Noname manuscript No.
(will be inserted by the editor)

Index-based, High-dimensional, Cosine Threshold Querying with Optimality Guarantees

Yuliang Li · Jianguo Wang · Benjamin Pullman · Nuno Bandeira · Yannis Papakonstantinou

Received: date / Accepted: date

Abstract Given a database of vectors, a cosine threshold query returns all vectors in the database having cosine similarity to a query vector above a given threshold θ. These queries arise naturally in many applications, such as document retrieval, image search, and mass spectrometry. The paper considers the efficient evaluation of such queries, as well as of the closely related top-k cosine similarity queries. It provides novel optimality guarantees that exhibit good performance on real datasets.We take as a starting point Fagin's well-known Threshold Algorithm (TA), which can be used to answer cosine threshold queries as follows: an inverted index is first built from the database vectors during pre-processing; at query time, the algorithm traverses the index partially to gather a set of candidate vectors to be later verified for θ-similarity. However, directly applying TA in its raw form misses significant optimization opportunities. Indeed, we

Yuliang Li
Megagon Labs
444 Castro Street Suite 720, Mountain View, CA 94041, United States
E-mail: yuliang@megagon.ai

Jianguo Wang
Department of Computer Science
Purdue University
610 Purdue Mall, West Lafayette, IN 47907, United States
E-mail: csjgwang@purdue.edu

Benjamin Pullman
UC San Diego
9500 Gilman Drive, San Diego, CA 92093, United States
E-mail: bpullman@cs.ucsd.edu

Nuno Bandeira
UC San Diego
9500 Gilman Drive, San Diego, CA 92093, United States
E-mail: bandeira@cs.ucsd.edu

Yannis Papakonstantinou
UC San Diego
9500 Gilman Drive, San Diego, CA 92093, United States
E-mail: yannis@cs.ucsd.edu

first show that one can take advantage of the fact that the vectors can be assumed to be normalized, to obtain an improved, tight stopping condition for index traversal and to efficiently compute it incrementally. Then we show that multiple real-world data sets from mass spectrometry, natural language process, and computer vision exhibit a certain form of data skewness and we exploit this property to obtain better traversal strategies. We show that under the skewness assumption, the new traversal strategy has a strong, near-optimal performance guarantee. The techniques developed in the paper are quite general since they can be applied to a large class of similarity functions beyond cosine.

1 Introduction

Cosine Similarity Search (CSS) [72,4,54] is a broad area where querying/search in vector databases is based on the cosine of two vectors. The cosine similarity search problem arises naturally in many applications including document retrieval [14], image search [48], recommender systems [54] and mass spectrometry. This work was motivated by and was applied in mass spectrometry, where billions of spectra are generated for the purpose of protein analysis [1,49,78]. Each spectrum is a collection of key-value pairs where the key is the mass-to-charge ratio of an ion contained in the protein and the value is the intensity of the ion. Essentially, each spectrum is a high-dimensional, non-negative and sparse vector with $\sim$2000 dimensions where $\sim$100 coordinates are non-zero.

There are two main variants of CSS in the various applications and the literature [72,73,4]: *cosine threshold queries* and *cosine top-k queries*. This work primarily focuses on processing cosine threshold queries and expands the cosine threshold queries techniques to cosine top-k queries.

Given a database of vectors, a cosine threshold query asks for all database vectors with cosine similarity to a query vector above a given threshold. Cosine threshold queries play an important role in analyzing spectra repositories. Example questions include "is the given spectrum similar to any spectrum in the database?", spectrum identification (matching query spectra against reference spectra), or clustering (matching pairs of unidentified spectra) or metadata queries (searching for public datasets containing matching spectra, even if obtained from different types of samples). For such applications with a large vector database, it is critically important to process cosine threshold queries efficiently – this is the fundamental topic addressed in this paper.

Definition 1 (Cosine Threshold Query) Let $\mathcal{D}$ be a collection of high-dimensional, non-negative vectors; $\mathbf{q}$ be a query vector; θ be a threshold $0 < \theta \leq 1$. Then the cosine threshold query returns the vector set $\mathcal{R} = \{\mathbf{s} | \mathbf{s} \in \mathcal{D}, \texttt{cos}(\mathbf{q}, \mathbf{s}) \geq \theta\}$. A vector $\mathbf{s}$ is called *θ-similar* to the query $\mathbf{q}$ if $\texttt{cos}(\mathbf{q}, \mathbf{s}) \geq \theta$ and the *score* of $\mathbf{s}$ is the value $\texttt{cos}(\mathbf{q}, \mathbf{s})$ when $\mathbf{q}$ is understood from the context.

The top-k version can be defined similarly:

Definition 2 (Cosine Top-k Query) Let $\mathcal{D}$ be a database of vectors; $\mathbf{q}$ be a query vector; k be a positive integer. The cosine top-k query returns a subset of $\mathcal{D}$ with the k highest cosine similarity with $\mathbf{q}$.

Observe that cosine similarity is insensitive to vector normalization. We will therefore assume without loss of generality that the database as well as query consist of unit vectors (otherwise, all vectors can be normalized in a pre-processing step).

Because of the unit-vector assumption, the scoring function $\texttt{cos}$ computes the dot product $\mathbf{q} \cdot \mathbf{s}$. Without the unit-vector assumption, cosine threshold querying is equivalent to *inner product threshold querying*, which is of interest in its own right. We summarize related work on cosine and inner product similarity search in Section 7.

In this paper, we develop novel techniques for the efficient evaluation of cosine threshold queries and expand the techniques to cosine top-k queries. We take as a starting point the well-known Threshold Algorithm (TA), by Fagin et al. [30], because of its simplicity, wide applicability, and optimality guarantees. We begin with a brief review of the TA algorithm.

Review of TA. On a database $\mathcal{D}$ of d-dimensional vectors $\{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$, given a query monotonic scoring function $F : \mathbb{R}^d \mapsto \mathbb{R}$ and a query parameter k, the Threshold Algorithm (TA) computes the k database vectors with the highest score $F(\mathbf{s})$. A scoring function F is monotonic if $F(\mathbf{s})$ is a non-decreasing function wrt each dimension of $\mathbf{s}$ (or non-increasing, we assume non-decreasing wlog). A wide range of commonly-used functions for scoring vectors are monotonic, including the weighted sum (or average) over $\mathbf{s}$, the $\min$ (or $\max$), and the median value of $\mathbf{s}$.

The TA works as follows. First, TA preprocesses the vector database by building an inverted index $\{L_i\}_{1\le i\le d}$ where each L_i is an inverted list that contains pairs of $(\mathsf{ref}(\mathbf{s}), \mathbf{s}[i])$ where $\mathsf{ref}(\mathbf{s})$ is a reference to the vector $\mathbf{s}$ and $\mathbf{s}[i]$ is the i-th dimension $\mathbf{s}$. Each L_i is sorted in descending order of $\mathbf{s}[i]$. When a query (F, k) arrives, TA proceeds as follows. It maintains a pointer b starting from 1 to all the inverted lists and increments b iteratively. At each iteration:

- Collect the set $\mathcal{C}$ of candidates of all references in L_i up to position b for all dimension i. Namely, $\mathcal{C} = \bigcup_{i=1}^{d}\{\mathsf{ref}(\mathbf{s}) | (\mathsf{ref}(\mathbf{s}), \mathbf{s}[i]) \in L_i[1, \ldots, b]\}$.
- Compute F_k the k-th highest score $F(\mathbf{s})$ for all $\mathsf{ref}(\mathbf{s}) \in \mathcal{C}$ by accessing $\mathbf{s}$ in the database with the reference.
- If the score F_k is no less than $F(L_1[b], \ldots, L_d[b])$, return the top-k highest score vectors in $\mathcal{C}$; otherwise continue to the next iteration with $b \leftarrow b + 1$.

By monotonicity of the function F, once the stopping condition is satisfied, it is guaranteed that no vector $\mathbf{s}$ below the pointer b can have $F(\mathbf{s})$ above the current k-th highest score. Thus the candidate set $\mathcal{C}$ contains the complete set of all the k highest score vectors in the database.

One nice property of TA is that it guarantees *instance optimality*. Informally, for any given database $\mathcal{D}$ and query $(\mathbf{q}, k)$, TA performs no worse than ANY algorithm that requires sequential access to the inverted index structure up to a multiplicative factor of d in terms of the number of data accesses.

Theorem 1 (Instance Optimality of TA **[30,31], Informal)** *Suppose that* OPT *is the minimal number of accesses to the inverted lists to answer the query* $(\mathbf{q}, k)$*, the number of accesses performed by* TA *is at most* $d \cdot \mathsf{OPT}$.

A TA**-like baseline index and algorithm and its shortcomings.** The TA algorithm can be easily adapted to our setting, yielding a first-cut approach to processing cosine threshold queries. We describe how this is done and refer to the resulting index and algorithm as the *TA-like baseline*. Note first that cosine threshold queries use $\mathsf{cos}(\mathbf{q}, \mathbf{s})$, which can be viewed as a particular family of functions $F(\mathbf{s}) = \mathbf{s} \cdot \mathbf{q}$ parameterized by $\mathbf{q}$, that are monotonic in $\mathbf{s}$ for unit vectors. However, TA produces the vectors with the top-k scores according to $F(\mathbf{s})$, whereas cosine threshold queries return all $\mathbf{s}$ whose score exceeds the threshold θ. We will show how this difference can be overcome straightforwardly.

We construct the baseline index and algorithm for answering cosine threshold queries as follows. Note that the algorithm is *exact*, which means that it returns all vectors no less than the threshold θ.

Identically to the TA, the baseline index consists of one sorted list for each of the d dimensions. In particular, the i-th sorted list has pairs $(\mathsf{ref}(\mathbf{s}), \mathbf{s}[i])$, where $\mathsf{ref}(\mathbf{s})$ is a reference to the vector $\mathbf{s}$ and $\mathbf{s}[i]$ is its value on the i-th dimension. The list is sorted in descending order of $\mathbf{s}[i]$.[1]

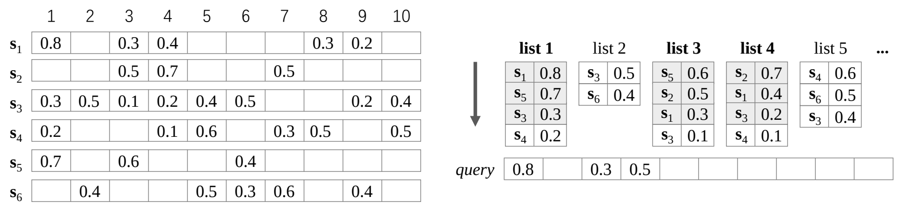
Next, the baseline, like the TA, proceeds into a *gathering phase* during which it collects a complete set of references to candidate result vectors. The TA shows that gathering can be achieved by reading the d sorted lists from top to bottom and terminating early when a *stopping condition* is finally satisfied. The condition guarantees that any vector that has not been seen yet has no chance of being in the query result. The baseline makes a straightforward change to the TA's stopping condition to adjust for the difference between the TA's top-k requirement and the threshold requirement of the cosine threshold queries. In particular, in each round the baseline algorithm has read the first b entries of each index. (Initially it is $b = 1$.) If it is the case that $\mathsf{cos}(\mathbf{q}, [L_1[b], \ldots, L_d[b]]) < \theta$ then it is guaranteed that the algorithm has already read (the references to) all the possible candidates and thus it is safe to terminate the gathering phase, see Figure 1 for an example. Every vector $\mathbf{s}$ that appears in the j-th entry of a list for $j \leq b$ is a candidate.

In the next phase, called the *verification* phase, the baseline algorithm (again like TA) retrieves the candidate vectors from the database and checks which ones actually score above the threshold.

For inner product queries, the baseline algorithm's gathering phase benefits from the same $d \cdot \mathsf{OPT}$ instance optimality guarantee as the TA. Namely, the gathering phase will access at most $d \cdot \mathsf{OPT}$ entries, where OPT is the optimal index access cost. More specifically, the notion of OPT is the *minimal number of sequential accesses* of the sorted inverted index during the gathering phase for any TA-like algorithm applied to the specific query and index instance.

There is an obvious optimization: Only the m dimensions that have non-zero values in the query vector $\mathbf{q}$ should participate in query processing – this leads to a $m \cdot \mathsf{OPT}$

[1] There is no need to include pairs with zero values in the list.



	1	2	3	4	5	6	7	8	9	10
s_1	0.8		0.3	0.4				0.3	0.2	
s_2			0.5	0.7			0.5			
s_3	0.3	0.5	0.1	0.2	0.4	0.5			0.2	0.4
s_4	0.2			0.1	0.6		0.3	0.5		0.5
s_5	0.7		0.6			0.4				
s_6		0.4			0.5	0.3	0.6		0.4	

list 1		list 2		**list 3**		**list 4**		list 5		...
s_1	0.8	s_3	0.5	s_5	0.6	s_2	0.7	s_4	0.6	
s_5	0.7	s_6	0.4	s_2	0.5	s_1	0.4	s_6	0.5	
s_3	0.3			s_1	0.3	s_3	0.2	s_3	0.4	
s_4	0.2			s_3	0.1	s_4	0.1			

query										
	0.8		0.3	0.5						

Fig. 1 An example of cosine threshold query with six 10-dimensional vectors. The missing values are 0's. We only need to scan the lists L_1, L_3, and L_4 since the query vector has non-zero values in dimension 1, 3 and 4. For $\theta = 0.6$, the gathering phase terminates after each list has examined three entries (highlighted) because the score for any unseen vector is at most $0.8 \times 0.3 + 0.3 \times 0.3 + 0.5 \times 0.2 = 0.43 < 0.6$. The verification phase only needs to retrieve from the database those vectors obtained during the gathering phase, i.e., $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$ and $\mathbf{s}_5$, compute the cosines and produce the final result.

guarantee for inner product queries[2]. But even this guarantee loses its practical value when m is a large number. In the mass spectrometry scenario m is $\sim$100. In document similarity and image similarity cases it is even higher.

For cosine threshold queries, the $m \cdot \mathsf{OPT}$ guarantee no longer holds. The baseline fails to utilize the unit vector constraint to reach the stopping condition faster, resulting in an unbounded gap from OPT because of the unnecessary accesses (see Section 3.1).[3] Furthermore, the baseline fails to utilize the skewing of the values in the vector's coordinates (both of the database's vectors and of the query vector) and the linearity of the similarity function. Intuitively, if the query's weight is concentrated on a few coordinates, the query processing should overweight the respective lists and may, thus, reach the stopping condition much faster than reading all relevant lists in tandem.

We retain the baseline's index and the gathering-verification structure which characterizes the family of TA-like algorithms. The decision to keep the gathering and verification stages separate is discussed in Section 2. We argue that this algorithmic structure is appropriate for cosine threshold queries, because further optimizations that would require merging the two phases are only likely to yield marginal benefits. Within this framework, we reconsider

1. *Traversal strategy optimization*: A *traversal strategy* determines the order in which the gathering phase proceeds in the lists. In particular, we allow the gathering phase to move deeper in some lists and less deep in others. For example, the gathering phase may have read at some point $b_1 = 106$ entries from the first list, $b_2 = 523$ entries from the second list, etc. Multiple traversal strategies are possible and, generally, each traversal strategy will reach the stopping condition with a different configuration of $[b_1, b_2, \ldots, b_n]$. The traversal strategy optimization problem asks that we efficiently identify a traversal path that minimizes the access cost $\sum_{i=1}^{d} b_i$. To enable such optimization, we will allow adding a lightweight auxiliary data structured pre-computed from the baseline index.

[2] This optimization is equally applicable to the TA's problem: Scan only the lists that correspond to dimensions that actually affect the function F.

[3] Notice, the unit vector constraint enables inference about the collective weight of the unseen coordinates of a vector.

Table 1 Summary of theoretical results for the near-convex case.

	Stopping Condition		**Traversal Strategy**	
	Baseline	*This work*	*Baseline*	*This work*
Inner Product	Tight		$m \cdot \mathsf{OPT}$	$\mathsf{OPT} + c$
Cosine	Not tight	Tight	NA	$\mathsf{OPT}(\theta - \epsilon) + c$

2. *Stopping condition optimization*: We reconsider the stopping condition so that it takes into account (a) the specifics of the cos function and (b) the unit vector constraint. Moreover, since the stopping condition is tested frequently during the gathering phase, it has to be evaluated very efficiently. Notice that optimizing the stopping condition is independent of the traversal strategy or skewness assumptions about the data.

Contributions and summary of results.

– We present a stopping condition for early termination of the index traversal (Section 3). We show that the stopping condition is *complete* and *tight*, informally meaning that (1) for any traversal strategy, the gathering phase will produce a candidate set containing all the vectors θ-similar to the query, and (2) the gathering terminates as soon as no more θ-similar vectors can be found (Theorem 3). In contrast, the stopping condition of the (TA-inspired) baseline is complete but not tight (Theorem 2). The proposed stopping condition takes into account that all database vectors are normalized and reduces the problem to solving a special quadratic program (Equation 2) that guarantees both completeness and tightness.

– We introduce a *hull-based* traversal strategy exploiting a common skewness property (Section 4). In particular, this skewness property requires that each sorted list L_i is "mostly convex", meaning that the shape of L_i is approximately the *lower convex hull* constructed from the set of points of L_i. This technique is quite general, as it can be extended to the class of *decomposable functions* which have the form $F(\mathbf{s}) = f_1(\mathbf{s}[1]) + \ldots + f_d(\mathbf{s}[d])$ where each f_i is non-decreasing.[4] Consequently, we provide the following optimality guarantee for inner product threshold queries: The number of accesses executed by the gathering phase (i.e., $\sum_{i=1}^{d} b_i$) is at most $\mathsf{OPT} + c$ (Theorem 6 and Corollary 1), where OPT is the number of accesses by the optimal strategy and c is the maximal number of points from each L_i in between two vertices of the lower convex hulls. Experiments show that in the real-world cases of image search, document search, and mass spectrometry, c is a very small fraction of OPT (only 1.3%, 7.9%, and 0.4% in the 3 datasets respectively).

– Despite the fact that cosine and its tight stopping condition are not decomposable, we show that the hull-based strategy can be adapted to cosine threshold queries by approximating the tight stopping condition with a carefully chosen decomposable function. We show that when the approximation is at most ϵ-away from the actual value, the access cost is at most $\mathsf{OPT}(\theta - \epsilon) + c$ (Theorem 7), where $\mathsf{OPT}(\theta - \epsilon)$ is the optimal access cost on the same query $\mathbf{q}$ with the threshold lowered by ϵ and c is a constant similar to the above decomposable cases. Experiments show that the

[4] The inner product threshold problem is the special case where $f_i(\mathbf{s}[i]) = q_i \cdot \mathbf{s}[i]$.

adjustment ϵ is very small in practice, e.g., 0.1. We summarize these new results in Table 1.

This work is the extended version of [56]. In addition to the contributions above and already presented in [56], this work makes the following additional contributions:

- We provide complete proofs of several main theorems (Theorem 2 on why the baseline stopping condition is not tight, Theorem 5 and 6 on the (near-)optimality of the proposed traversal strategies).
- We provide the details of the algorithm (Algorithm 2) for incrementally computing the tight stopping condition and analyze its running time. While a direct testing of the tight and complete stopping condition takes at least $\Omega(d)$ time each round, the incremental maintenance algorithm leverages a balanced binary search tree to reduce the cost per round to $\mathcal{O}(\log(d))$ where d is the vector dimension.
- We review the partial verification techniques for checking θ-similarity in Section 5. Partial verification avoids a full scan of each candidate vector by inspecting only the dominating dimensions of the vector. As a novel contribution, we show that partial verification has near-constant performance guarantee under a skewness assumption. We also verify this assumption in a practical setting. This result makes the hull-based strategies more attractive as optimizing the gathering phase becomes the dominating factor.
- Finally, in Section 6, we discuss generalization of the proposed stopping condition and traversal strategies to top-k cosine queries. We show how the proposed stopping condition for threshold queries can be applied to top-k queries. We also showed that the hull-based traversal strategies are applicable for inner product queries but need a number of adjustments for cosine queries.

The paper is organized as follows. We introduce the algorithmic framework and basic definitions in Section 2. Section 3 and 4 discuss the technical developments on optimizing the stopping conditions and traversal strategies. Section 5 and 6 provide additional details related to verification phase optimization and the generalization of the proposed techniques to top-k queries. Finally, we discuss related work in Section 7 and conclude in Section 8.

2 Algorithmic Framework

In this section, we present a Gathering-Verification algorithmic framework to facilitate optimizations in different components of an algorithm with a TA-like structure. We start with notations summarized in Table 2.

To support fast query processing, we build an index for the database vectors similar to the original TA. The basic index structure consists of a set of 1-dimensional sorted lists (a.k.a inverted lists in web search [14]) where each list corresponds to a vector dimension and contains vectors having non-zero values on that dimension, as mentioned earlier in Section 1. Formally, for each dimension i, L_i is a list of pairs $\{(\mathsf{ref}(\mathrm{s}), \mathrm{s}[i]) \mid \mathrm{s} \in \mathcal{D} \wedge \mathrm{s}[i] > 0\}$ where $\mathsf{ref}(\mathrm{s})$ is a reference to the vector s and $\mathrm{s}[i]$ is its value on the i-th dimension. In the interest of brevity, we will often write $(\mathrm{s}, \mathrm{s}[i])$ instead of $(\mathsf{ref}(\mathrm{s}), \mathrm{s}[i])$. As an example in Figure 1, the list L_1 is built for the first

Table 2 Notation

$\mathcal{D}$	the vector database	$\|\|\mathbf{s}\|\|$	the L2 norm of $\mathbf{s}$
N	the number of vectors in $\mathcal{D}$	θ	the similarity threshold
d	the number of dimensions	$\mathtt{cos}(\mathbf{p}, \mathbf{q})$	the cosine of vectors $\mathbf{p}$ and $\mathbf{q}$
$\mathbf{s}$ (bold font)	a data vector	L_i	the inverted list of the i-th dimension
$\mathbf{q}$ (bold font)	a query vector	$\mathbf{b} = (b_1, \ldots, b_d)$	a position vector
$\mathbf{s}[i]$ or s_i	the i-th dimensional value of $\mathbf{s}$	$L_i[b_i]$	the b_i-th value of L_i
$\|\mathbf{s}\|$	the L1 norm of $\mathbf{s}$	$L[\mathbf{b}]$	the vector $(L_1[b_1], \ldots, L_d[b_d])$

dimension and it includes 4 entries: $(\mathbf{s}_1, 0.8)$, $(\mathbf{s}_5, 0.7)$, $(\mathbf{s}_3, 0.3)$, $(\mathbf{s}_4, 0.2)$ because $\mathbf{s}_1$, $\mathbf{s}_5$, $\mathbf{s}_3$ and $\mathbf{s}_4$ have non-zero values on the first dimension. We denote by $L_i[j]$ for $j \geq 1$ the j-th pair in the descending order of L_i sorted by the values of $\mathbf{s}[i]$. For clarity of presentation, we assume a special entry $L_i[0] = (\mathsf{null}, 1.0)$ which corresponds to a position outside of the inverted list and does not refer to any vector in the database. When the context is clear, we use $L_i[j]$ to denote only the value part stored in the entry.

Next, we show the Gathering-Verification framework (Algorithm 1) that operates on the index structure. The framework includes two phases: the gathering phase and the verification phase.

Algorithm 1: Gathering-Verification Framework

```
  input  : ($\mathcal{D}$, $\{L_i\}_{1 \leq i \leq d}$, $\mathbf{q}$, $\theta$)
  output : $\mathcal{R}$ the set of $\theta$-similar vectors
  /* Gathering phase                                                    */
1 Initialize $\mathbf{b} = (b_1, \ldots, b_d) = (0, \ldots, 0)$;
  // $\varphi(\cdot)$ is the stopping condition
2 while $\varphi(\mathbf{b}) = \mathtt{false}$ do
      // $\mathcal{T}(\cdot)$ is the traversal strategy to determine which list to access
         next
3     $i \leftarrow \mathcal{T}(\mathbf{b})$;
4     $b_i \leftarrow b_i + 1$;
5     Put the vector $\mathbf{s}$ in $L_i[b_i]$ to the candidate pool $\mathcal{C}$;
  /* Verification phase                                                 */
6 $\mathcal{R} \leftarrow \{\mathbf{s} | \mathbf{s} \in \mathcal{C} \wedge \mathtt{cos}(\mathbf{q}, \mathbf{s}) \geq \theta\}$;
7 return $\mathcal{R}$;
```

Gathering phase (line 1 to line 5). The goal of the gathering phase is to collect a complete set of candidate vectors while minimizing the number of accesses to the sorted lists. The algorithm maintains a *position vector* $\mathbf{b} = (b_1, \ldots, b_d)$ where each b_i indicates the current position in the inverted list L_i. Initially, the position vector $\mathbf{b}$ is $(0, \ldots, 0)$. Then it traverses the lists according to a *traversal strategy* that determines the list (say L_i) to be accessed next (line 3). Then it advances the pointer b_i by 1 (line 4) and adds the vector $\mathbf{s}$ referenced in the entry $L_i[b_i]$ to a candidate pool $\mathcal{C}$ (line 5). The traversal strategy is usually stateful, which means that its decision is made based on information that has been observed up to position $\mathbf{b}$ and its past decisions. For example, a strategy may decide that it will make the next 20 moves along dimension 6 and thus it needs state in order to remember that it has already committed to 20 moves on dimension 6.

The gathering phase terminates once a *stopping condition* is met. Intuitively, based on the information that has been observed in the index, the stopping condition checks if a complete set of candidates has already been found.

Next, we formally define stopping conditions and traversal strategies. As mentioned above, the input of the stopping condition and the traversal strategy is the information that has been observed up to position $\mathbf{b}$, which is formally defined as follows.

Definition 3 Let $\mathbf{b}$ be a position vector on the inverted index $\{L_i\}_{1\leq i\leq d}$ of a database $\mathcal{D}$. The partial observation at $\mathbf{b}$, denoted as $\mathcal{L}(\mathbf{b})$, is a collection of lists $\{\hat{L}_i\}_{1\leq i\leq d}$ where for every $1 \leq i \leq d$, $\hat{L}_i = [L_i[1], \ldots, L_i[b_i]]$.

Definition 4 Let $\mathcal{L}(\mathbf{b})$ be a partial observation and $\mathbf{q}$ be a query with similarity threshold θ. A **stopping condition** is a boolean function $\varphi(\mathcal{L}(\mathbf{b}), \mathbf{q}, \theta)$ and a **traversal strategy** is a function $\mathcal{T}(\mathcal{L}(\mathbf{b}), \mathbf{q}, \theta)$ whose domain is $[d]$[5]. When clear from the context, we denote them simply by $\varphi(\mathbf{b})$ and $\mathcal{T}(\mathbf{b})$ respectively.

Verification phase (line 6). The verification phase examines each candidate vector $\mathbf{s}$ seen in the gathering phase to verify whether $\texttt{cos}(\mathbf{q}, \mathbf{s}) \geq \theta$ by accessing the database. Various techniques [72,5,54] have been proposed to speed up this process. Essentially, instead of accessing all the d dimensions of each $\mathbf{s}$ and $\mathbf{q}$ to compute exactly the cosine similarity, these techniques decide θ-similarity by performing a partial scan of each candidate vector. We review these techniques, which we refer to as *partial verification*, in Section 5. Additionally, as a novel contribution, we show that in the presence of data skewness, partial verification has a near-constant performance guarantee (Theorem 8) for verifying each candidate.

Remark on optimizing the gathering phase. Due to these optimization techniques, the *number of sequential accesses* performed during the gathering phase becomes the dominating factor of the overall running time. The reason behind this is that the number of sequential accesses is strictly greater than the number of candidates that need to be verified so reducing the sequential access cost also results in better performance of the verification phase. In practice, we observed that the sequential cost is indeed dominating: for 1,000 queries on 1.2 billion vectors with similarity threshold 0.6, the sequential gathering time is 16 seconds and the verification time is only 4.6 seconds. Such observation justifies our goal of designing a traversal strategy with near-optimal sequential access cost, as the dominant cost concerns the gathering stage.

Computation models. We consider optimizing the sequential access cost of algorithms that falls into the Gathering-Verification framework. We assume that during the Gathering phase, the algorithm can only sequentially access the inverted lists $\{L_i\}_{1\leq i\leq d}$ and the verification phase can randomly access the whole database $\mathcal{D}$. The stopping condition $\varphi(\cdot)$ and the traversal strategy $\mathcal{T}(\cdot)$ are of the standard Random-Access Machine (RAM) model ([67], Chapter 3). At a position vector $\mathbf{b}$, these two functions have access to (1) the entries of $\{L_i\}_{1\leq i\leq d}$ up to $\mathbf{b}$ (i.e., entries that have

[5] $[d]$ is the set $\{1, \ldots, d\}$

been accessed) and (2) stateful data structures internal to φ and $\mathcal{T}$. In addition, we allow the algorithms to preprocess the inverted lists $\{L_i\}_{1\le i\le d}$ into a query-independent auxiliary data structure which can be accessed by the traversal strategy $\mathcal{T}$. We use the standard big-O notation for measuring the time/space complexity of φ and $\mathcal{T}$.

3 Stopping condition

In this section, we introduce a fine-tuned stopping condition that satisfies the tight and complete requirements to early terminate the index traversal.

First, the stopping condition has to guarantee *completeness* (Definition 5), i.e. when the stopping condition φ holds on a position $\mathbf{b}$, the candidate set $\mathcal{C}$ must contain all the true results. Note that since the input of φ is the partial observation at $\mathbf{b}$, we must guarantee that for all possible databases $\mathcal{D}$ consistent with the partial observation $\mathcal{L}(\mathbf{b})$, the candidate set $\mathcal{C}$ contains all vectors in $\mathcal{D}$ that are θ-similar to the query $\mathbf{q}$. This is equivalent to require that if a unit vector $\mathbf{s}$ is found below position $\mathbf{b}$ (i.e. $\mathbf{s}$ does not appear above $\mathbf{b}$), then $\mathbf{s}$ is NOT θ-similar to $\mathbf{q}$. We formulate this as follows.

Definition 5 (Completeness) Given a query $\mathbf{q}$ with threshold θ, a position vector $\mathbf{b}$ on index $\{L_i\}_{1\le i\le d}$ is complete iff for every unit vector $\mathbf{s}$, $\mathbf{s} < L[\mathbf{b}]$ implies $\mathbf{s}\cdot\mathbf{q} < \theta$. A stopping condition $\varphi(\cdot)$ is complete iff for every $\mathbf{b}$, $\varphi(\mathbf{b}) = \texttt{True}$ implies that $\mathbf{b}$ is complete.

The second requirement of the stopping condition is *tightness*. It is desirable that the algorithm terminates immediately once the candidate set $\mathcal{C}$ contains a complete set of candidates, such that no additional unnecessary access is made. This can reduce not only the number of index accesses but also the candidate set size, which in turn reduces the verification cost. Formally,

Definition 6 (Tightness) A stopping condition $\varphi(\cdot)$ is tight iff for every complete position vector $\mathbf{b}$, $\varphi(\mathbf{b}) = \texttt{True}$.

3.1 The baseline stopping condition is not tight

It is desirable that a stopping condition is both complete and tight. However, as we will show next, the baseline stopping condition $\varphi_{\mathsf{BL}} = \big(\mathbf{q}\cdot L[\mathbf{b}] < \theta\big)$ is complete but not tight as it does not capture the unit vector constraint to terminate as soon as no unseen *unit vector* can satisfy $\mathbf{s}\cdot\mathbf{q} \ge \theta$.

Theorem 2 *The* baseline *stopping condition*

$$\varphi_{\mathsf{BL}}(\mathbf{b}) = \left(\sum_{i=1}^{d} q_i \cdot L_i[b_i] < \theta\right) \tag{1}$$

is complete but not tight.

Proof For every position vector $\mathbf{b}$, $\varphi_{\mathsf{BL}}(\mathbf{b}) = \mathtt{True}$ implies $\mathbf{q} \cdot L[\mathbf{b}] < \theta$. So for every $\mathbf{s} < L[\mathbf{b}]$, we also have $\mathbf{q} \cdot \mathbf{s} < \theta$ so φ_{BL} is complete.

To show the non-tightness, it is sufficient to show that for some position vector $\mathbf{b}$ where $\mathbf{b}$ is complete, $\varphi_{\mathsf{BL}}(\mathbf{b})$ is $\mathtt{False}$ so the traversal continues.

We illustrate a counterexample in Figure 2 with two dimensions (i.e., $d = 2$). Given a query $\mathbf{q}$, all possible θ-similar vectors form a hyper-surface defining the set $\mathsf{ans} = \{\mathbf{s} |\ \|\mathbf{s}\| = 1, \sum_{i=1}^{d} q_i \cdot s_i \geq \theta\}$. In Figure 2, $\|\mathbf{s}\| = 1$ is the circular surface and $\sum_{i=1}^{d} q_i \cdot s_i \geq \theta$ is a half-plane so the set of points ans is the arc $\widehat{AB}$.

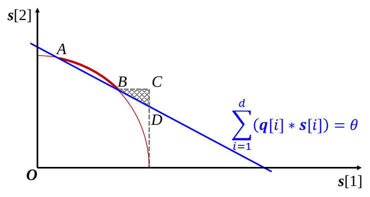


Fig. 2 A 2-d example of φ_{BL}'s non-tightness

By definition, a position vector $\mathbf{b}$ is complete if the set $\{\mathbf{s}|\mathbf{s} < L[\mathbf{b}]\}$ contains no point in ans. A position vector $\mathbf{b}$ satisfies φ_{BL} iff the point $L[\mathbf{b}]$ is above the hyper-plane $\sum_{i=1}^{d} q_i \cdot s_i = \theta$. It is clear from Figure 2 that if the point $L[\mathbf{b}]$ locates at the region BCD, then $\{\mathbf{s}|\mathbf{s} < L[\mathbf{b}]\}$ contains no point in $\widehat{AB}$ and is above the half-plane $\sum_{i=1}^{d} q_i \cdot s_i = \theta$. There exists a database of 2-d vectors such that $L[\mathbf{b}]$ resides in the BCD region for some position $\mathbf{b}$, so the stopping condition φ_{BL} is not tight. $\square$

Example. The example in Fig. 1 also illustrates why the baseline condition φ_{BL} is not tight. Assume that we use the baseline lockstep traversal strategy. The condition φ_{BL} is satisfied at $b = 3$ (i.e., $b_1 = b_3 = b_4 = 3$) where $\mathbf{q} \cdot L[\mathbf{b}] = 0.43 < \theta$ (recall that $\theta = 0.8$). At $b = 2$, since $\mathbf{q} \cdot L[\mathbf{b}] = 0.7 * 0.8 + 0.5 * 0.3 + 0.4 * 0.5 = 0.91 > \theta$, φ_{BL} is not satisfied. However, since $L[\mathbf{b}]^2 = 0.7^2 + 0.5^2 + 0.4^2 = 0.9 < 1$, we can be sure that all unit vectors have been scanned. Thus, the traversal can stop at $b = 2$ instead of 3 confirming that φ_{BL} is not tight.

Remark. The baseline stopping condition φ_{BL} is not tight because it does not take into account that all vectors in the database are unit vectors. In fact, one can show that φ_{BL} is tight and complete for inner product queries where the unit vector assumption is lifted. In addition, since φ_{BL} is not tight, any traversal strategy that works with φ_{BL} has no optimality guarantee in general since there can be a gap of arbitrary size between the stopping position by φ_{BL} and the one that is tight (i.e. there can be arbitrarily many points in the region BCD).

Next, we present a new stopping condition that is both complete and tight. To guarantee tightness, one can check at every snapshot during the traversal whether the current position vector $\mathbf{b}$ is complete and stop once the condition is true. However, directly testing the completeness is impractical since it is equivalent to testing whether there exists a real vector $\mathbf{s} = (s_1, \ldots, s_d)$ that satisfies the following following set of quadratic constraints:

$$(a)\ \sum_{i=1}^{d} s_i \cdot q_i \geq \theta, \quad (b)\ s_i \leq L_i[b_i],\ \forall\, i \in [d], \quad \text{and} \quad (c)\ \sum_{i=1}^{d} s_i^2 = 1. \tag{2}$$

We denote by $\mathbf{C}(\mathbf{b})$ (or simply $\mathbf{C}$) the set of $\mathbb{R}^d$ points defined by the above constraints. The set $\mathbf{C}(\mathbf{b})$ is infeasible (i.e. there is no satisfying $\mathbf{s}$) if and only if $\mathbf{b}$ is complete, but directly testing the feasibility of $\mathbf{C}(\mathbf{b})$ requires an expensive call to a quadratic programming solver. Depending on the implementation, the running time can be exponential or of high-degree polynomial [13]. We address this challenge by deriving an equivalently strong stopping condition that guarantees tightness and is efficiently testable:

Theorem 3 *Let τ be the solution of the equation $\sum_{i=1}^{d} \min\{q_i \cdot \tau, L_i[b_i]\}^2 = 1$ where $\|L[\mathbf{b}]\| > 1$ and*

$$\mathsf{MS}(L[\mathbf{b}]) = \sum_{i=1}^{d} \min\{q_i \cdot \tau, L_i[b_i]\} \cdot q_i \tag{3}$$

called the max-similarity. *The stopping condition $\varphi_{\mathsf{TC}}(\mathbf{b}) = (\mathsf{MS}(L[\mathbf{b}]) < \theta)$ is tight and complete.*

Remark. The solution τ to the equation $\sum_{i=1}^{d} \min\{q_i \cdot \tau, L_i[b_i]\}^2 = 1$ is unique when $\|L[\mathbf{b}]\| > 1$. This is because the LHS is a strictly increasing function with respect to τ and the range of the LSH is $(-\infty, \|L[\mathbf{b}]\|^2]$. When $\|L[\mathbf{b}]\| = 1$, any τ that satisfies $q_i \cdot \tau \geq L_i[b_i]$ for all i will satisfy the equation, resulting in $\min\{q_i \cdot \tau, L_i[b_i]\} = L_i[b_i]$ for all i and an unique value of $\mathsf{MS}(L[\mathbf{b}])$. If $\|L[\mathbf{b}]\| < 1$, then $\|\mathbf{s}\| < 1$ thus no unit vector $\mathbf{s}$ exists so the equation has no solution. However, $\|L[\mathbf{b}]\| \leq 1$ also implies that all unit vectors have been added to the candidate set $\mathcal{C}$ thus the traversal would have stopped. Thus, it is safe to assume that $\|L[\mathbf{b}]\| > 1$ and the solution τ is unique.

Proof The tight and complete stopping condition is obtained by applying the Karush-Kuhn-Tucker (KKT) conditions [47] for solving nonlinear programs. We first formulate the set of constraints in (2) as an optimization problem over $\mathbf{s}$:

$$\text{maximize} \sum_{i=1}^{d} s_i \cdot q_i \qquad \text{subject to} \sum_{i=1}^{d} s_i^2 = 1 \quad \text{and} \quad s_i \leq L_i[b_i], \quad \forall i \in [d] \tag{4}$$

So checking whether $\mathbf{C}$ is feasible is equivalent to verifying whether the maximal $\sum_{i=1}^{d} s_i \cdot q_i$ is at least θ. So it is sufficient to show that $\sum_{i=1}^{d} s_i \cdot q_i$ is maximized when $s_i = \min\{q_i \cdot \tau, L_i[b_i]\}$ as specified above.

The KKT conditions of the above maximization problem specify a set of necessary conditions that the optimal $\mathbf{s}$ needs to satisfy. More precisely, let

$$L(\mathbf{s}, \mu, \lambda) = \sum_{i=1}^{d} s_i q_i - \sum_{i=1}^{d} \mu_i (L_i[b_i] - s_i) - \lambda \left(\sum_{i=1}^{d} s_i^2 - 1 \right)$$

be the Lagrangian of (4) where $\lambda \in \mathbb{R}$ and $\mu \in \mathbb{R}^d$ are the Lagrange multipliers. Then,

Lemma 1 (derived from KKT) *The optimal* $\mathbf{s}$ *in (4) satisfies the following conditions:*

$$\begin{array}{ll} \nabla_{\mathbf{s}} L(\mathbf{s}, \mu, \lambda) = 0 & \textit{(Stationarity)} \\ \mu_i \geq 0,\ \forall\, i \in [d] & \textit{(Dual feasibility)} \\ \mu_i(L_i[b_i] - s_i) = 0,\ \forall\, i \in [d] & \textit{(Complementary slackness)} \end{array}$$

in addition to the constraints in (4) (called the Primal feasibility *conditions).*

By the Complementary slackness condition, for every i, if $\mu_i \neq 0$ then $s_i = L_i[b_i]$. If $\mu_i = 0$, then from the Stationarity condition, we know that for every i, $q_i + \mu_i - 2\lambda \cdot s_i = 0$ so $s_i = q_i/2\lambda$. Thus, the value of s_i is either $L_i[b_i]$ or $q_i/2\lambda$.

If $L_i[b_i] < q_i/2\lambda$ then since $s_i \leq L_i[b_i]$, the only possible case is $s_i = L_i[b_i]$. Otherwise, the objective function $\sum_{i=1}^{d} s_i \cdot q_i$ is maximized when each s_i is proportional to q_i, so $s_i = q_i/2\lambda$. Combining these two cases, we have $s_i = \min\{q_i/2\lambda, L_i[b_i]\}$.

Thus, for the λ that satisfies $\sum_{i=1}^{d} \min\{q_i/2\lambda, L_i[b_i]\}^2 = 1$, the objective function $\sum_{i=1}^{d} s_i \cdot q_i$ is maximized when $s_i = \min\{q_i/2\lambda, L_i[b_i]\}$ for every i. The theorem is obtained by letting $\tau = 1/2\lambda$. □

Remark of φ_{TC}. The tight stopping condition φ_{TC} computes the vector $\mathbf{s}$ below $L(\mathbf{b})$ with the maximum cosine similarity $\mathsf{MS}(L[\mathbf{b}])$ with the query $\mathbf{q}$. At the beginning of the gathering phase, $b_i = 0$ for every i so $\mathsf{MS}(L[\mathbf{b}]) = 1$ as $\mathbf{s}$ is not constrained. The cosine score is maximized when $\mathbf{s} = \mathbf{q}$ where $\tau = 1$. During the gathering phase, as b_i increases, the upper bound $L_i[b_i]$ of each s_i decreases. When $L_i[b_i] < q_i$ for some i, s_i can no longer be q_i. Instead, s_i equals $L_i[b_i]$, the rest of $\mathbf{s}$ increases proportional to $\mathbf{q}$ and τ increases. During the traversal, the value of τ monotonically increases and the score $\mathbf{s}(L[\mathbf{b}])$ monotonically decreases. This is because the space for $\mathbf{s}$ becomes more constrained by $L(\mathbf{b})$ as the pointers move deeper in the inverted lists.

3.2 Efficient computation of φ_{TC} with incremental maintenance

Testing the tight and complete condition φ_{TC} requires solving τ in Theorem (3), for which a direct application of the bisection method takes $\mathcal{O}(d)$ time. We show a novel efficient algorithm based on incremental maintenance which takes only $\mathcal{O}(\log d)$ time for each test of φ_{TC}.

Theorem 4 *The stopping condition* $\varphi_{\mathsf{TC}}(\mathbf{b})$ *can be incrementally computed in* $\mathcal{O}(\log d)$ *time.*

Proof According to the proof of Theorem 3, for an optimal solution $\mathbf{s}$ of (4) we know that

$$s_i = \begin{cases} L_i[b_i], & \tau \geq \dfrac{L_i[b_i]}{q_i}; \\ q_i \cdot \tau, & \text{otherwise.} \end{cases} \tag{5}$$

Wlog, suppose $\frac{L_1[b_1]}{q_1} \leq \cdots \leq \frac{L_d[b_d]}{q_d}$. Since $\sum_{i=1}^{d} s_i^2$ is monotonic increasing with respect to τ, to solve the τ such that $\sum_{i=1}^{d} s_i^2 = 1$, we simply need to find the range $[\frac{L_k[b_k]}{q_k}, \frac{L_{k+1}[b_{k+1}]}{q_{k+1}}]$ for the largest k such that

- $\sum_{i=1}^{d} s_i^2 \leq 1$ when $\tau = \frac{L_k[b_k]}{q_k}$ and
- $\sum_{i=1}^{d} s_i^2 > 1$ when $\tau = \frac{L_{k+1}[b_{k+1}]}{q_{k+1}}$.

Then the solution τ is in the range $[\frac{L_k[b_k]}{q_k}, \frac{L_{k+1}[b_{k+1}]}{q_{k+1}}]$. For such k, we have $s_i = L_i[b_i]$ for every $1 \leq i \leq k$ and $s_i = q_i \cdot \tau$ for $k < i \leq d$. Let $\mathsf{eval}(k, \tau)$ be the function

$$\mathsf{eval}(k, \tau) = \sum_{i=1}^{d} s_i^2 = \sum_{i=1}^{k} L_i[b_i]^2 + \sum_{i=k+1}^{d} q_i^2 \cdot \tau^2. \tag{6}$$

Then for the largest k such that $\mathsf{eval}(k, L_k[b_k]/q_k) \leq 1$, τ can be computed by solving

$$\sum_{i=1}^{k} L_i[b_i]^2 + \sum_{i=k+1}^{d} q_i^2 \cdot \tau^2 = 1 \Rightarrow \quad \tau = \left(\frac{1 - \sum_{i=1}^{k} L_i[b_i]^2}{1 - \sum_{i=1}^{k} q_i^2} \right)^{1/2}. \tag{7}$$

Then, $\mathsf{MS}(L[\mathbf{b}])$ can be computed as follows:

$$\mathsf{MS}(L[\mathbf{b}]) = \sum_{i=1}^{k} L_i[b_i] \cdot q_i + (1 - \sum_{i=1}^{k} q_i^2) \cdot \tau \ . \tag{8}$$

The equations above provide a simple algorithm for computing $\mathsf{MS}(L[\mathbf{b}])$: at each position $\mathbf{b}$, sort the $L_i[b_i]$'s by $L_i[b_i]/q_i$, find the largest k (such that $\mathsf{eval}(k, L_k[b_k]/q_k) \leq 1$), then compute τ and $\mathsf{MS}(L[\mathbf{b}])$ using Equation 7 and 8. However, such a simple algorithm requires $\Omega(d \log d)$ time for sorting, which is still too expensive as the stopping condition is checked in every step. Fortunately, we show that $\mathsf{MS}(L[\mathbf{b}])$ can be incrementally maintained in $\mathcal{O}(\log d)$ time as we describe below.

We use a binary search tree (BST) to maintain an order of the L_i's sorted by $L_i[b_i]/q_i$. The BST supports the following two operations:

- update(i): update $L_i[b_i] \to L_i[b_i + 1]$
- compute(): return the value of $\mathsf{MS}(L[\mathbf{b}])$

The compute() operation essentially performs the binary search of finding the largest k mentioned above. To ensure $\mathcal{O}(\log d)$ running time, we observe that from Equation (7) and (8), for any k, $\mathsf{MS}(L[\mathbf{b}])$ can be computed if $\sum_{i=1}^{k} L_i[b_i] \cdot q_i$, $\sum_{i=1}^{k} L_i[b_i]^2$, and $\sum_{i=1}^{k} q_i^2$ are available. Let T be the BST and each node in T is denoted as an integer n, meaning that the node represents the list L_{n}. We denote by subtree(n) the subtree of T rooted at node n, and keep track of the following values at each node n:

- n.key: $L_n[b_n]/q_n$,
- n.LQ: $\sum_{i \in \mathsf{subtree(n)}} L_i[b_i] \cdot q_i$,
- n.Q2: $\sum_{i \in \mathsf{subtree(n)}} q_i^2$ and
- n.L2: $\sum_{i \in \mathsf{subtree(n)}} L_i[b_i]^2$.

Thus, whenever there is a move on the list L_i, the key of the node i (i.e., $L_i[b_i]/q_i$) will be updated. Then we can remove the node i from the tree and insert it again using the new key, which takes $\mathcal{O}(\log d)$ time (with all the associated values being updated as well).

To compute $\mathsf{MS}(L[\mathbf{b}])$, we need to traverse a path in the BST to find the largest k while keeping track of the LQ, Q2, and L2 values from all nodes with less $L_i[b_i]/q_i$ value than the current node n. Such nodes can be either in: (1) the left subtree of n or (2) the left subtrees of all nodes n′ on the path to node n where $L_i[b_{n'}]/q_{n'} < L_i[b_n]/q_n$. (In other words, those are nodes where the traversal path makes a "right" turn.) We illustrate an example in Figure 3.

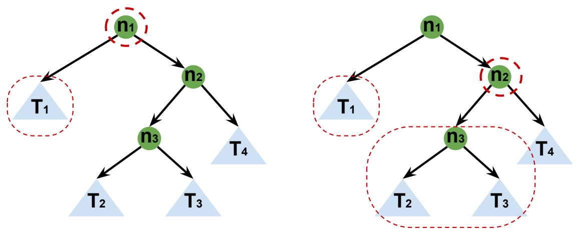


Fig. 3 Illustration of the BST traversal algorithm for computing $\mathsf{MS}(\cdot)$ incrementally. The tree traversal starts at the node n_1 and follows the path $\mathsf{n}_1 \to \mathsf{n}_2 \to \mathsf{n}_3$. At n_1, the score $\mathsf{MS}(L[\mathbf{b}])$ is computed using values stored in the subtree T_1. At n_2, the values used are from T_1 and the subtree rooted at n_3 (T_2 and T_3). At n_3, the values are from T_1 and T_2.

Algorithm 2: BST traversal algorithm for computing $\mathsf{MS}(\cdot)$

```
(LQ_parent, Q2_parent, L2_parent) ← $(0, 0, 0)$;
MS($L[\mathbf{b}]$) ← 1 ;                              // MS($L[\mathbf{b}]$) = 1 if $\forall i\ \tau \le L_i[b_i]/q_i$
n ← root($T$);
while n ≠ null do
    LQ ← LQ_parent + n.left.LQ + $L_n[b_n] \cdot q_n$;
    Q2 ← Q2_parent + n.left.Q2 + $q_n^2$;
    L2 ← L2_parent + n.left.L2 + $L_n[b_n]^2$;
    f(n) ← LQ + (1 − Q2) · n.key$^2$;        // Computing the RHS of Equation (6)
    if f(n) ≤ 1 then
        $\tau \leftarrow ((1 - \mathsf{L2})/(1 - \mathsf{Q2}))^{1/2}$;
        MS($L[\mathbf{b}]$) ← LQ + (1 − Q2) · $\tau$;
        n ← n.left ;                              // Left turn. No updates.
    else
        LQ_parent ← LQ_parent + n.left.LQ + $L_n[b_n] \cdot q_n$;
        Q2_parent ← Q2_parent + n.left.Q2 + $q_n^2$;
        L2_parent ← L2_parent + n.left.L2 + $L_n[b_n]^2$;
        n ← n.right ;     // Right turn. The accumulative values are updated.
return MS($L[\mathbf{b}]$);
```

Algorithm 2 shows the details of the traversal algorithm for computing $\mathsf{MS}(L[\mathbf{b}])$. For each node n, we denote by n.left (n.right) the left (right) child of n. We used the variables LQ_parent, Q2_parent, and L2_parent to accumulate the values when the right turns are made. The number of traversal steps is no more than the depth of the

BST and each step uses constant time. Thus, the overall cost of a single test of the stopping condition is $\mathcal{O}(\log d)$. □

4 Near-Optimal Traversal Strategy

Given the inverted lists index and a query, there can be many stopping positions that are both complete and tight. To optimize the performance, we need a traversal strategy that reaches one such position as fast as possible. Specifically, the goal is to design a traversal strategy $\mathcal{T}$ that minimizes $|\mathbf{b}| = \sum_{i=1}^{d} b_i$ where $\mathbf{b}$ is the first position vector satisfying the tight and complete stopping condition if $\mathcal{T}$ is followed. Minimizing $|\mathbf{b}|$ also reduces the number of collected candidates, which in turn reduces the cost of the verification phase. We call $|\mathbf{b}|$ the *access cost* of the strategy $\mathcal{T}$. Formally,

Definition 7 (Access Cost) Given a traversal strategy $\mathcal{T}$, we denote by $\{\mathbf{b}_i\}_{i\geq 0}$ the sequence of position vectors obtained by following $\mathcal{T}$. The access cost of $\mathcal{T}$, denoted by $\mathsf{cost}(\mathcal{T})$, is the minimal k such that $\varphi_{\mathsf{TC}}(\mathbf{b}_k) = \mathtt{True}$. Note that $\mathsf{cost}(\mathcal{T})$ also equals $|\mathbf{b}_k|$.

Definition 8 (Optimal Access Cost) Given a database $\mathcal{D}$ with inverted lists $\{L_i\}_{1\leq i\leq d}$, a query vector $\mathbf{q}$ and a threshold θ, the optimal access cost $\mathsf{OPT}(\mathcal{D}, \mathbf{q}, \theta)$ is the minimum $\sum_{i=1}^{d} b_i$ for position vectors $\mathbf{b}$ such that $\varphi_{\mathsf{TC}}(\mathbf{b}) = \mathtt{True}$. When it is clear from the context, we simply denote $\mathsf{OPT}(\mathcal{D}, \mathbf{q}, \theta)$ as $\mathsf{OPT}(\theta)$ or OPT.

At a position $\mathbf{b}$, a traversal strategy makes its decision locally based on what has been observed in the inverted lists up to that point, so the capability of making globally optimal decisions is limited. As a result, traversal strategies are often designed as simple heuristics, such as the lockstep strategy in the baseline approach. The lockstep strategy has a $d \cdot \mathsf{OPT}$ near-optimal bound which is loose in the high-dimensionality setting.

In this section, we present a traversal strategy for cosine threshold queries with tighter near-optimal bound by taking into account that the index values are skewed in many realistic scenarios. We approach the (near-)optimal traversal strategy in two steps.

First, we consider the simplified case with the unit-vector constraint ignored so that the problem is reduced to inner product queries. We propose a general traversal strategy that relies on convex hulls pre-computed from the inverted lists during indexing. During the gathering phase, these convex hulls are accessed as auxiliary data during the traversal to provide information on the increase/decrease rate towards the stopping condition. The hull-based traversal strategy not only makes fast decisions (in $\mathcal{O}(\log d)$ time) but is near-optimal (Corollary 1) under a reasonable assumption. In particular, we show that if the distance between any two consecutive convex hull vertices of the inverted lists is bounded by a constant c, the access cost of the strategy is at most $\mathsf{OPT} + c$. Experiments on real data show that this constant is small in practice.

The hull-based traversal strategy is quite general, as it applies to a large class of functions beyond inner product called the *decomposable functions*, which have the form

$\sum_{i=1}^{d} f_i(s_i)$ where each f_i is a non-decreasing real function of a single dimension s_i. Obviously, for a fixed query $\mathbf{q}$, the inner product $\mathbf{q} \cdot \mathbf{s}$ is a special case of decomposable functions, where each $f_i(s_i) = q_i \cdot s_i$. We show that the near-optimality result for inner product queries can be generalized to any decomposable function (Theorem 6).

Next, in Section 4.4, we consider the cosine queries by taking the normalization constraint into account. Although the function $\mathsf{MS}(\cdot)$ used in the tight stopping condition φ_{TC} is not decomposable so the same technique cannot be directly applied, we show that the hull-based strategy can be adapted by approximating $\mathsf{MS}(\cdot)$ with a decomposable function. In addition, we show that with a properly chosen approximation, the hull-based strategy is near-optimal with a small adjustment to the input threshold θ, meaning that the access cost is bounded by $\mathsf{OPT}(\theta - \epsilon) + c$ for a small ϵ (Theorem 7). Under the same experimental setting, we verify that ϵ is indeed small in practice.

4.1 Decomposable Functions

We start with defining the decomposable functions for which the hull-based traversal strategies can be applied:

Definition 9 (Decomposable Function) A decomposable function $F(\mathbf{s})$ is a real d-dimensional function where $F(\mathbf{s}) = \sum_{i=1}^{d} f_i(s_i)$ and each f_i is a non-decreasing real function.

Given a decomposable function F, the corresponding stopping condition is called a *decomposable condition*, which we define next.

Definition 10 (Decomposable Condition) A decomposable condition φ_F is a boolean function $\varphi_F(\mathbf{b}) = \big(F(L[\mathbf{b}]) < \theta\big)$ where F is a decomposable function and θ is a fixed threshold.

When the unit vector constraint is lifted, the decomposable condition is tight and complete for any scoring function F and threshold θ. The condition is tight because an unseen vector $\mathbf{s}$ can now be arbitrarily close to $L[\mathbf{b}]$ so the traversal has to continue as long as $F(L[\mathbf{b}]) \geq \theta$. As a result, the goal of designing a traversal strategy for F is to have the access cost as close as possible to OPT when the stopping condition is φ_F.

4.2 The max-reduction traversal strategy

To illustrate the high-level idea of the hull-based approach, we start with a simple greedy traversal strategy called the *Max-Reduction* traversal strategy $\mathcal{T}_{\mathsf{MR}}(\cdot)$. The strategy works as follows: at each snapshot, move the pointer b_i on the inverted list L_i that results in the maximal reduction on the score $F(L[\mathbf{b}])$. Formally, we define

$$\mathcal{T}_{\mathsf{MR}}(\mathbf{b}) = \underset{1 \leq i \leq d}{\operatorname{argmax}} \left(F(L[\mathbf{b}]) - F(L[\mathbf{b} + \mathbb{1}_i])\right) = \underset{1 \leq i \leq d}{\operatorname{argmax}} \left(f_i(L_i[b_i]) - f_i(L_i[b_i + 1])\right)$$

where $\mathbb{1}_i$ is the vector with 1 at dimension i and 0's else where. Such a strategy is reasonable since one would like $F(L[\mathbf{b}])$ to drop as fast as possible, so that once it is below θ, the stopping condition φ_F will be triggered and terminate the traversal.

Note that there are instances where the max-reduction strategy can be far from optimal. The strategy suffers from the drawback of making locally optimal but globally suboptimal decisions. The pointer b_i to an inverted list L_i might never be moved if choosing the current b_i only results in a small decrease in the score $F(L[\mathbf{b}])$, but there is a much larger decrease several steps ahead.

Is it possible that it is optimal under some assumption? The answer is positive: if for every list L_i, the values of $f_i(L_i[b_i])$ are decreasing at decelerating rate, then we can prove that its access cost is optimal. We state this ideal assumption next.

Assumption 1 (Ideal Convexity) *For every inverted list L_i, let $\Delta_i[j] = f_i(L_i[j]) - f_i(L_i[j+1])$ for $0 \leq j < |L_i|$.*[6] *The list L_i is ideally convex if the sequence Δ_i is non-increasing, i.e., $\Delta_i[j+1] \leq \Delta_i[j]$ for every j. Equivalently, the piecewise linear function passing through the points $\{(j, f_i(L_i[j]))\}_{0 \leq j \leq |L_i|}$ is convex for each i. A database $\mathcal{D}$ is ideally convex if every L_i is ideally convex.*

An example of an inverted list satisfying the above assumption is shown in Figure 4(a). The max-reduction strategy $\mathcal{T}_{\mathsf{MR}}$ is optimal under the ideal convexity assumption:

Theorem 5 (Ideal Optimality) *Given a decomposable function F, for every ideally convex database $\mathcal{D}$ and every threshold θ, the access cost of $\mathcal{T}_{\mathsf{MR}}$ is exactly* OPT.

We prove Theorem 5 with a simple greedy argument (detailed next): each move of $\mathcal{T}_{\mathsf{MR}}$ always results in the globally maximal reduction in the scoring function as guaranteed by the convexity condition.

Proof Let $\{\mathbf{b}_t\}_{1 \leq t \leq k}$ be the sequence of position vectors produced by the strategy $\mathcal{T}_{\mathsf{MR}}$.

Since each Δ_i is non-increasing and the strategy $\mathcal{T}_{\mathsf{MR}}$ chooses the dimension i with the maximal $\Delta_i[b_i]$, then at each step t, the multiset $\{\Delta_i[j] | 1 \leq i \leq d, 0 \leq j \leq \mathbf{b}_t[i]\}$ contains the first t largest values of all the $\Delta_i[j]$'s from the multiset $\{\Delta_i[j] | 1 \leq i \leq d, 0 \leq j < |L_i|\}$. Since the score $F(L[\mathbf{b}_t])$ equals

$$\sum_{i=1}^{d} f_i(L_i[0]) - \sum_{i=1}^{d} \sum_{j=1}^{\mathbf{b}_t[i]} \Delta_i[j] \ ,$$

it follows that for each $\mathbf{b}_t$ of $\mathcal{T}_{\mathsf{MR}}$, the score $F(L[\mathbf{b}_t])$ is the lowest score possible for any position vector reachable in t steps. Thus, if the optimal access cost OPT is t with an optimal stopping position $\mathbf{b}_{\mathsf{OPT}}$, then $\mathbf{b}_t$, the t-th position of $\mathcal{T}_{\mathsf{MR}}$, satisfies that $F(L[\mathbf{b}_t]) \leq F(L[\mathbf{b}_{\mathsf{OPT}}]) < \theta$. So $\mathcal{T}_{\mathsf{MR}}$ is optimal. □

4.3 The hull-based traversal strategy

Theorem 5 provides a strong performance guarantee but the ideal convexity assumption is usually not true on real datasets. As a result, the $\mathcal{T}_{\mathsf{MR}}$ strategy has no performance guarantee in general.

[6] Recall that $L_i[0] = 1$.

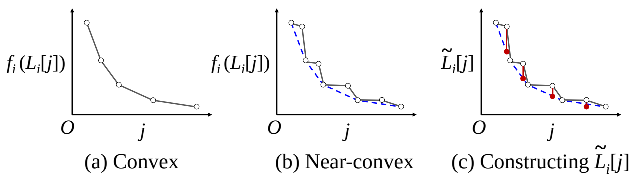


Fig. 4 Convexity and near-convexity

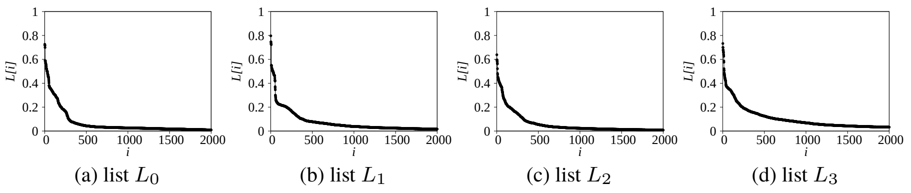


Fig. 5 The skewed inverted lists in mass spectrometry

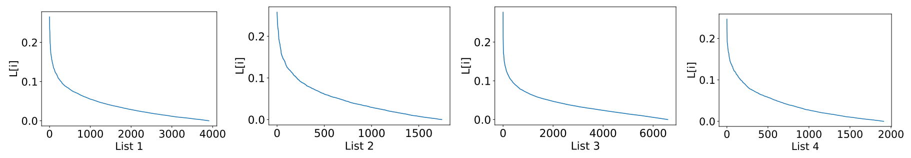


Fig. 6 The inverted lists of the first 4 dimensions of a document dataset. The dataset [74] contains 515k hotel reviews from the booking.com website . We convert each hotel review to a 300d vector by applying the doc2vec [25] model to transform each hotel review to a vector representation.

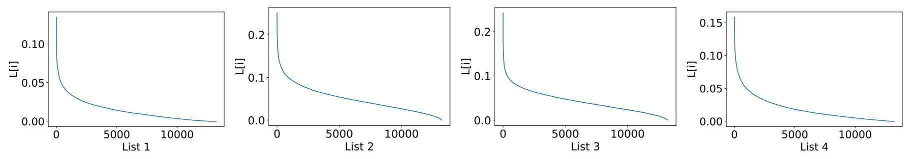


Fig. 7 The inverted lists of the first 4 dimensions of an image dataset. The dataset [50] contains 13,000 images of human faces collected from the web. We use the ResNet-18 model [40] to convert each image to a 512d vector.

In most practical scenarios that we have seen, we can bring the traversal strategy $\mathcal{T}_{\mathsf{MR}}$ to practicality by considering a relaxed version of Assumption 1. Informally, instead of assuming that each list $f_i(L_i)$ forms a convex piecewise linear function, we assume that $f_i(L_i)$ is "mostly" convex, meaning that if we compute the *lower convex hull* [22] of $f_i(L_i)$, the gap between any two consecutive vertices on the convex hull is small.[7] Intuitively, the relaxed assumption implies that the values at each list are decreasing at "approximately" decelerating speed. It allows list segments that do not follow the overall deceleration trend, as long as their lengths are bounded by a constant. We verified this property in the mass spectrometry dataset as illustrated in Figure 5, a document dataset, and an image dataset (Figure 6 and 7).

[7] We denote by $f_i(L_i)$ the list $[f_i(L_i[0]), f_i(L_i[1]), \ldots]$ for every L_i.

Assumption 2 (Near-Convexity) *For every inverted list L_i, let H_i be the lower convex hull of the set of 2-D points $\{(j, f_i(L_i[j]))\}_{0 \le j \le |L_i|}$ represented by a set of indices $\mathsf{H}_i = \{j_1, \ldots, j_n\}$ where for each $1 \le k \le n$, $(j_k, f_i(L_i[j_k]))$ is a vertex of the convex hull. The list L_i is near-convex if for every k, $j_{k+1} - j_k$ is upper-bounded by some constant c. A database $\mathcal{D}$ is near-convex if every inverted list L_i is near-convex with the same constant c, which we refer to as the convexity constant.*

Example 1 Intuitively, the near-convexity assumption captures the case where each $f_i(L_i)$ is decreasing with *approximately* decelerating speed, so the number of points between two convex hull vertices should be small. For example, when f_i is a linear function, the list L_i shown in Figure 4(b) is near-convex with convexity constant 2 since there is at most 1 point between each pair of consecutive vertices of the convex hull (dotted line). In the ideal case shown in Figure 4(a), the constant is 1 when the decrease between successive values is strictly decelerating.

Imitating the max-reduction strategy, for every pair of consecutive indices j_k, j_{k+1} in H_i and for every index $j \in [j_k, j_{k+1})$, let $\tilde{\Delta}_i[j] = \dfrac{f_i(L_i[j_k]) - f_i(L_i[j_{k+1}])}{j_{k+1} - j_k}$. Since the $(j_k, f_i(L_i[j_k]))$'s are vertices of a lower convex hull and each f_i is non-decreasing, each sequence $\tilde{\Delta}_i$ is non-decreasing. Then the *hull-based* traversal strategy is simply defined as

$$\mathcal{T}_{\mathsf{HL}}(\mathbf{b}) = \operatorname*{argmax}_{1 \le i \le d}(\tilde{\Delta}_i[b_i]). \tag{9}$$

Remark on data structures. In a practical implementation, to answer queries with scoring function F using the hull-based strategy, the lower convex hulls need to be ready before the traversal starts. If F is a general function unknown a priori, the convex hulls need to be computed online which is not practical. Fortunately, when F is the inner product $F(\mathbf{s}) = \mathbf{q} \cdot \mathbf{s}$ parameterized by the query $\mathbf{q}$, each convex hull H_i is exactly the convex hull for the points $\{(j, L_i[j])\}_{0 \le i \le |L_i|}$ from L_i. This is because the slope from any two points $(j, f_i(L_i[j]))$ and $(k, f_i(L_i[k]))$ is $\dfrac{q_i L_i[j] - q_i L_i[k]}{j - k}$, which is exactly the slope from $(j, L_i[j])$ and $(k, L_i[k])$ multiplied by q_i. So by using the standard convex hull algorithm [34], H_i can be pre-computed in $\mathcal{O}(|L_i|)$ time. Then the set of the convex hull vertices H_i can be stored as inverted lists and accessed for computing the $\tilde{\Delta}_i$'s during query processing. In the ideal case, H_i can be as large as $|L_i|$ but is much smaller in practice.

Moreover, during the traversal using the strategy $\mathcal{T}_{\mathsf{HL}}$, choosing the maximum $\tilde{\Delta}_i[b_i]$ at each step can be done in $\mathcal{O}(\log d)$ time using a max heap. This satisfies the requirement that the traversal strategy is efficiently computable.

Near-optimality results. We show that the hull-based strategy $\mathcal{T}_{\mathsf{HL}}$ is near-optimal under the near-convexity assumption.

Theorem 6 *Given a decomposable function F, for every near-convex database $\mathcal{D}$ and every threshold θ, the access cost of $\mathcal{T}_{\mathsf{HL}}$ is strictly less than $\mathsf{OPT} + c$ where c is the convexity constant.*

When the assumption holds with a small convexity constant, this near-optimality result provides a much tighter bound compared to the $d \cdot \mathsf{OPT}$ bound in the TA-inspired baseline. This is achieved under data assumption and by keeping the convex hulls as auxiliary data structure, so it does not contradict the lower bound results on the approximation ratio [30].

Proof Let $\mathcal{B} = \{\mathbf{b}_i\}_{i\geq 0}$ be the sequence of position vectors generated by $\mathcal{T}_{\mathsf{HL}}$. We call a position vector $\mathbf{b}$ a *boundary position* if every b_i is the index of a vertex of the convex hull H_i. Namely, $b_i \in \mathsf{H}_i$ for every $i \in [d]$. Notice that if we break ties consistently during the traversal of $\mathcal{T}_{\mathsf{HL}}$, then for any position $\mathbf{b}''$ between any pair of consecutive boundary positions $\mathbf{b}$ and $\mathbf{b}'$, $\mathcal{T}_{\mathsf{HL}}(\mathbf{b}'')$ will always be the same as $\mathcal{T}_{\mathsf{HL}}(\mathbf{b})$. We call the subsequence positions $\{\mathbf{b}_i\}_{l\leq i<r}$ of $\mathcal{B}$ where $\mathbf{b}_l = \mathbf{b}$ and $\mathbf{b}_r = \mathbf{b}'$ a *segment* with boundaries $(\mathbf{b}_l, \mathbf{b}_r)$. We show the following lemma.

Lemma 2 *For every boundary position vector* $\mathbf{b}$ *generated by* $\mathcal{T}_{\mathsf{HL}}$*, we have* $F(L[\mathbf{b}]) \leq F(L[\mathbf{b}^*])$ *for every position vector* $\mathbf{b}^*$ *where* $|\mathbf{b}^*| = |\mathbf{b}|$.

Intuitively, the above lemma says that if the traversal of $\mathcal{T}_{\mathsf{HL}}$ reaches a boundary position $\mathbf{b}$, then the score $F(L[\mathbf{b}])$ is the minimal possible score obtained by any traversal sequence of at most $|\mathbf{b}|$ steps.

Lemma 2 is sufficient for Theorem 6 because of the following. Suppose $\mathbf{b}_{\mathsf{stop}}$ is the stopping position in $\mathcal{B}$, which means that $\mathbf{b}_{\mathsf{stop}}$ is the first position in $\mathcal{B}$ that satisfies φ_F and the access cost is $|\mathbf{b}_{\mathsf{stop}}|$. Let $\{\mathbf{b}_i\}_{l\leq i<r}$ be the segment that contains $\mathbf{b}_{\mathsf{stop}}$. Given Lemma 2, Theorem 6 holds trivially if $\mathbf{b}_{\mathsf{stop}} = \mathbf{b}_l$. It remains to consider the case $\mathbf{b}_{\mathsf{stop}} \neq \mathbf{b}_l$. Since the traversal does not stop at $\mathbf{b}_l$, we have $F(L[\mathbf{b}_l]) \geq \theta$. By Lemma 2, $\mathbf{b}_l$ is the position with minimal $F(L[\cdot])$ obtained in $|\mathbf{b}_l|$ steps so $|\mathbf{b}_l| \leq \mathsf{OPT}$. Since $|\mathbf{b}_{\mathsf{stop}}| - |\mathbf{b}_l| < |\mathbf{b}_r| - |\mathbf{b}_l| \leq c$, we have that $|\mathbf{b}_{\mathsf{stop}}| < \mathsf{OPT} + c$. We illustrate this in Figure 8.

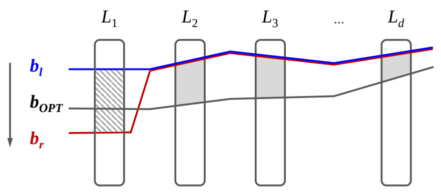


Fig. 8 $(\mathbf{b}_l, \mathbf{b}_r)$: the two boundary positions surrounding the stopping position $\mathbf{b}_{\mathsf{stop}}$ of $\mathcal{T}_{\mathsf{HL}}$; $\mathbf{b}_{\mathsf{OPT}}$: the optimal stopping position; It is guaranteed that (1) $|\mathbf{b}_{\mathsf{stop}}| - |\mathbf{b}_l| < |\mathbf{b}_r| - |\mathbf{b}_l| \leq c$ and (2) $|\mathbf{b}_l| < |\mathbf{b}_{\mathsf{OPT}}|$.

Now, it remains to prove Lemma 2. We do so by generalizing the greedy argument in the proof of Theorem 5.

We construct a new collection of inverted lists $\{\tilde{L}_i\}_{1\leq i\leq d}$ from the original lists as follows. For every i and every pair of consecutive indices j_k, j_{k+1} of H_i, we assign

$$\tilde{L}_i[j] = f_i(L_i[j_k]) - (j - j_k) \cdot \tilde{\Delta}_i[j] \quad \text{for every } j \in [j_k, j_{k+1}].$$

Intuitively, we construct each $\tilde{L}_i$ by projecting the set of 2D points $\{(i, f_i(L_i[j]))\}_{j\in[j_k,j_{k+1}]}$ onto the line passing through the two boundary points with index j_k and j_{k+1}, which is essentially projecting the set of points onto the piecewise linear function defined by the convex hull vertices in H_i (See Figure 4(c) for an illustration). The new $\{\tilde{L}_i\}_{1 \le i \le d}$ satisfies the following properties.

(i) By the construction of each convex hull H_i, we have $\tilde{L}_i[j] \le f_i(L_i[j])$ for every i and j.

(ii) For every boundary position $\mathbf{b}$, we have $\tilde{L}[\mathbf{b}] = F(L[\mathbf{b}])$ since for every index j on a convex hull H_i, $\tilde{L}[j] = f_i(L_i[j])$.

(iii) The collection $\{\tilde{L}_i\}_{1 \le i \le d}$ is ideally convex[8]. In addition, the strategy $\mathcal{T}_{\mathsf{HL}}$ produces exactly the same sequence reduced by the max-reduction strategy $\mathcal{T}_{\mathsf{MR}}$ when $\{\tilde{L}_i\}_{1 \le i \le d}$ is given as the input. By the same analysis for Theorem 5, for every position vector $\mathbf{b}$ generated by $\mathcal{T}_{\mathsf{HL}}$, $\tilde{L}[\mathbf{b}]$ is minimal among all position vectors reached within $|\mathbf{b}|$ steps.

Combining (ii) and (iii), for every boundary position vector $\mathbf{b}$ generated by $\mathcal{T}_{\mathsf{HL}}$ and every $\mathbf{b}^*$ where $|\mathbf{b}^*| = \mathbf{b}$, we have $F(L[\mathbf{b}]) = \tilde{L}[\mathbf{b}] \le \tilde{L}[\mathbf{b}^*]$. Finally, by (i) and since F is non-decreasing, we have $\tilde{L}[\mathbf{b}^*] \le F(L[\mathbf{b}^*])$ so $F(L[\mathbf{b}]) \le F(L[\mathbf{b}^*])$ for every $\mathbf{b}^*$. This completes the proof of Theorem 6. □

Since the baseline stopping condition φ_{BL} is tight and complete for inner product queries, one immediate implication of Theorem 6 is that

Corollary 1 *(Informal) The hull-based strategy $\mathcal{T}_{\mathsf{HL}}$ for inner product queries is near-optimal under the near-convexity assumption.*

Verifying the assumption. We demonstrate the practical impact of the near-optimality result in real mass spectrometry datasets. The near-convexity assumption requires that the gap between any two consecutive convex hull vertices has bounded size, which is hard to achieve in general. According to the proof of Theorem 6, for a given query, the difference from the optimal access cost is at most the size of the gap between the two consecutive convex hull vertices containing the last move of the strategy (the $\mathbf{b}_l$ and $\mathbf{b}_r$ in Figure 8). The size of this gap can be much smaller than the global convexity constant c, so the overall precision can be much better in practice. We verify this by running a set of 1,000 real queries on the dataset[9]. The gap size is 163.04 in average, which takes only 1.3% of the overall access cost of traversing the indices. This indicates that the near-optimality guarantee holds in the mass spectrometry dataset.

We also observed that the near-convexity assumption holds for document and image vectors, thus the proposed algorithms can potentially be applied to document/image similarity search. In the review dataset mentioned above, we ran 100 randomly sampled queries on a random subset of 10,000 reviews with threshold $\theta = 0.6$ using the hull-based strategy. The total number of accesses is 4,762,040 and the total sizes of the last gap is 374,521 which means that the cost additional to the optimal access is no more than 7.86% of the overall access cost. In the face image dataset, under the same setting, we ran 100 random queries of faces in the whole dataset with the same threshold $\theta = 0.6$. The total access cost is 118,795,452, the total size of last gaps is 473,999, so the additional access cost compared to OPT is no more than 0.40% of the overall access cost.

[8] where each f_i of the decomposable function F is the identity function

[9] `https://proteomics2.ucsd.edu/ProteoSAFe/index.jsp`

4.4 The traversal strategy for cosine

Next, we consider traversal strategies which take into account the unit vector constraint posed by the cosine function, which means that the tight and complete stopping condition is φ_{TC} introduced in Section 3. However, since the scoring function MS in φ_{TC} is not decomposable, the hull-based technique cannot be directly applied. We adapt the technique by approximating the original MS with a decomposable function $\tilde{F}$. Without changing the stopping condition φ_{TC}, the hull-based strategy can then be applied with the convex hull indices constructed with the approximation $\tilde{F}$. In the rest of this section, we first generalize the result in Theorem 6 to scoring functions having decomposable approximations and show how the hull-based traversal strategy can be adapted. Next, we show a natural choice of the approximation for MS with practically tight near-optimal bounds. Finally, we discuss data structures to support fast query processing using the traversal strategy.

We start with some additional definitions.

Definition 11 A d-dimensional function F is decomposably approximable if there exists a decomposable function $\tilde{F}$, called the *decomposable approximation* of F, and two non-negative constants ϵ_1 and ϵ_2 such that $F(\mathbf{s}) - \tilde{F}(\mathbf{s}) \in [-\epsilon_1, \epsilon_2]$ for every vector s.

When applied to a decomposably approximable function F, the hull-based traversal strategy $\mathcal{T}_{\mathsf{HL}}$ is adapted by constructing the convex hull indices and the $\{\tilde{\Delta}_i\}_{1 \le i \le d}$ using the approximation $\tilde{F}$. The following can be obtained by generalizing Theorem 6:

Theorem 7 *Given a function F approximable by a decomposable function $\tilde{F}$ with constants (ϵ_1, ϵ_2), for every near-convex database $\mathcal{D}$ wrt $\tilde{F}$ and every threshold θ, the access cost of $\mathcal{T}_{\mathsf{HL}}$ is strictly less than* $\mathsf{OPT}(\theta - \epsilon_1 - \epsilon_2) + c$ *where c is the convexity constant.*

Proof Denote by $\mathbf{b}_{\mathsf{last}}$ the last boundary position generated by $\mathcal{T}_{\mathsf{HL}}$ that does not satisfy the tight stopping condition for F (which is φ_{TC} when F is MS) so $F(L[\mathbf{b}_l]) \ge \theta$. Note that $\mathbf{b}_{\mathsf{last}}$ is the $\mathbf{b}_l$ in the proof of Lemma 2 when $\mathbf{b}_l \neq \mathbf{b}_{\mathsf{stop}}$. It is sufficient to show that for every vector $\mathbf{b}^*$ where $|\mathbf{b}^*| = |\mathbf{b}_{\mathsf{last}}|$, $F(L[\mathbf{b}^*]) \ge \theta - \epsilon_1 - \epsilon_2$ so no traversal can stop within $|\mathbf{b}_{\mathsf{last}}|$ steps, implying that the final access cost is no more than $|\mathbf{b}_{\mathsf{last}}| + c$ which is bounded by $\mathsf{OPT}(\theta - \epsilon_1 - \epsilon_2) + c$.

According to Lemma 2, we know that for every such $\mathbf{b}^*$, $\tilde{F}(L[\mathbf{b}^*]) \ge \tilde{F}(L[\mathbf{b}_{\mathsf{last}}])$. By definition of the approximation $\tilde{F}$, we know that $F(L[\mathbf{b}^*]) \ge \tilde{F}(L[\mathbf{b}^*]) - \epsilon_1$ and $\tilde{F}(L[\mathbf{b}_{\mathsf{last}}]) \ge F(L[\mathbf{b}_{\mathsf{last}}]) - \epsilon_2$. Combined together, for every $\mathbf{b}^*$ where $|\mathbf{b}^*| = |\mathbf{b}_{\mathsf{last}}|$, we have

$$F(L[\mathbf{b}^*]) \ge \tilde{F}(L[\mathbf{b}^*]) - \epsilon_1 \ge \tilde{F}(L[\mathbf{b}_{\mathsf{last}}]) - \epsilon_1 \ge F(L[\mathbf{b}_{\mathsf{last}}]) - \epsilon_1 - \epsilon_2 \ge \theta - \epsilon_1 - \epsilon_2.$$

This completes the proof of Theorem 7. □

Choosing the decomposable approximation. By Theorem 7, it is important to choose an approximation $\tilde{F}$ of MS with small ϵ_1 and ϵ_2 for a tight near-optimality result. By inspecting the formula (3) of MS, one reasonable choice of $\tilde{F}$ can be obtained by replacing the term τ with a fixed constant $\tilde{\tau}$. Formally, let

$$\tilde{F}(L[\mathbf{b}]) = \sum_{i=1}^{d} \min\{q_i \cdot \tilde{\tau}, L_i[b_i]\} \cdot q_i \tag{10}$$

be the decomposable approximation of MS where each component is a non-decreasing function $f_i(x) = \min\{q_i \cdot \tilde{\tau}, x\} \cdot q_i$ for $i \in [d]$.

Ideally, the approximation is tight if the constant $\tilde{\tau}$ is close to the final value of τ which is unknown in advance. We argue that when $\tilde{\tau}$ is properly chosen, the approximation parameter $\epsilon_1 + \epsilon_2$ is very small.

With the analysis detailed next, we obtain the following upper bound of ϵ by upper-bounding ϵ_1 and ϵ_2:

$$\epsilon \le \max\{0, \tilde{\tau} - 1/\mathsf{MS}(L[\mathbf{b}_{\mathsf{last}}])\} + \mathsf{MS}(L[\mathbf{b}_{\mathsf{last}}]) - \tilde{F}(L[\mathbf{b}_{\mathsf{last}}]).$$

Upper bounding ϵ_1: A trivial upper bound of ϵ_1 is $\tilde{\tau} - 1$ since τ is 1 when the traversal starts and the gap $\tilde{F}(L[\mathbf{b}]) - \mathsf{MS}(L[\mathbf{b}])$ is maximized when $\mathbf{b} = \mathbf{0}$. This upper bound can be improved as follows. We notice that in the proof of Theorem 7, given $|\mathbf{b}^*| = |\mathbf{b}_{\mathsf{last}}|$, we need to have $F(L[\mathbf{b}^*]) \ge \theta - \epsilon_1 - \epsilon_2$ for every such $\mathbf{b}^*$, which is equivalent to requiring that this property holds for the $\mathbf{b}^*$ that minimizes $F(L[\mathbf{b}^*])$ given $|\mathbf{b}^*| = |\mathbf{b}_{\mathsf{last}}|$. This $\mathbf{b}^*$ satisfies that $F(L[\mathbf{b}^*]) \le F(L[\mathbf{b}_{\mathsf{last}}])$ and $F(L[\mathbf{b}_{\mathsf{last}}])$ is known when the query is executed. This upper bound of $F(L[\mathbf{b}^*])$ implies a lower bound of τ at position $\mathbf{b}^*$, which also implies the following lower bound of $\mathsf{MS}(L[\mathbf{b}^*]) - \tilde{F}(L[\mathbf{b}^*])$:

Lemma 3 *Let* $\mathbf{b}$ *be an arbitrary position vector and let* $\mathbf{b}^*$ *be the position vector such that*

$$\mathbf{b}^* = \underset{\mathbf{b}':|\mathbf{b}|=|\mathbf{b}'|}{\operatorname{argmin}} \{\mathsf{MS}(L[\mathbf{b}'])\}.$$

Then

$$(\dagger) \quad \mathsf{MS}(L[\mathbf{b}^*]) - \tilde{F}(L[\mathbf{b}^*]) \ge \min\{0, 1/\mathsf{MS}(L[\mathbf{b}]) - \tilde{\tau}\}.$$

where $\tilde{F}$ *is the decomposable function where each component is* $f_i(x) = \min\{\tilde{\tau} \cdot q_i, x\} \cdot q_i$ *for constant* $\tilde{\tau}$ *and every* $1 \le i \le d$.

Proof Let τ^* be the value of τ at $L[\mathbf{b}^*]$. We consider two cases separately: $\tau^* \ge \tilde{\tau}$ and $\tau^* < \tilde{\tau}$.

Case One: When $\tau^* \ge \tilde{\tau}$, since each f_i is non-decreasing wrt $\tilde{\tau}$, we have

$$f_i(x) = \min\{\tilde{\tau} \cdot q_i, x\} \cdot q_i \le \min\{\tau^* \cdot q_i, x\} \cdot q_i$$

thus $\mathsf{MS}(L[\mathbf{b}^*]) \ge \tilde{F}(L[\mathbf{b}^*])$.

Case Two: Suppose $\tau^* < \tilde{\tau}$. We show the following Lemma.

Lemma 4 *For every position* $\mathbf{b}$ *and constant* c, $\mathsf{MS}(L[\mathbf{b}]) < c$ *implies that the* τ *at* $L[\mathbf{b}]$ *is at least* $1/c$.

Recall the notations $\mathsf{LQ} = \sum_{i:L_i[b_i]<\tau\cdot q_i} L_i[b_i]\cdot q_i$, $\mathsf{Q2} = \sum_{i:L_i[b_i]<\tau\cdot q_i} q_i^2$ and $\mathsf{L2} = \sum_{i:L_i[b_i]<\tau\cdot q_i} L_i[b_i]^2$. Then τ satisfies that

$$\mathsf{LQ} + (1-\mathsf{Q2})\cdot\tau < c \tag{11}$$

and

$$\mathsf{L2} + (1-\mathsf{Q2})\cdot\tau^2 = 1. \tag{12}$$

By rewriting Equation (12), we have $(1-\mathsf{Q2}) = (1-\mathsf{L2})/\tau^2$. Plug this into (11), we have $\mathsf{LQ} + (1-\mathsf{L2})/\tau < c$. Since $L_i[b_i]\le\tau\cdot q_i$, we have $\mathsf{LQ}\ge\mathsf{L2}/\tau$ so $1/\tau < c$ which means $\tau > 1/c$. This completes the proof of Lemma 4.

We know that since $\mathbf{b}^*$ minimizes $\mathsf{MS}(L[\mathbf{b}'])$ among all $\mathbf{b}'$ with $|\mathbf{b}'| = |\mathbf{b}|$, we have $\mathsf{MS}(L[\mathbf{b}^*]) \le \mathsf{MS}(L[\mathbf{b}])$. By Lemma 4, we have $\tau^* \ge 1/\mathsf{MS}(L[\mathbf{b}])$.

Now consider $\mathsf{MS}(L[\mathbf{b}^*]) - \tilde{F}(L[\mathbf{b}^*])$. Since $\tau^* < \tilde{\tau}$, $\mathsf{MS}(L[\mathbf{b}^*]) - \tilde{F}(L[\mathbf{b}^*])$ can be written as the sum of the following 3 terms:

– $\sum_{i:L_i[b_i]/q_i<\tau^*}(L_i[b_i]\cdot q_i - L_i[b_i]\cdot q_i)$, which is always 0,
– $\sum_{i:\tau^*\le L_i[b_i]/q_i<\tilde{\tau}}(q_i^2\cdot\tau^* - L_i[b_i]\cdot q_i)$ and
– $\sum_{i:\tilde{\tau}\le L_i[b_i]/q_i}(q_i^2\cdot\tau^* - q_i^2\cdot\tilde{\tau})$.

In the second term, since each $L_i[b_i]$ is at most $q_i\cdot\tilde{\tau}$, so this term is greater than or equal to $\sum_{i:\tau^*\le L_i[b_i]/q_i<\tilde{\tau}}(q_i^2\cdot\tau^* - q_i^2\cdot\tilde{\tau})$. Adding them together, we have $\mathsf{MS}(L[\mathbf{b}^*]) - \tilde{F}(L[\mathbf{b}^*])$ is lower bounded by

$$\sum_{i:\tau^*\le L_i[b_i]/q_i} q_i^2\cdot(\tau^* - \tilde{\tau})$$

which is greater than or equal to $\tau^* - \tilde{\tau}$. Since $\tau^* \ge 1/\mathsf{MS}(L[\mathbf{b}])$, we have $\mathsf{MS}(L[\mathbf{b}^*]) - \tilde{F}(L[\mathbf{b}^*]) \ge 1/\mathsf{MS}(L[\mathbf{b}]) - \tilde{\tau}$. Finally, combining case one and two together completes the proof of Lemma 3.

Thus, when the query is given, ϵ_1 is at most $\max\{0, \tilde{\tau} - 1/\mathsf{MS}(L[\mathbf{b}_{\mathsf{last}}])\}$. □

Upper bounding ϵ_2: In general, there is no upper bound for τ since it can be as large as $L_i[b_i]/q_i$ for some i. The gap $\mathsf{MS}(L[\mathbf{b}]) - \tilde{F}(L[\mathbf{b}])$ can be close to 1. However, the proof of Theorem 7 only requires ϵ_2 to be at least the difference between MS and $\tilde{F}$ at position $\mathbf{b}_{\mathsf{last}}$. So ϵ_2 is upper-bounded by $\mathsf{MS}(L[\mathbf{b}_{\mathsf{last}}]) - \tilde{F}(L[\mathbf{b}_{\mathsf{last}}])$ for a given query.

Summarizing the above analysis, the approximation factor ϵ is determined by the following two factors:

1. how much the approximation $\tilde{F}$ is smaller than the scoring function MS at $\mathbf{b}_{\mathsf{last}}$, the last boundary position that does not satisfy the stopping condition during the traversal, and
2. how much $\tilde{F}$ is bigger than MS at the optimal position with exactly $|\mathbf{b}_{\mathsf{last}}|$ steps that minimizes MS.

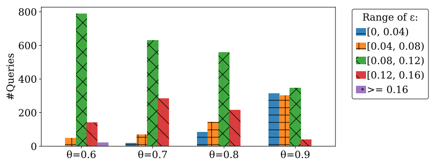


Fig. 9 The distribution of ϵ

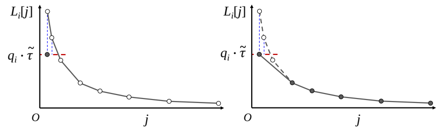


Fig. 10 The construction of convex hull $\tilde{\mathsf{H}}_i$

This yields the following upper bound of ϵ:

$$\epsilon \le \max\{0, \tilde{\tau} - 1/\mathsf{MS}(L[\mathbf{b}_{\mathsf{last}}])\} + \mathsf{MS}(L[\mathbf{b}_{\mathsf{last}}]) - \tilde{F}(L[\mathbf{b}_{\mathsf{last}}]). \tag{13}$$

Verifying the near-optimality. The upper bound 13 provides a convenient way of computing an upper bound of ϵ for each query at execution time. Next, we experimentally verify that the upper bound of ϵ is small. We ran the same set of queries as in Section 4.3 and show the distribution of ϵ's upper bounds in Figure 9. We set $\tilde{\tau} = 1/\theta$ for all queries so the first term of (13) becomes zero. Note that further tuning the parameter $\tilde{\tau}$ can yield better ϵ, but it is not done here for simplicity. Overall, the fraction of queries with an upper bound <0.12 (the sum of the first 3 bars for all θ) is 82.5% and the fraction of queries with $\epsilon > 0.16$ is 0.5%.

The convexity constant remains small similar to the case with inner product queries. The average of the convexity constant c is 193.39, which is only 4.8% of the overall access cost.

In addition to the above results, we have also conducted an experimental study on comparing the TA-baseline and the hull-based strategy in their access cost and the actual running time in real-world settings. We direct the interested readers to [76] (Chapter 4.8).

Remark on data structures. Similar to the inner product case, it is necessary that the convex hulls for $\mathcal{T}_{\mathsf{HL}}$ can be efficiently obtained without a full computation when a query comes in. For every $i \in [d]$, we let $\tilde{\mathsf{H}}_i$ be the convex hull for the i-th component f_i of $\tilde{F}$ and H_i be the convex hull constructed directly from the original inverted list L_i. Next, we show that each $\tilde{\mathsf{H}}_i$ can be efficiently obtained from H_i during query time so we only need to pre-compute the H_i's.

We observe that when $L_i[b_i] \ge q_i \cdot \tilde{\tau}$, $f_i(L_i[b_i])$ equals a fixed value $q_i^2 \cdot \tilde{\tau}$ otherwise is proportional to $L_i[b_i]$. As illustrated in Figure 10 (left), the list of values $\{f_i(L_i[j])\}_{j \ge 0}$ is essentially obtained by replacing the $L_i[j]$'s greater than $q_i \cdot \tilde{\tau}$ with $q_i \cdot \tilde{\tau}$. To obtain the new convex hull $\tilde{\mathsf{H}}_i$, we use the following Lemma:

Lemma 5 *For every $i \in [d]$, the convex hull $\tilde{\mathsf{H}}_i$ is*

- H_i, *if* $q_i \cdot \tilde{\tau} \ge L_i[0]$, *or*
- $\{0\}$, *if* $q_i \cdot \tilde{\tau} \le L_i[|L_i|]$, *otherwise*
- *a subset of* H_i *where an index* j_k *of* H_i *is in* $\tilde{\mathsf{H}}_i$ *iff* $k = 1$ *(and* $j_k = 0$*) or*

$$\big(q_i \cdot \tilde{\tau} - L_i[j_k]\big) \;/\; j_k \ge \big(L_i[j_k] - L_i[j_{k+1}]\big) \;/\; (j_{k+1} - j_k). \tag{14}$$

The first two options essentially cover the boundary cases where $q_i \cdot \tilde{\tau}$ is above/below all values in L_i. For the last case, the lemma says that $\tilde{\mathsf{H}}_i$ consists of the index 0 and a suffix of H_i which can be efficiently located using a binary search. We illustrate the construction of the new convex hull $\tilde{\mathsf{H}}_i$ in Figure 10 (right). Suppose that the maximum size of all H_i is h. The computation of the $\tilde{\mathsf{H}}_i$'s adds an extra $\mathcal{O}(d \log h)$ of overhead to the query processing time, which is insignificant in practice since h is likely to be much smaller than the size of the database.

Next, we provide a simple proof of Lemma 5.

Proof We use the following property derived from the classic Graham scan algorithm [34] for computing the lower convex hulls: for every vertex $(j_k, L_i[j_k])$ of the convex hull, the next vertex $(j_{k+1}, L_i[j_{k+1}])$ has the minimum slope (i.e., largest reduction rate from $(j_k, L_i[j_k])$) among the points $\{(j, L_i[j])\}_{j_k < j \le |L_i|}$. By this property, it suffices to show that the second vertex of $\tilde{\mathsf{H}}_i$ is in H_i. This is because the prefix of $f_i(L_i)$ that are equal to $q_i \cdot \tilde{\tau}$ cannot be part of $\tilde{\mathsf{H}}_i$ (except for $j = 0$). Starting from the index of the second vertex $\tilde{\mathsf{H}}_i$, $f_i(L_i[j]) = L_i[j]$ so the remaining part of $\tilde{\mathsf{H}}_i$ and H_i are identical.

Next, we prove by contradiction that the second vertex is in H_i. Suppose this vertex is not in H_i. We assume that the vertex is $B = (j', L_i[j'])$ and it is in between of two adjacent vertices $A = (j_a, L_i[j_a])$ and $C = (j_c, L_i[j_c])$ of H_i where $c = a + 1$ and $j_a < j' < j_c$. We illustrate the relative positions of point A, B, and C in Fig. 11. By the convex hull property aforementioned, because B is not in H_i, it is above the AC segment. Since B and C are in the new hull $\tilde{\mathsf{H}}_i$, the first vertex $(0, f_i(L_i[0]))$ is in the region marked as "(1)". Similarly, because A is not in $\tilde{\mathsf{H}}_i$, the vertex $(0, f_i(L_i[0]))$ can only be in region "(2)" which does not intersect with "(1)" – a contradiction. Therefore, the second vertex of $\tilde{\mathsf{H}}_i$ is in H_i. □

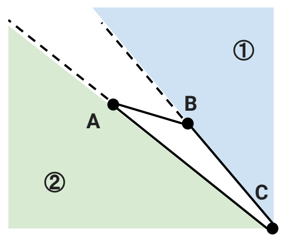


Fig. 11 The illustration of why the new convex hull $\tilde{\mathsf{H}}_i$ can only contain vertices from the original hull H_i. Here A and C are from H_i and B is not.

5 The Verification Phase

Next, we discuss optimizations in the verification phase where each gathered candidate is tested for θ-similarity with the query vector. The naive approach of verification is to fully access all the non-zero entries of each candidate **s** to compute the exact similarity score $\cos(\mathbf{q}, \mathbf{s})$ and compare it against θ, which takes $\mathcal{O}(d)$ time per candidate. Various techniques have been proposed [72,5,54] to decide θ-similarity by leveraging partial information about the candidate vector so the exact computation of $\texttt{cos}(\mathbf{q}, \mathrm{s})$ can be avoided. In this section, we revisit these existing techniques which we call *partial verification*. In addition, as a novel contribution, we show that in the presence of data skewness, partial verification can have a near-constant performance guarantee (Theorem 8).

Informally, while a vector s is scanned, based on what has been observed in s so far, it might be possible to infer that

(1) the similarity score $\texttt{cos}(\mathbf{q}, \mathrm{s})$ is certainly at least θ or
(2) the similarity score is certainly below θ.

In either case, we can stop without scanning the rest of s and return an accurate verification result. The problem is formally defined as follows:

Problem 1 (Partial Verification) A partially observed vector $\tilde{\mathrm{s}}$ is a d-dimensional vector in $(\mathbb{R}_{\geq 0} \cup \{\perp\})^d$ where $\perp$ refers to an unobserved value (i.e., null). Given a query $\mathbf{q}$ and a partially observed vector $\tilde{\mathrm{s}}$, compute whether for every vector s where $\mathrm{s}[i] = \tilde{\mathrm{s}}[i]$ for every $\tilde{\mathrm{s}}[i] \neq \perp$, it is $\texttt{cos}(\mathrm{s}, \mathbf{q}) \geq \theta$.

Intuitively, a partially observed vector $\tilde{\mathrm{s}}$ contains the entries of a candidate s either already observed during the gathering phase, or accessed for the actual values during the verification phase. The unobserved dimensions are replaced with a null value $\perp$. We say that a vector s is compatible with a partially observed one $\tilde{\mathrm{s}}$ if $\mathrm{s}[i] = \tilde{\mathrm{s}}[i]$ for every dimension i where $\tilde{\mathrm{s}}[i] \neq \perp$.

The partial verification problem can be solved by computing an upper and a lower bound of the cosine similarity between s and $\mathbf{q}$ when $\tilde{\mathrm{s}}$ is observed. We denote by $ub(\tilde{\mathrm{s}})$ and $lb(\tilde{\mathrm{s}})$ the upper bound and lower bound, so $ub(\tilde{\mathrm{s}}) = \max_{\mathrm{s}}\{\mathrm{s} \cdot \mathbf{q}\}$ and $lb(\tilde{\mathrm{s}}) = \min_{\mathrm{s}}\{\mathrm{s} \cdot \mathbf{q}\}$ where the maximum/minimum are taken over all s compatible with $\tilde{\mathrm{s}}$. By [72,5,54], the upper/lower bounds can be computed as follows:

Lemma 6 *Given a partially observed vector* $\tilde{\mathrm{s}}$ *and a query vector* $\mathbf{q}$*,*

$$ub(\tilde{\mathrm{s}}) = \sum_{\tilde{\mathrm{s}}[i] \neq \perp} \tilde{\mathrm{s}}[i] \cdot \mathbf{q}[i] + \sqrt{1 - \sum_{\tilde{\mathrm{s}}[i] \neq \perp} \tilde{\mathrm{s}}[i]^2} \cdot \sqrt{1 - \sum_{\tilde{\mathrm{s}}[i] \neq \perp} \mathbf{q}[i]^2}\,, \tag{15}$$

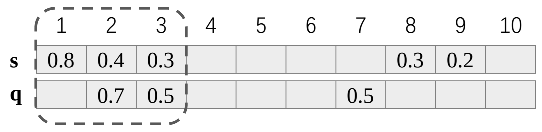
and

$$lb(\tilde{\mathrm{s}}) = \sum_{\tilde{\mathrm{s}}[i] \neq \perp} \tilde{\mathrm{s}}[i] \cdot \mathbf{q}[i] + \sqrt{1 - \sum_{\tilde{\mathrm{s}}[i] \neq \perp} \tilde{\mathrm{s}}[i]^2} \cdot \min_{\tilde{\mathrm{s}}[i] = \perp} \mathbf{q}[i]\,. \tag{16}$$

Example 2 Figure 12 shows an example of computing the lower/upper bounds of a partially scanned vector s with a query $\mathbf{q}$. Assume the first three dimensions have been scanned, then the lower bound and upper bound are computed as follows:

$$lb(\tilde{\mathrm{s}}) = 0.8 \times 0 + 0.4 \times 0.7 + 0.3 \times 0.5 = 0.43$$
$$ub(\tilde{\mathrm{s}}) = lb(\tilde{\mathrm{s}}) + \sqrt{1 - 0.8^2 - 0.4^2 - 0.3^2} \cdot \sqrt{1 - 0.7^2 - 0.5^2} = 0.6$$

If the threshold θ is 0.7, then it is certain that s is not θ-similar to $\mathbf{q}$ because $0.6 < 0.7$. The verification algorithm can stop and avoid the rest of the scan. Note that when $\mathbf{q}$ is sparse, most of the time we have $lb(\tilde{\mathrm{s}}) = \sum_{\tilde{\mathrm{s}}[i] \neq \perp} \tilde{\mathrm{s}}[i] \cdot \mathbf{q}[i]$ due to the existence of 0's.



	1	2	3	4	5	6	7	8	9	10
s	0.8	0.4	0.3					0.3	0.2	
q		0.7	0.5				0.5			

Fig. 12 An example of the lower and upper bounds

Performance Guarantee. Next, we show that when the data is skewed, partial verification achieves a strong performance guarantee. Along with each $\mathbf{s}$, we store a permutation π such that $\mathbf{s}[\pi[1]] \geq \mathbf{s}[\pi[2]] \geq \cdots \geq \mathbf{s}[\pi[d]]$. Then, $\mathbf{s}[i] \cdot \mathbf{q}[i]$ could be replaced by $\mathbf{s}[\pi[i]] \cdot \mathbf{q}[\pi[i]]$. We notice that partial verification saves data accesses when there is a gap between the true similarity score and the threshold θ. Intuitively, when this gap is close to 0, both the upper and lower bounds converge to θ as we scan $\mathbf{s}$ so we might not stop until the last element. When the gap is large and $\mathbf{s}$ is skewed, meaning that the first few values account for most of the candidate's weight, then the first few $\mathbf{s}[i] \cdot \mathbf{q}[i]$ terms can provide large enough information for the lower/upper bounds to decide θ-similarity. Formally,

Theorem 8 *Suppose that a vector* $\mathbf{s}$ *is skewed: there exists an integer* $k \leq d$ *and constant value* c *such that* $\sum_{i=1}^{k} \mathbf{s}[i]^2 \geq c$. *For every query* $(\mathbf{q}, \theta)$, *if* $|\mathtt{cos}(\mathbf{s}, \mathbf{q}) - \theta| \geq \sqrt{1-c}$, *then the number of accesses for verifying* $\mathbf{s}$ *is at most* k.

Equivalently, Theorem 8 says that if the true similarity is at least δ off the threshold θ (i.e. $|\mathtt{cos}(\mathbf{s}, \mathbf{q}) - \theta| \geq \delta$ for $\delta > 0$), then it is only necessary to access k entries of $\mathbf{s}$ with the smallest k that satisfies $\sum_{i=1}^{k} \mathbf{s}[i]^2 \geq 1 - \delta^2$. For example, if $\delta = 0.1$ and the first 20 entries of a candidate $\mathbf{s}$ account for >99% of $\sum_{i=1}^{d} \mathbf{s}[i]^2$, then it takes at most 20 accesses for verifying $\mathbf{s}$.

Proof **Case one:** We first consider the case where $\mathtt{cos}(\mathbf{s}, \mathbf{q}) - \theta \geq \sqrt{1-c}$. In this case, we need to show $\sum_{i=1}^{k} \mathbf{s}[i] \cdot \mathbf{q}[i] \geq \theta$. Since $\mathtt{cos}(\mathbf{s}, \mathbf{q}) - \theta \geq \sqrt{1-c}$, we know that

$$\sum_{i=1}^{k} \mathbf{s}[i] \cdot \mathbf{q}[i] + \sum_{i=k+1}^{d} \mathbf{s}[i] \cdot \mathbf{q}[i] \geq \theta + \sqrt{1-c}\,,$$

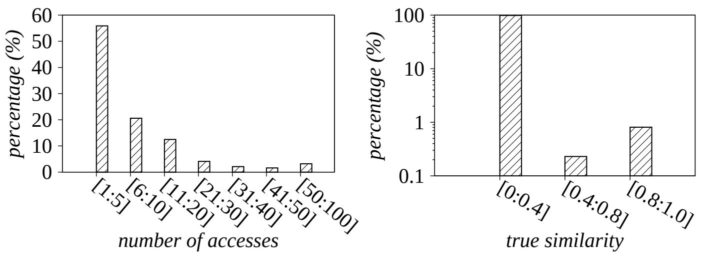
so it suffices to show that $\sum_{i=k+1}^{d} \mathbf{s}[i] \cdot \mathbf{q}[i] \leq \sqrt{1-c}$. This can be obtained by

$$\begin{aligned}\sum_{i=k+1}^{d} \mathbf{s}[i] \cdot \mathbf{q}[i] &\leq \sqrt{\sum_{i=k+1}^{d} \mathbf{s}[i]^2} \cdot \sqrt{\sum_{i=k+1}^{d} \mathbf{q}[i]^2} = \sqrt{1 - \sum_{i=1}^{k} \mathbf{s}[i]^2} \cdot \sqrt{1 - \sum_{i=1}^{k} \mathbf{q}[i]^2} \\ &\leq \sqrt{1-c}\,.\end{aligned}$$

Case two: We then consider the case where $\mathtt{cos}(\mathbf{s}, \mathbf{q}) - \theta \leq -\sqrt{1-c}$. In this case, we need to show

$$\sum_{i=1}^{k} \mathbf{s}[i] \cdot \mathbf{q}[i] + \sqrt{1 - \sum_{i=1}^{k} \mathbf{s}[i]^2} \cdot \sqrt{1 - \sum_{i=1}^{k} \mathbf{q}[i]^2} \leq \theta\,.$$

The first term of the LHS is bounded by $\sum_{i=1}^{d} \mathbf{s}[i] \cdot \mathbf{q}[i] \leq \theta - \sqrt{1-c}$. The second term of the LHS is bounded by $\sqrt{1 - \sum_{i=1}^{k} \mathbf{s}[i]^2} \leq \sqrt{1-c}$. Adding together, the LHS is bounded by θ. This completes the proof of Theorem 8. □



(a) number of accesses (b) true similarities

Fig. 13 The distribution of number of accesses and true similarities

Example 3 Figure 13a shows the distribution of the number of accesses in a query over the mass spectrometry dataset aforementioned. The query is randomly sampled from a real workload with a query vector of 100 non-zero values and $\theta = 0.6$. The result shows that for most candidates, the number of accesses is much smaller than 100. In particular, 55.9% candidates need less than five accesses and 93.1% candidates need less than 30 accesses. This is because as shown in Figure 13b, only 0.23% of candidates have true similarity within ± 0.2 compared to θ (the range [0.4, 0.8]). The rest of the candidates, according to Theorem 8, can be verified in a small number of steps (i.e., ≤ 20).

6 Applying the proposed techniques to top-k cosine queries

Finally, we briefly introduce how to apply the proposed stopping condition and traversal strategies to *top-k* cosine queries. In the top-k setting, each query is a pair $(\mathbf{q}, k)$ where $\mathbf{q}$ is a query vector and k is an integer parameter. A query $(\mathbf{q}, k)$ asks for the top k database vectors with the highest cosine similarity with the query vector $\mathbf{q}$.

Top-k cosine queries can be processed using the original algorithmic framework of TA with a slightly modified version of the stopping condition and traversal strategy. The algorithm retains the same indexing data structures with the 1-d inverted list and convex hulls. At query time, the algorithm traverses the inverted lists as follows. At each snapshot, instead of keeping track of the original candidate set, we keep track of the k vectors with the top-k highest similarity among the gathered vectors.

Note that to guarantee tightness, the exact similarity scores need to be computed online for each gathered candidate, which results in additional computation cost compared to the threshold version of the problem. To decide whether the traversal can be terminated, one can adapt the stopping condition as follows:

Theorem 9 *At a position vector* $\mathbf{b}$*, let* θ_k *be the* k*-th highest similarity score with the query vector* $\mathbf{q}$ *among vectors on or above* $\mathbf{b}$*. Then the following stopping condition is tight and complete:*

$$\varphi_{\mathit{top}\text{-}k}(\mathbf{b}) = \Big(\mathsf{MS}(L[\mathbf{b}]) \le \theta_k\Big).$$

Completeness is straightforward as $\mathsf{MS}(L[\mathbf{b}])$ equals to the largest similarity of the unseen vectors. If $\mathsf{MS}(L[\mathbf{b}]) \le \theta_k$ then it is guaranteed that no new unseen vectors can make to the top-k list. The proof of tightness follows the same proof to Theorem 3 by replacing θ with θ_k.

In terms of running time, the score $\mathsf{MS}(L[\mathbf{b}])$ can be computed using the same $\mathcal{O}(\log d)$ incremental maintenance algorithm in Section 3.2. The lower bound θ_k needs to be updated when a new candidate is gathered. Computing the similarity score takes $\mathcal{O}(d)$ time and updating θ_k can be done in $\mathcal{O}(\log k)$ using a binary heap.

The hull-based traversal strategy for inner product threshold queries can be directly applied to top-k inner product queries without any change. The same near-optimality result holds by following the same analysis. This is because although the threshold θ_k gets updated during the traversal, the final θ_k is fixed for a given query. In addition, the execution of the hull-based strategy for inner product queries is agnostic to the threshold. So processing the top-k query is the same as the processing a threshold query with the same query vector and threshold equal to the final θ_k.

Theorem 10 *For a top-k inner product query* $(\mathbf{q}, k)$*, the access cost of the hull-based traversal strategy* $\mathcal{T}_{\mathsf{HL}}$ *on a near-convex database* $\mathcal{D}$ *is at most* $\mathsf{OPT} + c$ *where* c *is the convexity constant of* $\mathcal{D}$*.*

For top-k cosine queries, the hull-based traversal strategy is also applicable but under minor changes. Recall that for cosine threshold queries, the hull-based traversal strategy operates on the convex hulls of a decomposable approximation $\tilde{F}$ where $\tilde{F}$ requires a pre-selected constant $\tilde{\tau}$ in order to compute the convex hulls. The choice of the right $\tilde{\tau}$ is easy for threshold queries but not obvious for top-k queries. As discussed in Section 4.4, $\tilde{\tau}$ can be set to $1/\theta$ for cosine threshold queries. However, for top-k queries, since the final threshold θ_k is unknown in advance, one might need to estimate the final θ_k and set $\tilde{\tau}$ accordingly. In a practical implementation, to avoid a bad choice of $\tilde{\tau}$, the constant $\tilde{\tau}$ can be made query-dependent or even dynamic during query execution. We leave these aspects for future work.

7 Related work

In this section, we summarize the techniques for cosine similarity search and for the closely related problems such as nearest neighbor search and maximum inner product search in Section 7.1 to 7.3. We also highlight the comparison and compatibility of our proposed techniques with these previous approaches. Moreover, there are related techniques specific to each domain such as keyword search in databases and mass spectrometry search. We summarize them in Section 7.4 and Section 7.5 respectively.

7.1 Cosine similarity search

The cosine threshold querying problem studied in this work is a special case of the *cosine similarity search* (CSS) problem [10,4,5] mentioned in Section 1. We first survey the techniques developed for CSS.

LSH. A widely used technique for cosine similarity search is locality-sensitive hashing (LSH) [63,6,38,41,71]. The main idea of LSH is to partition the whole database into buckets using a series of hash functions such that similar vectors have high probability to be in the same bucket. However, LSH is designed for *approximate* query processing, meaning that it is not guaranteed to return all the true results. In contrast, this work focuses on exact query processing which returns all the results.

TA-family algorithms. Another technique for cosine similarity search is the family of TA-like algorithms. Those algorithms were originally designed for processing top-k ranking queries that find the top k objects ranked according to an aggregation function (see [39] for a survey). We have summarized the classic TA algorithm [30], presented a baseline algorithm inspired by it, and explained its shortcomings in Section 1. The Gathering-Verification framework introduced in Section 2 captures the typical structure of the TA-family when applied to our setting.

The variants of TA (e.g., [35,9,23,15]) can have poor or no performance guarantee for cosine threshold queries since they do not fully leverage the data skewness and the unit vector condition. For example, Güntzer et al. developed Quick-Combine [35]. Instead of accessing all the lists in a lockstep strategy, it relies on a heuristic traversal strategy to access the list with the highest rate of changes to the ranking function in a fixed number of steps ahead. It was shown in [31] that the algorithm is not instance optimal. Although the hull-based traversal strategy proposed in this paper roughly follows the same idea, the number of steps to look ahead is variable and determined by the next convex hull vertex. Thus, for decomposable functions, the hull-based strategy makes globally optimal decisions and is near-optimal under the near-convexity assumption, while Quick-Combine has no performance guarantee because of the fixed step size even when the data is near-convex.

Bast et al. studied top-k ranked query processing with a different goal of minimizing the overall cost by scheduling the best sorted accesses and random accesses [9]. However, when the number of dimensions is high, it requires a large amount of online computations to frequently solve a Knapsack problem, which can be slow in practice. Note that, [9] only evaluated the number of sorted and random accesses in the experiments instead of the wall-clock time. Besides, it does not provide any theoretical guarantee, which is also applicable to [43]. Akbarinia et al. proposed BPA to improve TA, but the optimality ratio is the same as TA in the worst case [3]. Deshpande et al. solved a special case of top-k problem by assuming that the attributes are drawn from a small value space [23]. Zhang et al. developed an algorithm targeting for a large number of lists [86] by merging lists into groups and then apply TA. However, it requires the ranking function to be distributive that does not hold for the cosine similarity function. Yu et al. solved top-k query processing in subspaces [85] that are not applicable to general cosine threshold queries.

Some works considered non-monotonic ranking functions [87,83]. For example, [87] focused on the combination of a boolean condition with a regular ranking function and [83] assumed the ranking functions are lower-bounded.

COORD. Teflioudi et al. proposed the COORD algorithm based on inverted lists for CSS [73,72]. The main idea is to scan the whole lists but with an optimization to prune irrelevant entries using upper/lower bounds of the cosine similarity with the query. Thus, instead of traversing the whole lists starting from the top, it scans only those entries within a feasible range. We can also apply such a pruning strategy to the Gathering-Verification framework by starting the gathering phase at the top of the feasible range. However, there is no optimality guarantee of the algorithm. Also the optimization only works for high thresholds (e.g., 0.95), which can be too restricted in practice. For example, a common and well-accepted threshold in mass spectrometry search is 0.6, which is a medium-sized threshold, making the effect of the pruning negligible.

Partial verification. Anastasiu and Karypis proposed a technique for fast verification of θ-similarity between two vectors [4] without a full scan of the two vectors. We apply the same optimization to the verification phase of the Gathering-Verification framework (Section 5). Additionally, we prove that it has a novel near-constant performance guarantee in the presence of data skewness.

CSS in Hamming Space. [28] and [62] studied cosine similarity search in the Hamming space where each vector contains only binary values. They proposed to store the binary vectors efficiently into multiple hash tables. However, the techniques proposed there cannot be applied since transforming real-valued vectors to binary vectors loses information, and thus correctness is not guaranteed.

Other variants. There are several studies focusing on cosine similarity join to find out all pairs of vectors from the database such that their similarity exceeds a given threshold [10,4,5]. The main idea is to build inverted lists and prune the vectors that are impossible to be similar to any of the vectors in the database in order to limit the search space. However, this work is different since the focus is comparing to a given query vector **q** rather than join. As a result, the techniques in [10,4,5] are not directly applicable: (1) The inverted index is built online instead of offline, meaning that at least one full scan of the whole data is required, which is inefficient for search.[10] (2) The index in [10,4,5] is built for a fixed query threshold, meaning that the index cannot be used for answering arbitrary query thresholds as concerned in this work. The theoretical aspects of similarity join were discussed recently in [2,38].

7.2 Euclidean distance threshold queries

The cosine threshold queries can also be answered by techniques for distance threshold queries (the threshold variant of nearest neighbor search) in Euclidean space. This is because there is a one-to-one mapping between the cosine similarity θ and the Euclidean distance r for unit vectors, i.e., $r = 2\sin(\arccos(\theta)/2)$. Thus, finding vectors that are θ-similar to a query vector is equivalent to finding the vectors whose

[10] The linear scan is feasible for join because the naive approach of join is quadratic.

Euclidean distance is within r. Next, we review exact approaches for distance queries while leaving the discussion of approximate approaches later on. There are four main types of techniques for exact approaches: tree-based indexing, pivot-based indexing, clustering, and dimensionality reduction.

Tree-based indexing. Several tree-based indexing techniques (such as R-tree, KD-tree, Cover-tree [11]) were developed for range queries (so they can also be applied to distance queries), see [12] for a survey. However, they are not scalable to high dimensions (say thousands of dimensions as studied in this work) due to the well known dimensionality curse issue [81].

Pivot-based indexing. The main idea is to pre-compute the distances between data vectors and a set of selected pivot vectors. Then during query processing, use triangle inequalities to prune irrelevant vectors [17,37]. However, it does not scale in high-dimensional space as shown in [17] since it requires a large space to store the pre-computed distances.

Clustering-based (or partitioning-based) methods. The main idea of clustering is to partition the database vectors into smaller clusters of vectors during indexing. Then during query processing, irrelevant clusters are pruned via the triangle inequality [66, 65]. Clustering is an optimization orthogonal to the proposed techniques, as they can be used to process vectors within each cluster to speed up the overall performance.

Dimensionality reduction. Since many techniques are affected negatively by the large number of dimensions, one potential solution is to apply dimensionality reduction (e.g. PCA, Johnson-Lindenstrauss) [58,57,44] before any search algorithm. However, this does not help much if the dimensions are not correlated and there are no hidden latent variables to be uncovered by dimensionality reduction. For example, in the mass spectrometry domain, each dimension represents a chemical compound/element and there is no physics justifying a correlation. We applied dimensionality reduction to the data vectors and turned out that only 4.3% of dimensions can be removed in order to preserve 99% of the distance.

Approximate approaches. Besides LSH, there are many other approximate approaches for high-dimensional similarity search, e.g., graph-based methods [33,26,82], product quantization [42,7,52], randomized KD-trees [69], priority search k-means tree[60], rank cover tree [36], randomized algorithms [80], HD-index [8], and clustering-based methods[53,19].

7.3 Inner product search

The cosine threshold querying is also related to inner product search where vectors may not be unit vectors, otherwise, inner product search is equivalent to cosine similarity search.

Teflioudi et al. proposed the LEMP framework [73,72] to solve the inner product search where each vector may not be normalized. The main idea is to partition the vectors into buckets according to vector lengths and then apply an existing cosine similarity search (CSS) algorithm to each bucket. This work provides an efficient way for CSS that can be integrated to the LEMP framework.

Li et al. developed an algorithm FEXIPRO [54] for inner product search in recommender systems where vectors might contain negative values. Since we focus on non-negative values in this work, the proposed techniques (such as length-based filtering and monotonicity reduction) are not directly applicable.

There are also tree-based indexing and techniques for inner product search [64, 20], but they are not scalable to high dimensions [45]. Another line of research is to leverage machine learning to solve the problem [61, 32, 68]. However, they do not provide accurate answers. Besides, they do not have any theoretical guarantee.

7.4 Keyword search

This work is different from keyword search although the similarity function is (a variant of) the cosine function and the main technique is inverted index [59]. There are two main differences: (1) keyword search generally involves a few query terms [77]; (2) keyword search tends to return Web pages that contain all the query terms [59]. As a result, search engines (e.g., Apache Lucene) mainly sort the inverted lists by document IDs [14, 24, 16] to facilitate boolean intersection. Thus, those algorithms cannot be directly applied to cosine threshold queries. Although there are cases where the inverted lists are sorted by document frequency (or score in general) that are similar to this work, they usually follow TA [24]. This work significantly improves TA for cosine threshold queries.

7.5 Mass spectrometry search

For mass spectrometry search, the state-of-the-art approach is to partition the spectrum database into buckets based on the spectrum mass (i.e., molecular weight) and only search the buckets that have the similar mass to the query [46]. Each bucket is scanned linearly to obtain the results. The techniques proposed in this work can be applied to each bucket and improve the performance dramatically.

Library search, wherein a spectrum is compared to a series of reference spectra to find the most similar match, providing a putative identification for the query spectrum [84]. Methods in library search often use cosine [49] or cosine-derived [78] functions to compute spectrum similarity. Many of the state-of-the-art library search algorithms, e.g., SpectraST [49], Pepitome [21], X!Hunter [18], and [79] find candidates by doing a linear scan of peptides in the library that are within a given parent mass tolerance and computes a distance function against relevant spectra. This is likely because the libraries that are searched against have been small, but spectral libraries are getting larger, e.g., MassIVE-KB now has more than 2.1 million precursors. This work fundamentally improves on these, as it uses inverted lists with many optimizations to significantly reduce the candidate spectra from comparison. M-SPLIT used both a prefiltering and branch and bound technique to reduce the candidate set for each search [75]. This work improves on both these by computing the similar spectra directly, while still applying the filtering.

Database search [29] is where the experimental spectrum is compared to a host of theoretical spectra that are generated from a series of sequences. The theoretical spectra differ from experimental spectra in that they do not have intensities for each peak and contain all peaks which could be in the given peptide fragmentation. In other words, the values in the vectors are not important and the similarity function evaluates the number of shared dimensions that are irrelevant to the values. While the problem is different in practice than that described in this paper, it is still interesting to consider optimizations. Dutta and Chen proposed an LSH-based technique which embeds both the experimental spectra and the theoretical spectra onto a higher dimensional space [27]. While this method performs well at the expense of low recall rate, our method is guaranteed to not lose any potential matches. There are also other methods optimizing this problem by using preindexing [70,46] but none do this kind of preindexing for calculating matches between experimental spectra.

7.6 Others

This work is also different from [55] because that work focused on similarity search based on set-oriented p-norm similarity, which is different from cosine similarity.

In the literature, there are skyline-based approaches [51] to solve a special case of top-k ranked query processing where no ranking function is specified. However, those approaches cannot be used to solve the problems that only involves *unit vectors* such that all the vectors belong to skyline points.

8 Conclusion

In this work, we proposed optimizations to the index-based, TA-like algorithms for answering cosine threshold queries as well as cosine top-k queries, which lie at the core of numerous applications. The novel techniques include a complete and tight stopping condition computable incrementally in $\mathcal{O}(\log d)$ time and a family of convex hull-based traversal strategies with near-optimality guarantees for a larger class of decomposable functions beyond cosine. With these techniques, we show near-optimality first for inner-product threshold queries, then extend the result to the full cosine threshold queries using approximation. These results are significant improvements over a baseline approach inspired by the classic TA algorithm. In addition, we have verified with experiments on real data the assumptions required by the near-optimality results.

Acknowledgement

We thank the anonymous reviewers of ICDT and ToCS for the very constructive and helpful comments. We are also grateful to Victor Vianu who helped us improve the presentation of the paper. This work was supported in part by the National Science Foundation (NSF) under awards BIGDATA 1447943 and ABI 1759980, and by the National Institutes of Health (NIH) under awards P41GM103484 and R24GM127667.

References

1. Ruedi Aebersold and Matthias Mann. Mass-spectrometric exploration of proteome structure and function. *Nature*, 537:347–355, 2016.
2. Thomas Dybdahl Ahle, Rasmus Pagh, Ilya Razenshteyn, and Francesco Silvestri. On the complexity of inner product similarity join. In *PODS*, pages 151–164, 2016.
3. Reza Akbarinia, Esther Pacitti, and Patrick Valduriez. Best position algorithms for top-k queries. In *VLDB*, pages 495–506, 2007.
4. David C. Anastasiu and George Karypis. L2AP: Fast cosine similarity search with prefix L-2 norm bounds. In *ICDE*, pages 784–795, 2014.
5. David C. Anastasiu and George Karypis. PL2AP: Fast parallel cosine similarity search. In *IA3*, pages 8:1–8:8, 2015.
6. Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *NIPS*, pages 1225–1233, 2015.
7. Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. *PVLDB*, 9(4):288–299, 2015.
8. Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. HD-Index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *PVLDB*, 11(8):906–919, 2018.
9. Holger Bast, Debapriyo Majumdar, Ralf Schenkel, Martin Theobald, and Gerhard Weikum. Io-top-k: Index-access optimized top-k query processing. In *VLDB*, pages 475–486, 2006.
10. Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.
11. Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *ICML*, pages 97–104, 2006.
12. Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *CSUR*, 33(3):322–373, 2001.
13. Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
14. Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM*, pages 426–434, 2003.
15. Nicolas Bruno, Luis Gravano, and Amélie Marian. Evaluating top-k queries over web-accessible databases. In *ICDE*, pages 369–380, 2002.
16. Kaushik Chakrabarti, Surajit Chaudhuri, and Venkatesh Ganti. Interval-based pruning for top-k processing over compressed lists. In *ICDE*, pages 709–720, 2011.
17. Lu Chen, Yunjun Gao, Baihua Zheng, Christian S. Jensen, Hanyu Yang, and Keyu Yang. Pivot-based metric indexing. *PVLDB*, 10(10):1058–1069, 2017.
18. R Craig, J C Cortens, D Fenyo, and R C Beavis. Using annotated peptide mass spectrum libraries for protein identification. *Journal of Proteome Research*, 5(8):1843–1849, 2006.
19. Bin Cui, Jiakui Zhao, and Gao Cong. ISIS: A new approach for efficient similarity search in sparse databases. In *DASFAA*, pages 231–245, 2010.
20. Ryan R. Curtin, Alexander G. Gray, and Parikshit Ram. Fast exact max-kernel search. In *SDM*, pages 1–9, 2013.
21. Surendra Dasari, Matthew C Chambers, Misti A Martinez, Kristin L Carpenter, Amy-Joan L Ham, Lorenzo J Vega-Montoto, and David L Tabb. Pepitome: Evaluating improved spectral library search for identification complementarity and quality assessment. *Journal of Proteome Research*, 11(3):1686–95, 2012.
22. Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Introduction*. Springer, 2008.
23. Prasad M Deshpande, Deepak P, and Krishna Kummamuru. Efficient online top-k retrieval with arbitrary similarity measures. In *EDBT*, pages 356–367, 2008.
24. Shuai Ding and Torsten Suel. Faster top-k document retrieval using block-max indexes. In *SIGIR*, pages 993–1002, 2011.
25. Doc2Vec. *https://radimrehurek.com/gensim/models/doc2vec.html*.
26. Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, pages 577–586, 2011.
27. Debojyoti Dutta and Ting Chen. Speeding up tandem mass spectrometry database search: metric embeddings and fast near neighbor search. *Bioinformatics*, 23(5):612–618, 2007.

28. Sepehr Eghbali and Ladan Tahvildari. Cosine similarity search with multi index hashing. *CoRR*, abs/1610.00574, 2016. URL: `http://arxiv.org/abs/1610.00574`.
29. Jimmy K. Eng, Ashley L. McCormack, and John R. Yates. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry*, 5(11):976–989, 1994.
30. Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.
31. Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.
32. Marco Fraccaro, Ulrich Paquet, and Ole Winther. Indexable probabilistic matrix factorization for maximum inner product search. In *AAAI*, pages 1554–1560, 2016.
33. Cong Fu, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with navigating spreading-out graphs. *CoRR*, abs/1707.00143, 2017.
34. Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Info. Pro. Lett.*, 1:132–133, 1972.
35. Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kiebling. Optimizing multi-feature queries for image databases. In *VLDB*, pages 419–428, 2000.
36. Michael E. Houle and Michael Nett. Rank-based similarity search: Reducing the dimensional dependence. *PAMI*, 37(1):136–150, 2015.
37. Vagelis Hristidis, Nick Koudas, and Yannis Papakonstantinou. PREFER: A system for the efficient execution of multi-parametric ranked queries. In *SIGMOD*, pages 259–270, 2001.
38. Xiao Hu, Yufei Tao, and Ke Yi. Output-optimal parallel algorithms for similarity joins. In *PODS*, pages 79–90, 2017.
39. Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *CSUR*, 40(4):1–58, 2008.
40. Img2Vec. *https://github.com/christiansafka/img2vec*.
41. Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *ICDT*, pages 604–613, 1998.
42. Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *PAMI*, 33(1):117–128, 2011.
43. Wen Jin and Jignesh M. Patel. Efficient and generic evaluation of ranked queries. In *SIGMOD*, pages 601–612, 2011.
44. William B. Johnson, Joram Lindenstrauss, and Gideon Schechtman. Extensions of lipschitz maps into banach spaces. *Israel Journal of Mathematics*, 54(2):129–138, 1986.
45. Omid Keivani, Kaushik Sinha, and Parikshit Ram. Improved maximum inner product search with better theoretical guarantees. In *IJCNN*, pages 2927–2934, 2017.
46. Andy T. Kong, Felipe V. Leprevost, Dmitry M. Avtonomov, Dattatreya Mellacheruvu, and Alexey I. Nesvizhskii. Msfragger: Ultrafast and comprehensive peptide identification in mass spectrometry-based proteomics. *Nature Methods*, 14:513–520, 2017.
47. Harold W Kuhn and Albert W Tucker. Nonlinear programming. In *Traces and Emergence of Nonlinear Programming*, pages 247–258. 2014.
48. B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137, 2009.
49. Henry Lam, Eric W. Deutsch, James S. Eddes, Jimmy K. Eng, Nichole King, Stephen E. Stein, and Ruedi Aebersold. Development and validation of a spectral library searching method for peptide identification from ms/ms. *Proteomics*, 7(5), 2007.
50. Erik Learned-Miller, Gary B Huang, Aruni RoyChowdhury, Haoxiang Li, and Gang Hua. Labeled faces in the wild: A survey. In *Advances in face detection and facial image analysis*, pages 189–248. Springer, 2016.
51. Jongwuk Lee, Hyunsouk Cho, and Seung-won Hwang. Efficient dual-resolution layer indexing for top-k queries. In *ICDE*, pages 1084–1095, 2012.
52. Victor Lempitsky. The inverted multi-index. In *CVPR*, pages 3069–3076, 2012.
53. Chen Li, Edward Chang, Hector Garcia-Molina, and Gio Wiederhold. Clustering for approximate similarity search in high-dimensional spaces. *TKDE*, 14(4):792–808, 2002.
54. Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. Fexipro: Fast and exact inner product retrieval in recommender systems. In *SIGMOD*, pages 835–850, 2017.
55. Wenhai Li, Lingfeng Deng, Yang Li, and Chen Li. Zigzag: Supporting similarity queries on vector space models. In *SIGMOD*, 2018.

56. Yuliang Li, Jianguo Wang, Benjamin Pullman, Nuno Bandeira, and Yannis Papakonstantinou. Index-Based, High-Dimensional, Cosine Threshold Querying with Optimality Guarantees. In *ICDT*, volume 127, pages 11:1–11:20, 2019.
57. Xiang Lian and Lei Chen. General cost models for evaluating dimensionality reduction in high-dimensional spaces. *TKDE*, 21(10):1447–1460, 2009.
58. Yi-Ching Liaw, Maw-Lin Leou, and Chien-Min Wu. Fast exact k nearest neighbors search using an orthogonal search tree. *PR*, 43(6):2351–2358, 2010.
59. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
60. Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *PAMI*, 36(11):2227–2240, 2014.
61. Stephen Mussmann and Stefano Ermon. Learning and inference via maximum inner product search. In *ICML*, pages 2587–2596, 2016.
62. Jianbin Qin, Yaoshu Wang, Chuan Xiao, Wei Wang, Xuemin Lin, and Yoshiharu Ishikawa. GPH: Similarity search in hamming space. In *ICDE*, 2018.
63. Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.
64. Parikshit Ram and Alexander G. Gray. Maximum inner-product search using cone trees. In *SIGKDD*, pages 931–939, 2012.
65. Sharadh Ramaswamy and Kenneth Rose. Adaptive cluster distance bounding for high-dimensional indexing. *TKDE*, 23(6):815–830, 2011.
66. Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.
67. John E Savage. Models of computation–exploring the power of computing. 1998.
68. Fumin Shen, Wei Liu, Shaoting Zhang, Yang Yang, and Heng Tao Shen. Learning binary codes for maximum inner product search. In *ICCV*, pages 4148–4156, 2015.
69. Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In *CVPR*, pages 1–8, 2008.
70. Wilfred H. Tang, Benjamin R. Halpern, Ignat V. Shilov, Sean L. Seymour, Sean P. Keating, Alex Loboda, Alpesh A. Patel, Daniel A. Schaeffer, and Lydia M. Nuwaysir. Discovering known and unanticipated protein modifications using ms/ms database searching. *Analytical Chemistry*, 77(13):3931–3946, 2005.
71. Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009.
72. Christina Teflioudi and Rainer Gemulla. Exact and approximate maximum inner product search with lemp. *TODS*, 42(1):5:1–5:49, 2016.
73. Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. Lemp: Fast retrieval of large entries in a matrix product. In *SIGMOD*, pages 107–122, 2015.
74. The Booking.com Dataset. *https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe*.
75. Jian Wang, Josué Pérez-Santiago, Jonathan E Katz, Parag Mallick, and Nuno Bandeira. Peptide identification from mixture tandem mass spectra. *MCP*, 9(7):1476–1485, 2010.
76. Jianguo Wang. *Query Processing of Sorted Lists on Modern Hardware*. University of California, San Diego, 2019.
77. Jianguo Wang, Chunbin Lin, Ruining He, Moojin Chae, Yannis Papakonstantinou, and Steven Swanson. MILC: Inverted list compression in memory. *PVLDB*, 10(8):853–864, 2017.
78. Mingxun Wang and Nuno Bandeira. Spectral library generating function for assessing spectrum-spectrum match significance. *Journal of Proteome Research*, 12(9):3944–3951, 2013.
79. Mingxun Wang, Jeremy J. Carver, and Nuno Bandeira. Sharing and community curation of mass spectrometry data with global natural products social molecular networking. *Nature Biotechnology*, 34(8):828–837, 2016.
80. Yiqiu Wang, Anshumali Shrivastava, Jonathan Wang, and Junghee Ryu. Randomized algorithms accelerated over cpu-gpu for ultra-high dimensional similarity search. In *SIGMOD*, 2018.
81. Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.
82. Yubao Wu, Ruoming Jin, and Xiang Zhang. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *SIGMOD*, pages 1139–1150, 2014.
83. Dong Xin, Jiawei Han, and Kevin C. Chang. Progressive and selective merge: Computing top-k with ad-hoc ranking functions. In *SIGMOD*, pages 103–114, 2007.

84. John R. Yates, Scott F. Morgan, Christine L. Gatlin, Patrick R. Griffin, and Jimmy K. Eng. Method to compare collision-induced dissociation spectra of peptides: Potential for library searching and subtractive analysis. *Analytical Chemistry*, 70(17):3557–3565, 1998.
85. Albert Yu, Pankaj K. Agarwal, and Jun Yang. Top-k preferences in high dimensions. In *ICDE*, pages 748–759, 2014.
86. Shile Zhang, Chao Sun, and Zhenying He. Listmerge: Accelerating top-k aggregation queries over large number of lists. In *DASFAA*, pages 67–81, 2016.
87. Zhen Zhang, Seung-won Hwang, Kevin Chen-Chuan Chang, Min Wang, Christian A. Lang, and Yuan-chi Chang. Boolean + ranking: Querying a database by k-constrained optimization. In *SIGMOD*, pages 359–370, 2006.