x\|\|y\|}$. If both x, y are unit vectors, it holds that $\cos(x, y) = 1 - \frac{D(x,y)^2}{2}$.

The cosine value can be regarded as a similarity measure between two vectors, but unfortunately it does not satisfy the triangle inequality. However, the angular distance does. Formally, for three vectors x, y, z, it holds that $angle(x, z) \leq angle(x, y) + angle(y, z)$. We have the following lemma as a bound on the cosine value of angular difference.

Lemma 1. *Given three vectors $x, y, z \in \mathbb{R}^d$, it holds that* $\cos(x, y) \leq \cos(x, z)\cos(y, z) + \sin(x, z)\sin(y, z)$.

Proof. There are several important facts to be pointed out. First, $angle(x, y)$ takes value in $[0, \pi]$ for any $x, y \in \mathbb{R}^d$. Second, the cosine function is monotonically decreasing on $[0, \pi]$. Third, the cosine function is an even function which satisfies $\cos(x) = \cos(-x)$ for any real number x.

Utilizing the above observations and starting from the triangular inequality on angular distance, we have

$$\begin{aligned} angle(x, y) &\geq |angle(x, z) - angle(y, z)| \\ \Rightarrow \cos(x, y) &\leq \cos(|angle(x, z) - angle(y, z)|) \\ &= \cos(angle(x, z) - angle(y, z)) \\ &= \cos(x, z)\cos(y, z) + \sin(x, z)\sin(y, z) \end{aligned}$$

The first "$\Rightarrow$" is based on the monotonically decreasing property of cosine function on $[0, \pi]$. The next '=' is according to the property of even function. Finally, the last '=' is the well known cosine difference formula. □

The bound is also effective in the following case, where a separating bound is available on the distances between unit vectors x, y and z. The following corollary will be intensively used in the proofs in this paper.

Corollary 1. *Given three unit vectors x, y and z, let $D(x, z) \leq \eta \leq D(y, z)$, and denote $\cos\theta = 1 - \eta^2/2$, $\sin\theta = \sqrt{1 - \cos^2\theta}$, where $\eta > 0$ is a positive real value. Then it holds that* $\cos(x, y) \leq \cos(y, z)\cos\theta + \sin(y, z)\sin\theta$.

Proof. By the given condition, it holds that $angle(z, x) \leq \theta \leq angle(z, y)$. Then $\cos(x, y)$ reaches the maximum only when $angle(z, x) = \theta$. The corollary follows by applying Lemma 1 on x, y, z and letting x satisfy $angle(z, x) = \theta$. □

k-Maximum Inner-Product Search (k-MIPS). The definition of exact k-MIPS is given below. When the query point q is given and fixed, we use the *inner-product value* of point p to refer to $\langle q, p\rangle$.

Definition 1 (exact k-MIPS)**.** *Given a point set $P \subset \mathbb{R}^d$, a query point $q \in \mathbb{R}^d$, and an integer k, find the set of k points in P with maximum inner-product value, denoted as $MIP(k, q, P)$. Formally, find $MIP(k, q, P) \subset P$ such that (1) $|MIP(k, q, P)| = k$, and (2) $\langle q, p\rangle \geq \langle q, x\rangle$ for $\forall p \in MIP(k, q, P)$ and $\forall x \in P \setminus MIP(k, q, P)$.*

From now on $MIP(k, q, P)$ will be used to denote the k-MIPS result where q is query point and P is the point set to query against. Definition 1 assumes that no two points in P are of the same inner-product value. This assumption can be removed by define k-MIPS using multi-set on the inner-product values. For simplicity, the multi-set definition is omitted and all the discussions in this paper are based on the simper Definition 1.

It is easy to see that the norm of query point does not affect the results of k-MIPS. Formally, let u be a unit vector, and c_1, c_2 be two arbitrary positive real values, then it holds that $MIP(k, c_1u, P) = MIP(k, c_2u, P)$. Thus, it is asserted that the query point q is a unit vector from now on. In such way, $\langle q, p\rangle = \|p\|\cos(q, p)$.

Approximate k-MIPS. The definition of approximate k-MIPS is similar with widely-used multiplicative approximation in combinatorial optimization. However, the inner-product value can be negative, while the multiplicative approximation only applies to positive values. Thus, the following Definition 2 imposes an extra assumption that no negative inner-product value exists. Note that the assumption seems strong but can be ensured in a lot of real applications such as image retrieval and recommendation system.

Definition 2 (approximate k-MIPS)**.** *Given point set $P \subset \mathbb{R}^d$ and query point $q \in \mathbb{R}^d$, assume $\langle q, p\rangle \geq 0$ for $\forall p \in P$. Then, given an integer k and approximation factor $\epsilon \leq 1$, find a set $MIP_\epsilon(k, q, P) \subset P$ which satisfies (1) $|MIP_\epsilon(k, q, P)| = k$, and (2) $\min_{p\in MIP_\epsilon(k,q,P)} \langle p, q\rangle \geq \min_{p\in MIP(k,q,P)} \langle p, q\rangle \times \epsilon$.*

From now on $MIP_\epsilon(k, q, P)$ will be used to denote the approximate k-MIPS result, where q is query point, ϵ is the approximate factor, and P is the point set to query against.

Note that the above definition only asks to bound the approximation ratio for the smallest inner-product value in approximate result. A stronger version of approximation requires that the approximation quality is bounded for each point in the approximate result. Formally, $\langle q, p'_i\rangle \geq \epsilon\langle q, p^*_i\rangle$ for $1 \leq i \leq k$, where $\langle q, p'_i\rangle$ and $\langle q, p^*_i\rangle$ is the i-th largest inner-product value in the approximate and exact results sets, respectively. Based on the experiments in Section VI, Definition 2 is enough to find satisfactory approximate results. Thus, the stronger approximation is not considered here and left as future work.

IV. The LRUS-CoverTree

A. The idea behind LRUS-CoverTree

Let us first introduce the idea of designing LRUS-CoverTree by examining the inner-product formula $\langle q, p\rangle = \|p\|\|q\|\cos(q,p)$. When the query point q is fixed, the inner-product value can be regarded as the multiplication of norm and cosine. We introduce the following two observations.

First, points with large norm tend to have larger inner-product value and tend to be in the result of k-MIPS. Thus, to design an efficient tree structure for k-MIPS, points with larger norm should be placed in the upper levels of the tree so that they can be visited earlier than those with smaller norm.

Second, a tree structure must have the ability to bound the cosine value to support an efficient k-MIPS algorithm, since the cosine value contributes as a part of the inner-product formula. By observing the cosine formula $\cos(q,p) = \frac{\|q\|^2+\|p\|^2-D(q,p)^2}{2\|q\|\|p\|}$, for unit vectors q, p, the formula simplifies to $\cos(q,p) = 1 - \frac{D(q,p)^2}{2}$. Thus, we come up the idea of using Euclidean distance between unit vectors to organize the tree structure for k-MIPS. This allows designing and utilizing effective bounds on the cosine value, and in turn allows the development of efficient k-MIPS algorithms.

We must note that, despite the simplicity of the aforementioned ideas, they unfortunately have not been effectively utilized to design efficient tree structures for k-MIPS. Our proposed LRUS-CoverTree, designed on the basis of the above two ideas, immediately achieves great improvement against existing tree based k-MIPS methods.

B. The structure of LRUS-CoverTree

The LRUS-CoverTree, which stands for *Long Root Unit Sphere* Cover Tree, is a leveled tree where each node is associated with an integer scale. The scale decreases while the tree is descended. Let $\mathcal{N}$ denote a LRUS-CoverTree node, and $\mathcal{N}$.scale be its scale. Also, let $\mathcal{N}$.point denote the data point associated with node $\mathcal{N}$, and let $\mathcal{N}$.upoint$= \frac{\mathcal{N}.\text{point}}{\|\mathcal{N}.\text{point}\|}$ which is a unit vector. By such denotations, each LRUS-CoverTree node is regarded as a compound data structure storing various information (See Table II). The definition of LRUS-CoverTree is given in Definition 3.

Definition 3. *Given a set of data points $P \subset \mathbb{R}^d$ and a minimum scale $\delta \leq 0$, the LRUS-CoverTree is a tree structure satisfying the following invariants.*

*1. (**Unique Representation**) Each node represents exactly one data point, and there are no two nodes representing the same data point.*

*2. (**Long Root**) For any node $\mathcal{N}$, any descendant $\mathcal{N}'$ of $\mathcal{N}$ satisfies $\|\mathcal{N}'.point\| \leq \|\mathcal{N}.point\|$.*

*3. (**Unit Sphere Descendants Covering**) For any node $\mathcal{N}$, any descendant $\mathcal{N}'$ of $\mathcal{N}$ satisfies $D(\mathcal{N}'.upoint, \mathcal{N}.upoint) \leq 2^{(\mathcal{N}.scale)}$.*

*4. (**Unit Sphere Sibling Packing**) For any node $\mathcal{N}$, any two children $\mathcal{N}_{c1}, \mathcal{N}_{c2}$ satisfies $D(\mathcal{N}_{c1}.upoint, \mathcal{N}_{c2}.upoint) > 2^{(\mathcal{N}.scale-1)}$.*

TABLE II: Information stored in LRUS-CoverTree nodes

notation	explanation
$\mathcal{N}$	an LRUS Cover Tree node
self information	
$\mathcal{N}$.scale	the scale of $\mathcal{N}$
$\mathcal{N}$.point	the data point associated to node $\mathcal{N}$
$\mathcal{N}$.norm	the norm, defined as $\|\mathcal{N}.\text{point}\|$
$\mathcal{N}$.upoint	the unit point, defined as $\mathcal{N}.\text{point} / \|\mathcal{N}.\text{point}\|$
parent information	
$\mathcal{N}$.parent	the parent node of $\mathcal{N}$
$\mathcal{N}$.pdist	parent distance, defined as $D(\mathcal{N}.\text{upoint}, \mathcal{N}.\text{parent.upoint})$
$\mathcal{N}$.pcos	parent cosine, defined as $cos(\mathcal{N}.\text{upoint}, \mathcal{N}.\text{parent.upoint})$
$\mathcal{N}$.psine	parent sine, defined as $sin(\mathcal{N}.\text{upoint}, \mathcal{N}.\text{parent.upoint})$
descendant information	
$\mathcal{N}$.descs	the set of descendant points of $\mathcal{N}$ (before $Expand$ is executed on it)
$\mathcal{N}$.cds	the set of close descendant points of $\mathcal{N}$ (See the 5th term in Definition 3)
$\mathcal{N}$.furdesc	the furthest descendant, defined as $\arg\max_{p\in\mathcal{N}.\text{descs}} D(\frac{p}{\|p\|}, \mathcal{N}.\text{upoint})$
$\mathcal{N}$.fddist	the furthest descendant distance, defined as $\max_{p\in\mathcal{N}.\text{descs}} D(\frac{p}{\|p\|}, \mathcal{N}.\text{upoint})$
$\mathcal{N}$.fdcos	the furthest descendant cosine, defined as $\cos(p, \mathcal{N}.\text{furdesc})$
$\mathcal{N}$.fdsine	the furthest descendant sine, defined as $\sin(p, \mathcal{N}.\text{furdesc})$
$\mathcal{N}$.ldn	the longest descendant norm, defined as $\max_{p\in\mathcal{N}.\text{descs}} \|p\|$
parent-descendant information	
$\mathcal{N}$.pfddist	the parent-furthest descendant distance, $D(\mathcal{N}.\text{parent.upoint}, \frac{\mathcal{N}.\text{furdesc}}{\|\mathcal{N}.\text{furdesc}\|})$
$\mathcal{N}$.pfdcos	the cosine value between parent and furthest descendant
$\mathcal{N}$.pfdsine	the sine between parent and furthest descendant

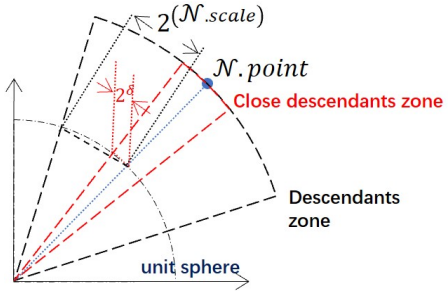


Fig. 2: The covering region for a node in LRUS-CoverTree

*5. (**Close Descendants**) For any node $\mathcal{N}$, if a point $x \in P$ satisfies $D(\mathcal{N}.upoint, \frac{x}{\|x\|}) \leq 2^{\delta}$ then x is included in the close descendants set storing in $\mathcal{N}$, and does not appear anywhere else in the tree.*

Figure 2 shows the covering region of a LRUS-CoverTree node $\mathcal{N}$. By the Long Root invariant, all descendants of $\mathcal{N}$ must have a smaller norm than $\mathcal{N}$.point. By the Unit Sphere Descendant Covering invariant, the maximum distance between the unit vectors determines a maximum angle between $\mathcal{N}$.point and each descendant point. Thus, the covering region is a *sector*. The thing is similar for the close descendants region. In Figure 2, all the close descendants of $\mathcal{N}$ fall in the red dotted sector, and all the other descendants are located in the black dotted sector excepting the red dotted sector.

LRUS-CoverTree can be regarded as a new variant of the cover tree introduced by Beygelzimer et al. [42]. However, the five invariants in the definition of LRUS-CoverTree make it a dedicated and efficient structure for k-MIPS. The *Unique Representation* invariant ensures that each point is associated with only one tree node, which reduces space requirement and improves search efficiency. The *Long Root* invariant ensures that points with larger norm will be visited earlier than those with smaller norm during the top-down search, which reflects the idea that points with larger norm tends to be the result of

k-MIPS. The *Unit Sphere Covering* and *Packing* invariants make it possible to use Euclidean distance between unit vectors to place bounds on the cosine value. Combined with the *Long Root* invariant, the LRUS-CoverTree provides the ability to up-bound the inner-product value of descendant points using parent node information. Note that the bounds provided by LRUS-CoverTree is new and more effective than those provided by existing tree methods. Finally, the *Close Descendant* invariant sets up a cut-off threshold for the tree height and significantly reduces the tree construction time.

C. The construction of LRUS-CoverTree

The construction algorithm of the LRUS-CoverTree is given in Algorithm 1. The core procedure is the $Expand$ operation, which constructs the children nodes based on a set of descendants. We need the following denotations to describe the algorithm. Let $\mathcal{N}$.descs be the set of descendant points of node $\mathcal{N}$, and $\mathcal{N}$.cds be the set of close descendants for node $\mathcal{N}$.

The algorithm first constructs the $root$ node and $root$.point is set to be the point in P with the largest norm. Then the rest of the points in P are traversed, where points p s.t. $D(root.\text{point}, \frac{p}{\|p\|}) \leq 2^{\delta}$ are put in $root$.cds, and the other points are put in $root$.descs. In such way, $root$.descs$= P \setminus \{root.\text{point}\} \setminus root.\text{cds}$. Then the $Exapnd$ function is invoked on $root$ to construct the LRUS-CoverTree recursively.

The $Expand$ operation on a node $\mathcal{N}$ will iteratively pick the point p^* with largest norm in $\mathcal{N}$.descs and construct the child node $\mathcal{N}_c$ to represent p^*. The scale of $\mathcal{N}_c$ is firstly set to $\mathcal{N}$.scale-1. Then the points in $\mathcal{N}$.descs are examined one by one, such that the points satisfying the Unit Sphere Descendant Covering invariant with $\mathcal{N}_c$ are transferred from $\mathcal{N}$.descs to $\mathcal{N}_c$.descs, and the points satisfying Close Descendants invariant are transferred to $\mathcal{N}_c$.cds. After the descendants of $\mathcal{N}_c$ are determined, the scale of it is set to $\lceil \log_2 \mathcal{N}_c.\text{fddist} \rceil$, where $\mathcal{N}_c$.fddist$= \max_{p \in \mathcal{N}_c.\text{descs}} D(\frac{p}{\|p\|}, \mathcal{N}_c.\text{upoint})$ and it is called the furthest descendant distance. This operation is to make the scale of $\mathcal{N}_c$ minimized. The above procedure proceeds until $\mathcal{N}$.descs becomes empty, and that is when the children nodes of $\mathcal{N}$ are determined. Finally the $Expand$ operation is recursively executed on each child node of $\mathcal{N}$.

The following theorem shows the correctness of the $Expand$ construction algorithm. See [44] for omitted proofs.

Theorem 1. *Algorithm 1 correctly constructs the LRUS-CoverTree in accordance with Definition 3.*

Proof. We prove all the invariants in Definition 3 are ensured.
(1) Unique Representation: it is easy to see that $\mathcal{N}$.point is different with any of its descendant points for any node $\mathcal{N}$, and the points in separate subtrees are different.
(2) Long Root: it is ensured by Line 2 and 11 of Algorithm 1, i.e., each time the point with largest norm among the descendants is chosen as the root of a subtree.
(3) Unit Sphere Descendant Covering: for the root node, $root$.scale is initially set to 1, and it is true that $D(x, y) \leq 2$ for any two unit vectors x, y. For other nodes, the condition given in Line 16 ensures the Descendant Covering invariant. Also, updating the scale at Line 6 and 19 does not violate the Descendant Covering invariant.
(4) Unit Sphere Sibling Packing: it is easy to see that any point violating the Sibling Packing invariant must satisfy the Descendant Covering invariant. Thus, if two sibling nodes $\mathcal{N}_1$ and $\mathcal{N}_2$ violates the Sibling Packing invariant, then $\mathcal{N}_2$.point must be included in $\mathcal{N}_1$.descs, contradicting with that they are sibling nodes. Then it is clear that the violation of Sibling Packing invariant can not exist.
(5) Close Descendants: the $AddDescdant$ function correctly separates the close descendants and the other descendants. Besides, since the $Expand$ operation only works with the descendant points and the close descendants will not be touched, it is true that any close descendants will not appear anywhere else in the tree. □

Algorithm 1: Construction of the LRUS-CoverTree

Input: dataset $P = \{p_1, \cdots, p_n\}$, minimum scale δ.
Output: a LRUS-CoverTree.

1 Let $U = \{u_1, u_2, \cdots, u_n\}$ be the normalized dataset, i.e., $u_i = \frac{p_i}{\|p_i\|}$ for $1 \leq i \leq n$;
2 Let s_1 be the point in P with the largest norm;
3 Construct the root node with $root$.point$= s_1$, and set $root$.scale$= 1$;
4 **foreach** $i \in [1, n], p_i \neq s_1$ **do**
5 $\quad AddDescendant(root, p_i)$;
6 $root$.scale$\leftarrow \lceil \log_2 root.\text{fddist} \rceil$;
7 Sort $root$.cds on descending order of norm;
8 $Expand(root)$;
9 **Procedure** *Expand*($\mathcal{N}$)**:**
10 $\quad$ **while** $\mathcal{N}$.*descs*$\neq \emptyset$ **do**
11 $\quad\quad p^* \leftarrow \arg\max_{p \in \mathcal{N}.\text{descs}} \|p\|$;
12 $\quad\quad$ Construct a node $\mathcal{N}_c$ as a child of $\mathcal{N}$;
13 $\quad\quad \mathcal{N}_c$.point$\leftarrow p^*, \mathcal{N}_c$.upoint$\leftarrow \frac{p^*}{\|p^*\|}$;
14 $\quad\quad \mathcal{N}_c$.scale$\leftarrow \mathcal{N}$.scale-1;
15 $\quad\quad$ **foreach** $p_i \in \mathcal{N}$.*descs* **do**
16 $\quad\quad\quad$ **if** $D(\frac{p_i}{\|p_i\|}, \mathcal{N}_c.\textit{upoint}) \leq 2^{\mathcal{N}_c.\textit{scale}}$ **then**
17 $\quad\quad\quad\quad AddDescendant(\mathcal{N}_c, p_i)$;
18 $\quad\quad\quad\quad$ Delete p_i from $\mathcal{N}$.desc;
19 $\quad\quad \mathcal{N}_c$.scale$\leftarrow \lceil \log_2 \mathcal{N}_c.\text{fddist} \rceil$;
20 $\quad\quad$ Sort $\mathcal{N}_c$.cds on descending order of norm;
21 $\quad$ **foreach** *child* $\mathcal{N}_c$ *of* $\mathcal{N}$ **do**
22 $\quad\quad Expand(\mathcal{N}_c)$;
23 **Procedure** *AddDescendant*($\mathcal{N}, p$)**:**
24 $\quad$ **if** $D(\mathcal{N}.\text{upoint}, \frac{p}{\|p\|}) \leq 2^{\delta}$ **then** add p into $\mathcal{N}$.cds;
25 $\quad$ **else** add p into $\mathcal{N}$.descs;
26 $\quad$ Update $\mathcal{N}$.fddist;

The next work is to prove the time complexity of the construction algorithm, and before that the following properties of the LRUS-CoverTree need to be proved. We first cite a well-known fact for the Euclidean space, denoted as Ball-cover fact.

Fact 1 (Ball-cover, [43]). *Denote a ball in $\mathbb{R}^d$ as $B(p,r) = \{x \in \mathbb{R}^d \mid D(p,x) \leq r\}$, calling p its center and r its radius. The Ball-cover fact says, any ball with radius r can be covered by at most $2^{O(d)}$ balls with radius $r/2$.*

From now on, denote c_d as the covering constant in $\mathbb{R}^d$, such that any ball with radius r can be covered by c_d balls with radius $r/2$. It is easy to see that $c_d = 2^{O(d)}$.

Lemma 2 (Children size bound). *For each node $\mathcal{N}$ of the LRUS-CoverTree, the number of children of $\mathcal{N}$ is at most c_d^2.*

Proof. Consider any LRUS-CoverTree node $\mathcal{N}$. By the Descendants Covering invariant, $\mathcal{N}$.descs can be covered by a ball with radius $2^{(\mathcal{N}.\text{scale})}$. By the Sibling Packing invariant, any ball with radius $2^{(\mathcal{N}.\text{scale}-2)}$ can not cover more than one child of $\mathcal{N}$. Combined with the ball-cover fact, the ball with radius $2^{(\mathcal{N}.\text{scale})}$ covering the descendants of $\mathcal{N}$ can be covered by at most c_d^2 balls of radius $2^{(\mathcal{N}.\text{scale}-2)}$, where each small ball covers at most one child of $\mathcal{N}$. In conclusion, the number of children for any node $\mathcal{N}$ is at most c_d^2. □

Lemma 3 (Descendant size bound). *Given a minimum scale δ, $|\mathcal{N}.descs| = (c_d)^{\mathcal{N}.scale-\delta+1}$ for each node $\mathcal{N}$.*

Proof. Given the minimum scale δ, every two points x, y represented by some LRUS-CoverTree node must satisfy $D(\frac{x}{\|x\|}, \frac{y}{\|y\|}) > 2^{\delta}$, since otherwise one of them would be included in the close descendant set of the other one. Thus, the number of descendants for node $\mathcal{N}$ can be bounded based on the ball-cover fact. By similar techniques used in Lemma 2, it can be proved that $|\mathcal{N}.\text{descs}| = (c_d)^{\mathcal{N}.\text{scale}-\delta+1}$. □

Lemma 4 (Tree size bound). *The number of nodes in a LRUS-CoverTree is at most $\min\{n, c_d^{2-\delta} + 1\}$.*

Proof. By the Unique Representation invariant, the number of tree nodes is at most n. By applying Lemma 3 on the root node, the maximum number of nodes is $c_d^{2-\delta} + 1$. Then this lemma is proved by taking minimum over the two values. □

Lemma 5 (Height bound). *Given the minimum scale $\delta > -\infty$, the height of LRUS-CoverTree is $2-\delta$. If $\delta = -\infty$, the height is at most $\lfloor \log_2 \frac{4}{Dmin} \rfloor$, where $Dmin = \min_{x,y \in P} \{D(\frac{x}{\|x\|}, \frac{y}{\|y\|}) > 0 \mid D(\frac{x}{\|x\|}, \frac{y}{\|y\|})\}$, that is the minimum non-zero pairwise distance among the unit vectors of input points.*

Proof. first notice that the close descendant set may still be non-empty if there exist $x, y \in P$ s.t. $D(\frac{x}{\|x\|}, \frac{y}{\|y\|}) = 0$. With the denotation of $Dmin$, the minimum possible furthest descendant distance of any node is then $Dmin$. Recalling the scale of node $\mathcal{N}$ is set to $\lceil \log_2 \mathcal{N}.\text{fddist} \rceil$, the minimum possible scale is $\lceil \log_2 Dmin \rceil$. The maximum height is then $2 - \lceil \log_2 Dmin \rceil = \lfloor \log_2 \frac{4}{Dmin} \rfloor$. □

Theorem 2. *The time complexity of constructing the LRUS-CoverTree is $O(c_d^2(2-\delta) \times n) = O(n)$.*

Proof. Fixing a point $p \in P$, let $\mathcal{N}_0, \mathcal{N}_1, \cdots, \mathcal{N}_l$ be a path where $\mathcal{N}_0$ is the root, and either $p = \mathcal{N}_l$.point or $p \in \mathcal{N}_l$.cds. For each node $\mathcal{N}_i$ along the path, and for each sibling node of $\mathcal{N}_i$ denoted as $\mathcal{N}_{ij}$, the $Expand$ function will compute distance between $\mathcal{N}_{ij}$.upoint and p. Thus, the maximum number of distance computations executed on point p is at most $|children| \times height$, where $|children|$ is the maximum number of children of any node and $height$ is the tree height. By applying the children bound and height bound, the number of distance computations incurred by each point is bounded by $c_d^2(2-\delta)$. The total time complexity of tree construction is then $O(c_d^2(2-\delta) \times n) = O(n)$. □

Theorem 3. *LRUS-CoverTree takes $O(n)$ space complexity.*

Proof. This theorem can be directly proved by the tree size bound in Lemma 4. □

V. k-MIPS ALGORITHM BASED ON LRUS-COVERTREE

A. Exact k-MIPS

The overall procedure of exact k-MIPS on the LRUS-CoverTree is a priority-first branch-and-bound algorithm. Different with depth-first or width-first search, the priority-first search uses a priority-queue to ensure that the node with current highest priority score would be chosen as the next node to visit. Some $Score$ functions are used to calculate the priority score for each node, which are not only used to guide the priority-first search but also used to prune the tree nodes. If the score of a node is lower than the candidate maximum inner-product threshold, the sub-tree rooted at this node will be pruned. The candidates of the maximum inner-product values are updated during the search, and the search will be terminated once the current highest score is no higher than the current candidate maximum inner-product threshold.

Intuitively, the $score$ of a LRUS-CoverTree node $\mathcal{N}$ is an upper bound on the inner-product value between query point q and any descendant points of $\mathcal{N}$. There are two requirements for calculating the $score$ for each node to obtain an efficient branch-and-bound algorithm. First, the $score$ must be an upper bound as tight as possible, since a looser bound will force the search to visit unnecessarily more nodes. Second, the $score$ must be calculated in $o(d)$ time where d is the number of dimension. Because calculating the inner-product value requires $\theta(d)$ time, it will be meaningless if the time of calculating an upper bound is the same with calculating the inner-product value.

To achieve the goal of calculating a tighter $score$ in $o(d)$ time, our solution is to store some pre-calculated information in each node of the LRUS-CoverTree, and record some useful information of each node during the search procedure to be utilized for its children nodes. The extra information is listed in Table II and Table III. Note that although a lot of extra information is stored in the LRUS-CoverTree node, the construction time is only raised by a constant factor.

Lemma 6. *An LRUS-CoverTree with all the extra information listed in Table II can be constructed in $O(c_d^2(2-\delta) \times n)$ time, which is the same with Theorem 2.*

Proof. The information listed in Table II is grouped into self information, parent information, descendant information, and

parent-descendant information. The time to create the four parts of information are considered separately as follows.
Parent information. They can be created in $O(d)$ time, since the parent node is known immediately when a node is constructed. The distance should use $O(d)$ time to compute and the others need only $O(1)$ time.
Descendant information. The furthest and longest descendant can be maintained in the $AddDescendant$ function in $O(1)$ time. The other descendant information can be computed in $O(1)$ time once the furthest descendant is determined.
In conclusion, maintaining the information listed in Table II only increases the constant but does not increase the asymptotic time complexity of tree construction. □

With the aid of the extra information listed in Table II and III, our exact k-MIPS algorithm is given in Algorithm 2. The algorithm maintains the candidate result set of k-MIPS denoted as $CandRes$, as well as the candidate inner-product threshold denoted as $CandIPThres$.

$$CandIPThres = \begin{cases} -\infty, & if\ |CandRes| < k \\ \min\limits_{p \in CandRes} \langle q, p \rangle, & otherwise \end{cases}$$

Algorithm 2: Exact k-MIPS on LRUS-CoverTree

Input: normalized query vector q, LRUS-CoverTree T, interger k.
Output: exact k-MIPS result.

Initialize empty priority queue Q;
$TryAddResult(root)$;
$VisitCloseDescendants(root)$;
$Q.push(CreateInfo(root))$;
while Q *is not empty* **do**
  $ParentInfo \leftarrow Q.pop()$;
  if $ParentInfo$**.***score*$\leq CandIPThres$ **then**
    break;
  foreach *child node* $\mathcal{N}_c$ *of* $ParentInfo$**.***node* **do**
    if $ScoreSubtree(\mathcal{N}_c, ParentInfo) \geq CandIPThres$ **then**
      $TryAddResult(\mathcal{N}_c)$;
      $VisitCloseDescendants(\mathcal{N}_c)$;
      $NodeInfo \leftarrow CreateInfo(\mathcal{N}_c)$;
      if $ScoreDescendants(\mathcal{N}_c, NodeInfo) \geq CandIPThres$ **then**
        $Q.push(NodeInfo)$;
Procedure $TryAddResult(\mathcal{N})$**:**
  $ipvalue \leftarrow \langle q, \mathcal{N}$**.**point$\rangle > CandIPThres$;
  if $ipvalue > CandIPThres$ **then**
    Update $CandRes$ and $CandIPThres$;
Procedure $CreateInfo(\mathcal{N})$**:**
  return a data structure containing the information listed in Table III for $\mathcal{N}$;

Algorithm 2 uses a priority queue to guide the search as mentioned before, where each element in the priority queue is a compound data structure storing the information

TABLE III: Search Information

notation	explanation
node	a visited node whose children are to be visited
score	the score of the node
dist	$D(q, node$**.**upoint$)$
cosine	$\cos(q, node$**.**point$)$
sine	$\sin(q, node$**.**point$)$
covered	boolean value indicating whether $D(q, node$**.**upoint$) \leq node$**.**fddist
ipvalue	the inner-product value $\langle q, node$**.**point$\rangle$

listed in Table III, and the $score$ information acts as the priority score. Each time the top element in the priority queue, denoted as $ParentInfo$, is visited, the children nodes of $ParentInfo$**.**node will be traversed. For each child node $\mathcal{N}_c$, an upper bound on the inner-product value of any point in the subtree rooted at $\mathcal{N}_c$ will be calculated using the $ScoreSubtree$ function. Only when the upper bound is larger than $CandIPThres$ will $\mathcal{N}_c$ be actually visited, otherwise it will be pruned.

Note that the root node is visited at the beginning of Algorithm to start the search. The visiting process of a node $\mathcal{N}$ includes three steps. The first step is to calculate $\langle q, \mathcal{N}$**.**point$\rangle$ and update $CandRes$ if $\langle q, \mathcal{N}$**.**point$\rangle > CandIPThres$, which is the functionality of $TryAddResult$ function. The second step is to traverse the close descendants of $\mathcal{N}$, i.e., $\mathcal{N}$**.**cds, where a pruning rule will be utilized. This function is denoted as $VisitCloseDescendants$. The third step is to check if the descendants of $\mathcal{N}_c$ are still valid to be visited using the $ScoreDescendants$ function. If so, the search information data structure will be created for $\mathcal{N}_c$ using the $CreateInfo$ function, and pushed into the priority queue.

The $VisitCloseDescendants$, $ScoreSubtree$ and $ScoreDescendants$ functions will be explained in more detail because they involve calculating upper bounds on the inner-product value.

1) $VisitCloseDescendants$: This function will traverse the close descendants of node $\mathcal{N}$, and invoke $TryAddResult$ on the examined point if the calculated upper bound on the inner-product value is higher than $CandIPThres$. The following lemma proves the effectiveness of the upper bound used in the pseudo codes.

Procedure $VisitCloseDescendants(\mathcal{N})$**:**
  $p \leftarrow \mathcal{N}$**.**upoint, $\delta \leftarrow$ the minimum scale;
  $\cos\theta \leftarrow 1 - 2^{2\delta}/2$, $\sin\theta \leftarrow \sqrt{1 - cos^2\theta}$;
  if $D(q,p) \leq 2^{\delta}$ **then**
    $cosmax \leftarrow 1$;
  else $cosmax \leftarrow \cos(q,p)\cos\theta + \sin(q,p)\sin\theta$;
  foreach c *in* $\mathcal{N}.cds$ **do**
    if $\|c\| \times cosmax \geq CandIPThres$ **then**
      $TryAddResult(\mathcal{N})$;
    else return;

Lemma 7. *Given a minimum scale δ, for a LRUS-CoverTree node $\mathcal{N}$ and query point q, denoting $p = \mathcal{N}.upoint$, any point*

$c \in \mathcal{N}$.cds satisfies $\cos(q,c) \le cosmax$, where

$$cosmax = \begin{cases} 1, & if\ D(p,q) \le 2^{\delta} \\ \cos(q,p)\cos\theta + \sin(q,p)\sin\theta, & otherwise \end{cases}$$

and $\cos\theta = 1 - 2^{2\delta}/2$, $\sin\theta = \sqrt{1-\cos^2\theta}$.
Besides, $cosmax$ can be computed in $O(1)$ time after $TryAddResult(\mathcal{N})$ is invoked.

Proof. 2^{δ} is the maximum distance between p and any close descendant point $c \in \mathcal{N}$.cds. Thus, if $D(q,p) \le 2^{\delta}$, it is possible that some $c \in \mathcal{N}$.cds can achieve the maximum cosine value which is 1. If $D(q,p) > 2^{\delta}$, the condition in Corollary 1 is satisfied since $D(c,p) \le 2^{\delta} < D(q,p)$, and the second term of $cosmax$ can be directly obtained by applying Corollary 1.

The information needed to calculate $cosmax$ includes $D(p,q)$, $\cos(p,q)$ and $\sin(p,q)$, which are all known after $TryAddResult(\mathcal{N})$ is invoked. $\cos\theta$ and $\sin\theta$ are known constants given the minimum scale δ. Thus, $cosmax$ can be computed in $O(1)$ time. □

By the pseudo codes of $VisitCloseDescendants$ function, the close descendants points are visited serially since they are already sorted in descending order of norm. Once the condition $\|c\| \times cosmax < CandIPThres$ happens, the rest of points in the close descendant set can not update the k-MIPS candidates since the norm of them are no more than $\|c\|$. Then it is safe to terminate the $VisitCloseDescendants$ function.

2) $ScoreSubtree$: The $ScoreSubtree(\mathcal{N})$ function is invoked before a child node $\mathcal{N}$ is visited, and only when the score is larger than $CandIPThres$ will $\mathcal{N}$ be actually visited. The $ScoreSubtree(\mathcal{N})$ function aims to calculate an upper bound on the inner-product value of any point in the subtree rooted at $\mathcal{N}$ with the aid of the parent node information. The following lemma proves the effectiveness of the bound.

```
1  Procedure ScoreSubtree(N, ParentInfo):
2      if ParentInfo.covered=false then
3          return N.norm×(ParentInfo.cosine×N.pfdcos+
4          ParentInfo.sine×N.pfdsin);
5      if sibling N_sib has covered query then
6          cosmax ← the bound given in Lemma 9;
7          return N.norm×cosmax;
8      else if ParentInfo.dist≤ N.pfddist then
9              return N.norm;
10         else return N.norm×
11         (ParentInfo.cosine×N.pfdcos+
12         ParentInfo.sine×N.pfdsin);
```

Lemma 8 (Parent bound)**.** *For any LRUS-CoverTree node $\mathcal{N}$, denoting $p = \mathcal{N}$.parent.upoint, for any point x in the sub-tree rooted at $\mathcal{N}$, it holds that $\langle q,x\rangle \le \mathcal{N}$.norm $\times$ $cosmax$, where*

$$cosmax = \begin{cases} 1, & if\ D(q,p) \le \mathcal{N}.pfddist \\ \cos(q,p)\times\mathcal{N}.pfdcos & \\ \quad + \sin(q,p) \times \mathcal{N}.pfdsin, & otherwise \end{cases}$$

Besides, $cosmax$ can be computed in $O(1)$ time with the parent node information provided.

Proof. Since $\langle q,x\rangle = \|x\|\cos(q,x)$, we place bound on norm and cosine respectively. First, by the Long Root invariant, it holds that $\|x\| \le \mathcal{N}$.norm since x is in the sub-tree rooted at $\mathcal{N}$. Second, to place bound on $\cos(q,x)$, consider whether the subtree rooted at $\mathcal{N}$ can cover the query point q by comparing $D(q,p)$ and $D(p,\mathcal{N}.\text{furdesc})$, where the latter one is exactly $\mathcal{N}$.pfddist. If $D(q,p) \le \mathcal{N}$.pfddist, there may exist a point x in the subtree rooted at $\mathcal{N}$ such that $\cos(q,x) = 1$. If $D(q,p) > \mathcal{N}$.pfddist, the bound on $\cos(q,x)$ is obtained by applying Corollary 1 on q,p and $\mathcal{N}$.furdesc. Also, note that $\cos(p,\mathcal{N}.\text{furdesc}) = \mathcal{N}$.pfdcos, and $\sin(p,\mathcal{N}.\text{furdesc}) = \mathcal{N}$.pfdsin which are pre-computed and stored in $\mathcal{N}$. Combining the two cases the bound on $cosmax$ is obtained. The bound on $\langle q,x\rangle$ is proved by multiplying the bound on $\|x\|$ and $\cos(q,x)$.

Finally, all the values needed to compute $cosmax$ are known by the time that $ScoreSubtree$ is invoked. Specifically, $D(q,p)$, $\cos(q,p)$ and $\sin(q,p)$ have been computed when the parent node was visited, and can be accessed in $O(1)$ time from the provided parent search information (denoted as $ParentInfo$ in the pseudo codes). Also, $\mathcal{N}$.pfddist, $\mathcal{N}$.pfdcos and $\mathcal{N}$.pfdsin can be obtained from the information stored in the node $\mathcal{N}$. Then it is true that $cosmax$ can be computed in $O(1)$ time with all the provided information. □

Lemma 9 (Sibling bound)**.** *For a LRUS-CoverTree node $\mathcal{N}$ with $\mathcal{N}$.scale=i, let the children of $\mathcal{N}$ be $\{\mathcal{N}_{c1}, \mathcal{N}_{c2}, \cdots \mathcal{N}_{cl}\}$. Let $\cos\theta_i = 1 - (2^{2(i-1)})/2$ and $\sin\theta_i = \sqrt{1-\cos^2\theta_i}$. If there exists a child node $\mathcal{N}_{cj}$, $1 \le j \le l$, such that $D(q, \mathcal{N}_{cj}.upoint) \le 2^{i-1}$, denoting $p_{sib} = \mathcal{N}_{cj}.upoint$, then for any point x in the subtree rooted at any other sibling node $\mathcal{N}_c \ne \mathcal{N}_{cj}$, it holds that $cos(x,q) \le \cos(q,p_{sib})\cos\theta_i + \sin(q,p_{sib})\sin\theta_i$. Besides, the bound is available for all the other sibling nodes in $O(1)$ time after $\mathcal{N}_{cj}$ is visited.*

Proof. First, it holds that $D(\frac{x}{\|x\|}, p_{sib}) > 2^{i-1}$, since otherwise x would be a descendant of $\mathcal{N}_{cj}$. Noticing that the condition in Corollary 1 is satisfied since $D(q,p_{sib}) \le 2^{i-1} < D(\frac{x}{\|x\|}, p_{sib})$, the lemma can be proved by applying Corollary 1 on q,x and p_{sib}. Finally, since $\cos\theta_i$ and $\sin\theta_i$ are constants, and $\cos(q,p_{sib})$ $\sin(q,p_{sib})$ are known after $\mathcal{N}_{cj}$ is visited, it is true that the bound given in the lemma can be computed in $O(1)$ time and can be used for the other sibling nodes. □

3) $ScoreDescendants$: The $ScoreDescendants$ function is invoked after a node $\mathcal{N}$ is visited, to calculate an upper bound on the inner-product value of any descendant point of $\mathcal{N}$. Only when the score is larger than $CandIPThres$ will the search information of $\mathcal{N}$ be pushed into the priority queue for further search procudure, otherwise the descendants will be pruned. The following lemma explains the bound used in the $ScoreDescendants$ function.

```
1 Procedure ScoreDescendants(𝒩, NodeInfo):
2     if NodeInfo.dist≤ 𝒩.fddist then return 𝒩.ldn;
3     else return 𝒩.ldn×(NodeInfo.cosine×𝒩.fdcos
4                        +NodeInfo.sine×𝒩.fdsin);
```

Lemma 10. *Denoting $p = \mathcal{N}$.upoint, For any point x that is a descendant of $\mathcal{N}$, it holds that $\langle q, x\rangle \leq \mathcal{N}$.ldn$\times cosmax$, where*

$$cosmax = \begin{cases} 1, \ if\ D(q,p) \leq \mathcal{N}.fddist \\ \cos(q,p) \times \mathcal{N}.fdcos \\ \quad + \sin(q,p) \times \mathcal{N}.fdsin, \ otherwise \end{cases}$$

Besides, the $cosmax$ value can be computed in $O(1)$ time with the information of $\mathcal{N}$ provided.

Proof. The proof is similar with Lemma 8. The upper bound on the norm of descandant points is now $\mathcal{N}$.ldn. For the bound on cosine, the two cases for whether the descendants of $\mathcal{N}$ can cover the query are considered. For the case where $D(q,p) \leq \mathcal{N}$.fddist, it is easy to see $cosmax = 1$. For the other case where $D(q,p) > \mathcal{N}$.fddist, Corollary 1 is applied on q, p and $\mathcal{N}$.furdesc to obtained the desired bound. Also, $cosmax$ can be computed in $O(1)$ time since the information needed to compute it can either be obtained from the search information created for $\mathcal{N}$ (denoted as $NodeInfo$ in the pseudo codes), or have been pre-computed and stored in node $\mathcal{N}$. □

By now we have proved the effectiveness of all the bounds used in Algorithm 2, then the correctness of Algorithm 2 can be easily proved. The time complexity of the algorithm is $O(n)$ in worst case since it only uses branch-and-bound techniques. However, Algorithm 2 runs fast empirically due to the effectiveness of the bounds. See the experimental results in Section VI for validation.

B. Approximate k-MIPS

The approximate k-MIPS algorithm is given in Algorithm 3, which uses almost the same algorithm framework with the exact k-MIPS algorithm. The only differences lie in Line 8, 10 and 14, where the stop condition related to the $score$ functions involves the approximation factor ϵ. The following lemma proves the correctness of the approximate algorithm.

Theorem 4. *Algorithm 3 correctly finds $MIP_\epsilon(k, q, P)$ in accordance with Definition 2.*

Proof. Denote $True\text{-}kMIP$ and $Appr\text{-}kMIP$ as the exact and approximate k-th largest inner-product value, respectively. Let $Visited$ be the set of nodes visited by the algorithm, then it is true that the algorithm returns the k points with maximum inner-product value among the visited points, i.e., $MIP_\epsilon(k, q, P) = MIP(k, q, Visited)$. Then, it only needs to prove that $\epsilon \times \langle q, p\rangle \leq Appr\text{-}kMIP$ for those unvisited point p. First of all, by the definition of $CandIPThres$ we have $CandIPThres \leq Appr\text{-}kMIP$. Then we consider the following two cases that a node may be pruned.

Algorithm 3: Approximate k-MIPS on LRUS-CoverTree

Input: normalized query vector q, LRUS-CoverTree T, interger k, and approximate factor $\epsilon \leq 1$.
Output: approximate k-MIPS results.

```
1  Initialize empty priority queue Q;
2  TryAddResult(root);
3  VisitCloseDescendants(root);
4  Q.push(CreateInfo(root));
5  while Q is not empty do
6      ParentInfo ← Q.pop();
7      if ϵ × ParentInfo.score < CandIPThres then
8          break;
9      foreach child node 𝒩_c of ParentInfo.node do
10         if ϵ × ScoreSubtree(𝒩_c, ParentInfo)
             ≥ CandIPThres then
11             TryAddResult(𝒩_c);
12             VisitCloseDescendants(𝒩_c);
13             NodeInfo ← CreateInfo(𝒩_c);
14             if ϵ × ScoreDescendants(𝒩_c, NodeInfo)
                 ≥ CandIPThres then
15                 Q.push(NodeInfo);
```

Case 1: the algorithm terminates at Line 8, and all points which are descendants of a node in the remaining priority queue are pruned. Let $\mathcal{N}_a$ be the ancestor node in the priority queue of a pruned point p, and let $score(\mathcal{N}_a)$ be the score of $\mathcal{N}_a$ stored in the search information data structure associated with $\mathcal{N}_a$.

Case 2: a subtree is pruned at Line 10 or 14. Let $\mathcal{N}$ be the pruned subtree root and p be one of its descendant point, and let $score$ be the score computed by $ScoreSubtree$ or $ScoreDescendants$ for node $\mathcal{N}$. By the definition of the score functions, it holds that $\langle q, p\rangle \leq score$. By the pruning condition at Line 10 or 14, it holds that $\epsilon \times score \leq CandIPThres \leq Appr\text{-}kMIP$.

By now the lemma is proved. □

The next work is to analyze the time complexity of the approximate k-MIPS algorithm. We need to prove the following lemma providing another bound on the inner-product value.

Lemma 11. *For any node $\mathcal{N}$ in a LRUS-CoverTree, denoting $p = \mathcal{N}$.point, for any point $x \in \mathcal{N}$.descs, it holds that $\langle q, x\rangle \leq \langle q, p\rangle + \|p\| \times 2^{\mathcal{N}.scale}$.*

Proof. By the linearity of inner-product function on real valued vectors, it holds that $\langle q, x\rangle = \langle q, p\rangle + \langle q, x - p\rangle$, where $x - p$ is the vector difference of x and p. Thus, it only needs to prove $\langle q, x-p\rangle \leq \|p\| \times 2^{\mathcal{N}.\text{scale}}$. We consider the following two cases.

Case 1: $\mathcal{N}$.fddist$\geq D(q, \frac{p}{\|p\|})$. By Lemma 10, in this case we have $\langle q, x\rangle \leq \mathcal{N}$.ldn$\leq \mathcal{N}$.norm $= \|p\|$, where the latter inequality is based on the Long Root invariant. Then we have $\langle q, x - p\rangle = \langle q, x\rangle - \langle q, p\rangle \leq \|p\| - \|p\| \cos(q, p)$. Knowing that $\cos(q, p) = 1 - \frac{D(q, p/\|p\|)^2}{2}$, we then have

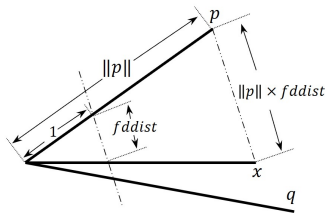


Fig. 3: The demonstration of Case 2 in Lemma 11.

$\langle q, x-p\rangle \leq \|p\|\frac{D(q,p/\|p\|)^2}{2} \leq \|p\|D(q,\frac{p}{\|p\|})$, where the last inequality is based on $D(q,\frac{p}{\|p\|}) \leq 2$ since q and $\frac{p}{\|p\|}$ are both unit vectors. Finally, by the given condition in this case, $D(q,\frac{p}{\|p\|}) \leq \mathcal{N}.\text{fddist} \leq 2^{\mathcal{N}.\text{scale}}$, and it is proved that $\langle q, x-p\rangle \leq \|p\| \times 2^{\mathcal{N}.\text{scale}}$.

Case 2: $\mathcal{N}.\text{fddist} < D(q,\frac{p}{\|p\|})$. In this case, since $\langle q,x\rangle = \|x\|\cos(q,x)$, the maximum possible value of $\langle q,x\rangle$ is obtained when $\|x\|$ and $\cos(q,x)$ both takes the maximum possible value. On one hand, $\|x\| \leq \|p\|$. On the other hand, $\cos(q,x)$ achieves maximum when $D(\frac{p}{\|p\|},\frac{x}{\|x\|})$ achieves maximum which is $\mathcal{N}.\text{fddist}$. This extreme case is demonstrated in Figure 3. It can be noticed there is a pair of similar triangles, one formed by the vectors p, x, and the other formed by the unit vectors $\frac{p}{\|p\|},\frac{x}{\|x\|}$. Thus, by the property of similar triangles, we have $\|x-p\| = \|p\| \times D(\frac{p}{\|p\|},\frac{x}{\|x\|}) \leq \|p\| \times \mathcal{N}.\text{fddist} \leq \|p\| \times 2^{\mathcal{N}.\text{scale}}$. Finally, we have $\langle q, x-p\rangle \leq \|x-p\| \leq \|p\| \times 2^{\mathcal{N}.\text{scale}}$.

In both cases, we have proved the desired inequality $\langle q, x-p\rangle \leq \|p\| \times 2^{\mathcal{N}.\text{scale}}$, and thus the lemma is proved. □

The bound given in Lemma 11 is looser than that proved in Lemma 8, 9 and 10, but it is provided in a good form. Lemma 11 says that, $\langle q,p\rangle$ is an approximation of $\langle q,x\rangle$ where x is any descendant point, and the approximation error is bounded by the multiplication of norm and covering distance. Also, the approximation error drops exponentially as the tree is descended. This good property assures the good theoretical efficiency of the approximate k-MIPS algorithm.

Lemma 12. *For the approximate k-MIPS problem defined in Definition 2, Algorithm 3 visits at most $\log_2 \frac{4}{(1-\epsilon)\epsilon}$ levels of nodes to achieve the desired approximation quality. That is, denoting $x^* \in P$ and $x_M \in P$ as the points with k-th largest inner-produce value in exact k-MIPS result and approximate k-MIPS result found by the algorithm, respectively, it holds that $\langle q, x_M\rangle \geq \epsilon\langle q, x^*\rangle$.*

Proof. If $x_M = x^*$ then the approximate algorithm finds the true exact result. Otherwise, x^* is unvisited and must be a descendant of some visited but pruned node $\mathcal{N}$. Assume that $\mathcal{N}$ is the one node among the visited ancestors of x with the lowest scale, and denote $\mathcal{N}.\text{scale} = i$. Since the descendants of $\mathcal{N}$ are pruned by the algorithm, it must be the case that $\epsilon \times ScoreDescendants(\mathcal{N}) \leq CandIPThres$ according to the pseudo codes. On one hand, it is easy to see $CandIPThres \leq \langle q, x_M\rangle$. On the other hand, by Lemma 10, $ScoreDescendants(\mathcal{N}) \leq \mathcal{N}.\text{norm}$. We consider the relaxed case that $ScoreDescendants(\mathcal{N}) = \mathcal{N}.\text{norm}$, where the looser bound must force the algorithm to visit more nodes. In such a case, we have $\epsilon \times \mathcal{N}.\text{norm} \leq \langle q, x_M\rangle$.

Now, by Lemma 11, it holds that $\langle q, x^*\rangle \leq \langle q, \mathcal{N}.\text{point}\rangle + \mathcal{N}.\text{norm} \times 2^i \leq \langle q, x_M\rangle + \mathcal{N}.\text{norm} \times 2^i$. Recall the assumption that no negative inner-product value exists, and we have $\frac{\langle q,x_M\rangle}{\langle q,x^*\rangle} \geq 1 - \frac{\mathcal{N}.\text{norm}\times 2^i}{\langle q,x^*\rangle} \geq 1 - \frac{\mathcal{N}.\text{norm}\times 2^i}{\langle q,x_M\rangle} \geq 1 - \frac{2^i}{\epsilon}$. To make the approximation quality better than ϵ, it suffices to let $1 - \frac{2^i}{\epsilon} \geq \epsilon \Rightarrow i \leq \log_2(1-\epsilon)\epsilon$.

Finally, notice that the above bound is obtained by letting $ScoreDescendants(\mathcal{N}) = \mathcal{N}.\text{norm}$ which is a looser bound. For our true algorithm given in Algorithm 3, it must visit less nodes. Then it is proved that Algorithm 3 visits at most $2 - \log_2(1-\epsilon)\epsilon = \log_2 \frac{4}{(1-\epsilon)\epsilon}$ levels of nodes to achieve the desired approximation quality. □

Theorem 5. *LRUS-CoverTree based approximate k-MIPS algorithm runs in $O\left((\frac{4}{(1-\epsilon)\epsilon})^d + |CloseDesc|\right)$ time, where $|CloseDesc|$ denotes the total number of close descendant points included in the LRUS-CoverTree.*

Proof. By combining Lemma 12 and 2, the number of tree nodes visited by the algorithm is $O((\frac{4}{(1-\epsilon)\epsilon})^d)$. The number of close descendant points visited is obviously $O(|CloseDesc|)$. Then the desired complexity follows. □

The $O(|CloseDesc|)$ factor in Theorem 5 is closely related to data distribution and also the minimum scale parameter δ. The other part $O((\frac{4}{(1-\epsilon)\epsilon})^d)$ is independent with the input size n but exponentially dependent with the number of dimension d. However, the empirical running time of our algorithm does not grow exponentially with d. Actually, it is a looser upper bound since it counts the number of nodes all through the lowest visited scale, but many nodes in these scales may not be visited. See the next section for experimental results.

VI. Experiments

A. Preparations

1) Our methods: The proposed exact and approximate algorithms are denoted as *LRUS-E* and *LRUS-A*, respectively (codes in [44]). Note that *LRUS-E* can be regarded as a special case of *LRUS-A* setting the approximation factor $\epsilon = 1$.

2) Methods compared: Since LSH, similarity graph and quantization based methods can only find approximate k-MIPS results, the main focus of the experiments is to compare *LRUS-A* with the state-of-the-art of the three kinds of methods. *FARGO* [26] is the state-of-the-art LSH based method, and in that paper the authors stated that it outperforms all the LSH based methods they compared. However, similarity graph and quantization based methods are not compared with *FARGO* in their paper [26]. *ip-nsw+* [33] and *NAPG* [34] are the state-of-the-art similarity graph based methods. *ScaNN* is an open-source library that implements the state-of-the-art quantization based method [36]. The codes of the above methods are available online [45]–[48]. *RPT* [9] is a tree based method usually used in recent related works to compare against, and we implement this method based on the Random Projection

TABLE IV: datasets information summarization

dataset	**audio**	**gist**	**tiny5m**	**tencentwordvec**	**yahoomusic**
number	54387	1000000	5000000	12287936	136736
dimension	192	960	384	100	300

Tree implemented in MLPACK [49]. For the exact method, *LRUS-E* is only compared with the cover tree based algorithm [12] denoted as *covertree*, to show the advantage of LRUS-CoverTree against the original cover tree on k-MIPS problem.

3) Environment setup: All the experiments ran on a machine with Intel Core i7-10700 CPU and 128GB RAM running Ubuntu 20.04 operating system. All the codes except for *ScaNN* are written in C++ and compiled using g++ in C++ 20 standard with -O3 optimization. The *ScaNN* library is invoked by Python bindings but written in C, and it is still reasonable to compare against.

4) Datasets: 5 real-world datasets [50] are used which have also been commonly used in recent related works to evaluate the performance of k-MIPS algorithms. These datasets cover a variety of application scenarios of k-MIPS. **audio** is audio wave data which can be regarded as times series. **gist** and **tiny5m** are pixel-wise image data. **tencentwordvec** is a dataset of word embedding vectors for Chinese words [51]. **yahoomusic** is the latent factors of user preferences for musics on the yahoo music website which is used for recommendation. The information of these datasets are listed in Table IV.

5) Comparison metrics: The comparison metrics considered here include *preprocessing time*, *query time*, *recall* and *ratio*. Recall is defined as $\frac{|MIP_\epsilon(k,q,P)\cap MIP(k,q,P)|}{k}$. Ratio is defined as $\frac{1}{k}\sum_{i=1}^{k}\frac{\langle q,x_i\rangle}{\langle q,x_i^*\rangle}$, where $\langle q,x_i\rangle$ and $\langle q,x_i^*\rangle$ are the i-th largest inner-product value in the approximate and exact result sets, respectively.

6) Default parameters: The parameter k of k-MIPS is varied from 10 to 150 with gap 20, and is defaulted to 50. From each dataset 200 points are randomly chosen as the query points. The tunable parameters of each method are kept as the same with the codes available online and left unchanged for all datasets.

7) Varying parameters to obtain the curves: The tested methods vary specific parameter to control the search time and accuracy and obtain the recall-time and ratio-time curves. For *LRUS-A*, the curves are obtained by setting the approximation parameter $\epsilon \in \{0.7, 0.8, 0.85, 0.9, 0.95, 1\}$. For *ip-nsw+* and *NAPG*, the curves are obtained by varying the search parameter called ef [34]. For *ScaNN*, the search time and accuracy is controlled by the parameter $leaves\_to\_search$ [36].

B. Self evaluation

1) Influence of the minimum scale: There exists one hyper-parameter for our method which is the minimum scale δ. The tree construction time and exact result query time for different minimum scales are summarized in Table V. The recall-time and ratio-time curves of *LRUS-A* for different minimum scales are shown in Figure 4 and 5. Note that in the recall-time and ratio-time figures, the curves residing in the top-left part are preferred, since it means the algorithm can reach the same recall/ratio with less query time.

TABLE V: Comparing exact methods

(a) Tree construction time (in seconds)

		audio	**gist**	**tiny5m**	**tencentwordvec**	**yahoomusic**
LRUS-E	$\delta=-4$	0.458	2857.77	4279.98	1359.03	1.370
	$\delta=-3$	0.410	2847.21	5060.84	1135	1.088
	$\delta=-2$	0.363	106.245	116.708	231.802	0.845
	$\delta=-1$	0.326	6.934	11.521	29.898	0.484
	$\delta=0$	0.185	3.037	6.471	7.987	0.176
covertree		126.89	>24 hours	>24 hours	>24 hours	297.16

(b) Query time (in milliseconds, $k=50$)

		audio	**gist**	**tiny5m**	**tencentwordvec**	**yahoomusic**
LRUS-E	$\delta=-4$	0.392	4.738	29.157	27.335	2.227
	$\delta=-3$	0.304	4.822	21.443	26.881	3.879
	$\delta=-2$	0.384	3.042	19.784	19.001	7.705
	$\delta=-1$	0.797	5.054	24.241	23.904	14.187
	$\delta=0$	0.962	3.381	23.850	18.311	18.174
covertree		4.83	no result	no result	no result	14.91

Since δ is a cut-off value of the minimum scale and controls the tree height, the tree construction time drops as δ increases. It can be noticed that the tree construction time for the large datasets **gist**, **tiny5m** and **tencentwordvec** drop significantly when $\delta=-2$. This indicates that lots of unit vectors in these two datasets are within 2^{-2} distance with each other.

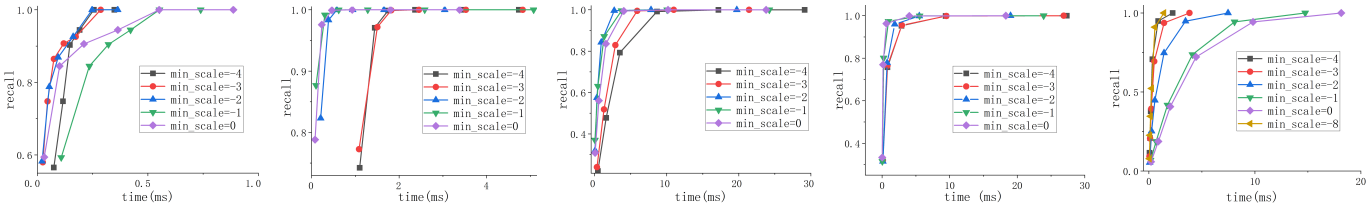
When δ becomes too large the LRUS-CoverTree will become too flat to support efficient query. For example, when $\delta=0$ there are only 2 levels in LRUS-CoverTree, and the construction time is minimized but the query on LRUS-CoverTree becomes close to linear scanning the input points. This phenomenon can be verified from Table V(b), and also by the recall-time and ratio-time curves in Figure 4 and 5. Consequently, there exists a best choice of δ that balances the tree construction time and query performance. According to the empirical results here, the best choice of δ is -2 for **audio** and **gist**, -1 for **tiny5m** and **tencentwordvec**, and -8 for **yahoomusic**. These settings of δ are adopted when comparing against the other k-MIPS methods.

C. Comparison

1) Exact methods: Form the results shown in Table V, it can be concluded that *LRUS-E* significantly outperforms the cover tree based k-MIPS method, both in tree construction time and query time. LRUS-CoverTree can be regarded as a variant of the cover tree, and its dedicated design for k-MIPS problem indeed leads to significant improvement against its original version.

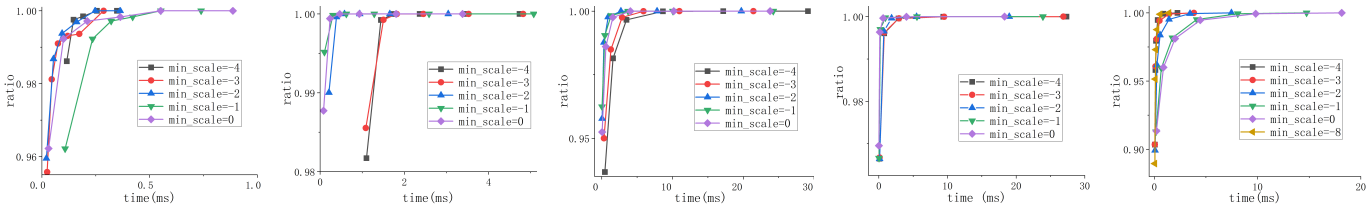
2) Approximate methods: First we must note that the *RPT* method is indeed not comparable with other methods. It runs out of memory on **gist**, **tiny5m** and **tencentwordvec**, and runs two orders of magnitude slower than all the other methods on **audio** and **yahoomusic**. We have to drop the experimental results of *RPT* to improve the readability of figures.

By the experimental results shown in Figure 6 and 7, the most notable observation is that *LRUS-A* outperforms the state-of-the-art LSH based *FARGO* method on all the five datasets. The curves for *LRUS-A* lie strictly in the top-left part to the curves for *FARGO*. Compared with the other methods, the ratio-time performance of *LRUS-A* is relatively favorable since



(a) audio (b) gist (c) tiny5m (d) tencentwordvec (e) yahoomusic

Fig. 4: Recall-time curve of LRUS-A for different minimum scales (k=50).



(a) audio (b) gist (c) tiny5m (d) tencentwordvec (e) yahoomusic

Fig. 5: Ratio-time curve of LRUS-A for different minimum scales (k=50).

TABLE VI: Preprocessing time (in seconds)

	LRUS-A	*FARGO*	*ip-nsw+*	*NAPG*	*ScaNN*
audio	**0.363** ($\delta=-2$)	1.33684	9.933	20.619	4.343
gist	**3.042** ($\delta=-2$)	112.064	1028.45	2702.29	189.305
tiny5m	**11.521** ($\delta=-1$)	226.649	5365.44	41340.4	297.950
tencentwordvec	**29.898** ($\delta=-1$)	161.415	6631.43	4179.56	192.07
yahoomusic	**1.350** ($\delta=-8$)	4.90903	37.9303	755.139	14.57786

TABLE VII: Extra memory usage (in MB)

	LRUS-A	*FARGO*	*ip-nsw+*	*NAPG*	*ScaNN*
audio	**26.58** ($\delta=-2$)	78.17	27.17	64.17	513.64
gist	735.46 ($\delta=-2$)	**642.89**	4178.16	7914.17	12196.26
tiny5m	**1559.93** ($\delta=-1$)	2296.78	9506.17	16651.17	22674.24
tencentwordvec	**1426.10** ($\delta=-1$)	4842.52	14367.17	13131.17	14548.20
yahoomusic	**87.90** ($\delta=-8$)	92.52	344.17	331.17	1058.90

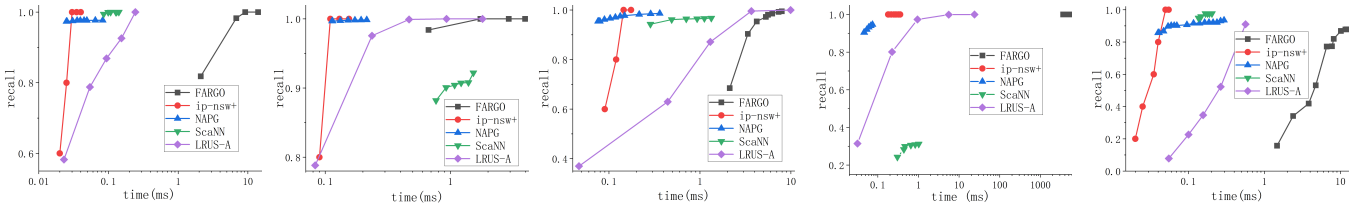
the ratio-time curves for *LRUS-A* stick close to those for *ip-nsw+*, *NAPG* and *ScaNN*. Note that *NAPG* usually reports a ratio exceeding 1 which is impossible according to the definition. This issue may be attributed to float point precision problem. On the other hand, we have to admit that our method *LRUS-A* are not better on recall-time performance than *ip-nsw+*, *NAPG* and *ScaNN* on the tested datasets. The recall-time curves for those methods lie strictly in the top-left part to the curves for *LRUS-A*. An exception is that *ScaNN* performs poorly on **tencentwordvec**, indicating it may be unsuitable for large datasets. The reason for *LRUS-A* achieving higher ratio but lower recall is that *LRUS-A* is designed to satisfy Definition 2 which is closely related to the ratio metric. *LRUS-A* will terminate once the ratio condition is met, leaving a lot of points unvisited and resulting in a lower recall.

Although the query performance of *LRUS-A* is only comparable in ratio and not better in recall than similarity graph and quantization based methods, it is achieved by far low preprocessing cost and relatively small memory usage. From Table VI, the preprocessing time of *LRUS-A* method is one order of magnitude lower than *ScaNN*, and more than two order of magnitudes lower than *ip-nsw+* and *NAPG*. From Table VII, the extra memory usage (total memory usage minus input data size) of *LRUS-A* is the lowest except on **gist**. Note the extra memory usage for *LRUS-A* includes the tree index and the other data structures used in search procedure such as the priority queue. Considering the significantly low preprocessing cost and relatively low memory usage, the query performance of *LRUS-A* is quite plausible.

D. Summary of experimental results

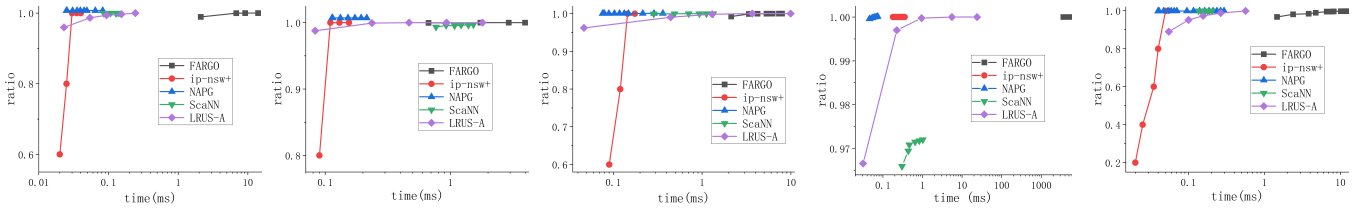
Summarizing the analysis above, the primary goal of this paper which is to design an efficient tree structure for k-MIPS is successfully achieved. The performance of LRUS-CoverTree indeed tremendously surpasses the existing tree based methods such as cover tree and random partition tree. By setting the minimum scale δ to an appropriate value (usually $\delta = -2$ is good enough), *LRUS-A* also outperforms the state-of-the-art LSH based method, which has not been compared with similarity graph and quantization based methods. Although *LRUS-A* can only achieve comparable but not better query efficiency than *ip-nsw+*, *NAPG* and *ScaNN*, the preprocessing time of *LRUS-A* is far less than them.

Here we mention an important scenario where our method shows advantage, which is the Retrieval Augmented Generation method [52], [53] for Large Language Model (LLM). A vector database stores the vector embeddings of a large corpus. When the user raises a question to LLM, the vectors with highest similarity with the question vector should be retrieved from the vector database using k-MIPS method. Due to the intrinsic synonymy phenomenon of natural language, there



(a) audio (b) gist (c) tiny5m (d) tencentwordvec (e) yahoomusic

Fig. 6: recall-time curve (k=50).



(a) audio (b) gist (c) tiny5m (d) tencentwordvec (e) yahoomusic

Fig. 7: ratio-time curve (k=50).

will be multiple vectors similar to a given question vector which represent a group of synonyms, and the inner-product values of them are close to each other. Thus, if the overall ratio is high enough, it is very likely that near-synonyms are found, although not the exact word with the highest similarity to the question vector. It then means that a lower recall is not a big problem if the overall ratio is high. Also, the vector database is updated regularly to ensure the LLM can access the up-to-date knowledge, which requires a low index construction time when the data is updated. In such a case, our method *LRUS-A* is prominent since it has the lowest construction time.

After all, our *LRUS-A* method is a great improvement against former tree based k-MIPS methods which makes tree based methods a comparable choice against the other kinds of methods for k-MIPS.

VII. Conclusion

In this paper we proposed a novel tree structure dedicated for k-MIPS named LRUS-CoverTree, and proved new upper-bounds on the inner-product value which are designed based on the properties of LRUS-CoverTree. Using LRUS-CoverTree as the index structure, new exact and approximate k-MIPS algorithms are proposed. The theoretical foundation of the LRUS-CoverTree and the new k-MIPS algorithms are rigorously established in this paper. Experimental results show that our proposed algorithms significantly outperform existing tree based and LSH based methods, and achieve comparable query performance with the other state-of-the-art methods for k-MIPS. Most notably, the desirable query performance is achieved by far lower preprocessing time than similarity graph and quantization based methods. In conclusion, the LRUS-CoverTree remarkably improves the performance of tree based methods for k-MIPS, challenging the common belief that tree based methods can not perform well for k-MIPS problem.

References

[1] Y. Cao, M. Long, J. Wang, and S. Liu, "Deep visual-semantic quantization for efficient image retrieval," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1328–1337.

[2] Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins, "Sparse, dense, and attentional representations for text retrieval," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 329–345, 2021.

[3] M. Zhang, W. Wang, X. Liu, J. Gao, and Y. He, "Navigating with graph representations for fast and scalable decoding of neural language models," *Advances in neural information processing systems*, vol. 31, 2018.

[4] W. Krichene, N. Mayoraz, S. Rendle, L. Zhang, X. Yi, L. Hong, E. Chi, and J. Anderson, "Efficient training on very large corpora via gramian estimation," *arXiv preprint arXiv:1807.07187*, 2018.

[5] M. H. Abdi, G. Okeyo, and R. W. Mwangi, "Matrix factorization techniques for context-aware collaborative filtering recommender systems: A survey," 2018.

[6] S. Kläbe, S. Hagedorn, and K.-U. Sattler, "Exploration of approaches for in-database ml," in *Proceedings of the 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-March 31*, 2023.

[7] K. Hirata, D. Amagata, and T. Hara, "Cardinality estimation in inner product space," *IEEE Open Journal of the Computer Society*, vol. 3, pp. 208–216, 2022.

[8] S. Dasgupta and Y. Freund, "Random projection trees and low dimensional manifolds," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, ser. STOC '08. New York, NY, USA: ACM, 2008, pp. 537–546.

[9] O. Keivani, K. Sinha, and P. Ram, "Improved maximum inner product search with better theoretical guarantee using randomized partition trees," *Machine Learning*, vol. 107, no. 6, pp. 1069–1094, 2018.

[10] P. Ram and A. G. Gray, "Maximum inner-product search using cone trees," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 931–939.

[11] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 97–104.

[12] R. R. Curtin, P. Ram, and A. G. Gray, "Fast exact max-kernel search," in *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 2013, pp. 1–9.

[13] T. D. Ahle, R. Pagh, I. Razenshteyn, and F. Silvestri, "On the complexity of inner product similarity join," in *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2016, pp. 151–164.

[14] L. Chen, "On the hardness of approximate and exact (bichromatic) maximum inner product," *arXiv preprint arXiv:1802.02325*, 2018.

[15] P. Indyk and R. Motwani, "Approximate nearest neighbors," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC '98*. New York, New York, USA: ACM Press, 1998, pp. 604–613. [Online]. Available: http://portal.acm.org/citation.cfm?doid=276698.276876

[16] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry - SCG '04*. New York, New York, USA: ACM Press, 2004, p. 253. [Online]. Available: http://portal.acm.org/citation.cfm?doid=997817.997857

[17] A. Andoni, T. Laarhoven, I. Razenshteyn, and E. Waingarten, "Optimal Hashing-based Time-Space Trade-offs for Approximate Near Neighbors," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA: Society for Industrial and Applied Mathematics, jan 2017, pp. 47–66.

[18] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases*, 2007, pp. 950–961.

[19] B. Neyshabur and N. Srebro, "On symmetric and asymmetric lshs for inner product search," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1926–1934.

[20] X. Yan, J. Li, X. Dai, H. Chen, and J. Cheng, "Norm-ranging lsh for maximum inner product search," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[21] Q. Huang, G. Ma, J. Feng, Q. Fang, and A. K. Tung, "Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1561–1570.

[22] N. Pham, "Simple yet efficient algorithms for maximum inner product search via extreme order statistics," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1339–1347.

[23] A. Shrivastava and P. Li, "Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips)," *Advances in neural information processing systems*, vol. 27, 2014.

[24] ——, "Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips)," *arXiv preprint arXiv:1410.5410*, 2014.

[25] 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 *Proceedings of the 8th ACM Conference on Recommender systems*, 2014, pp. 257–264.

[26] 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," *Proceedings of the VLDB Endowment*, vol. 16, no. 5, pp. 1100–1112, 2023.

[27] Y. Song, Y. Gu, R. Zhang, and G. Yu, "Promips: Efficient high-dimensional c-approximate maximum inner product search with a lightweight index," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 1619–1630.

[28] J. W. Jaromczyk and G. T. Toussaint, "Relative neighborhood graphs and their relatives," *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1502–1517, 1992.

[29] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014.

[30] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[31] D. P. Dobkin, S. J. Friedman, and K. J. Supowit, "Delaunay graphs are almost as good as complete graphs," *Discrete & Computational Geometry*, vol. 5, pp. 399–407, 1990.

[32] S. Morozov and A. Babenko, "Non-metric similarity graphs for maximum inner product search," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[33] J. Liu, X. Yan, X. Dai, Z. Li, J. Cheng, and M.-C. Yang, "Understanding and improving proximity graph based maximum inner product search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, 2020, pp. 139–146.

[34] S. Tan, Z. Xu, W. Zhao, H. Fei, Z. Zhou, and P. Li, "Norm adjusted proximity graph for fast inner product retrieval," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1552–1560.

[35] Y. Guo, G. Ding, and J. Han, "Robust quantization for general similarity search," *IEEE Transactions on Image Processing*, vol. 27, no. 2, pp. 949–963, 2017.

[36] R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar, "Accelerating large-scale inference with anisotropic vector quantization," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3887–3896.

[37] J. Zhang, Q. Liu, D. Lian, Z. Liu, L. Wu, and E. Chen, "Anisotropic additive quantization for fast inner product search," in *Proceedings of the AAAI conference on Artificial Intelligence*, vol. 36, no. 4, 2022, pp. 4354–4362.

[38] M. Tiwari, R. Kang, J.-Y. Lee, D. Lee, C. Piech, S. Thrun, I. Shomorony, and M. J. Zhang, "Faster maximum inner product search in high dimensions," *arXiv preprint arXiv:2212.07551*, 2022.

[39] R. Liu, T. Wu, and B. Mozafari, "A bandit approach to maximum inner product search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4376–4383.

[40] H.-F. Yu, C.-J. Hsieh, Q. Lei, and I. S. Dhillon, "A greedy approach for budgeted maximum inner product search," *Advances in neural information processing systems*, vol. 30, 2017.

[41] G. Ballard, T. G. Kolda, A. Pinar, and C. Seshadhri, "Diamond sampling for approximate maximum all-pairs dot-product (mad) search," in *Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM)*, 2015, pp. 11–20.

[42] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proceedings of the 23rd international conference on Machine learning - ICML '06*. New York, New York, USA: ACM Press, 2006, pp. 97–104.

[43] R. Krauthgamer and J. R. Lee, "Navigating nets: simple algorithms for proximity search," in *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, 2004, pp. 798–807.

[44] https://github.com/Haposola/LRUS-CoverTree-MIPS.

[45] https://github.com/Jacyhust/FARGO.

[46] https://github.com/jerry-liujie/ip-nsw/tree/GraphMIPS.

[47] https://github.com/ThrunGroup/napg.

[48] https://github.com/google-research/google-research/tree/master/scann.

[49] R. R. Curtin, M. Edel, O. Shrit, S. Agrawal, S. Basak, J. J. Balamuta, R. Birmingham, K. Dutt, D. Eddelbuettel, R. Garg, S. Jaiswal, A. Kaushik, S. Kim, A. Mukherjee, N. G. Sai, N. Sharma, Y. S. Parihar, R. Swain, and C. Sanderson, "mlpack 4: a fast, header-only C++ machine learning library," *J. Open Source Softw.*, vol. 8, no. 82, p. 5026, 2023. [Online]. Available: https://doi.org/10.21105/joss.05026

[50] https://www.cse.cuhk.edu.hk/systems/hash/gqr/datasets.html.

[51] https://ai.tencent.com/ailab/nlp/en/embedding.html.

[52] https://www.promptingguide.ai/techniques/rag.

[53] D. Cai, Y. Wang, L. Liu, and S. Shi, "Recent advances in retrieval-augmented text generation," in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022, pp. 3417–3419.