View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

Index-Based, High-Dimensional, Cosine Threshold Querying with Optimality Guarantees

Yuliang Li
Megagon Labs, Mountain View, California, USA
UC San Diego, San Diego, California, USA

Jianguo Wang
UC San Diego, San Diego, California, USA

Benjamin Pullman
UC San Diego, San Diego, California, USA

Nuno Bandeira
UC San Diego, San Diego, California, USA

Yannis Papakonstantinou
UC San Diego, San Diego, California, USA

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. These queries arise naturally in many applications, such as document retrieval, image search, and mass spectrometry. The present paper considers the efficient evaluation of such queries, providing novel optimality guarantees and exhibiting 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 against the similarity threshold. However, directly applying TA in its raw form misses significant optimization opportunities. Indeed, we 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 one can take advantage of data skewness to obtain better traversal strategies. In particular, we show a novel traversal strategy that exploits a common data skewness condition which holds in multiple domains including mass spectrometry, documents, and image databases. 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.

2012 ACM Subject Classification Theory of computation → Data structures and algorithms for data management; Theory of computation → Database query processing and optimization (theory); Information systems → Nearest-neighbor search

Keywords and phrases Vector databases, Similarity search, Cosine, Threshold Algorithm

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.11

Related Version A full version of the paper is available at `https://arxiv.org/abs/1812.07695`.

Acknowledgements We are very grateful to Victor Vianu who helped us significantly improve the presentation of the paper. We also thank the anonymous reviewers for the very constructive and helpful comments. 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.

© Yuliang Li, Jianguo Wang, Benjamin Pullman, Nuno Bandeira, and Yannis Papakonstantinou;
licensed under Creative Commons License CC-BY
22nd International Conference on Database Theory (ICDT 2019).
Editors: Pablo Barcelo and Marco Calautti; Article No. 11; pp. 11:1–11:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

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.

This problem arises in many applications including document retrieval [11], image search [24], recommender systems [26] and mass spectrometry. For example, in mass spectrometry, billions of spectra are generated for the purpose of protein analysis [1, 25, 33]. 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 ∼2000 dimensions where ∼100 coordinates are non-zero.

Cosine threshold queries play an important role in analyzing such 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.*

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).

In the literature, cosine threshold querying is a special case of Cosine Similarity Search (CSS) [31, 3, 26], where other aspects like approximate answers, top-k queries and similarity join are considered. Our work considers specifically CSS with exact, threshold and single-vector queries, which is the case of interest to many applications.

Because of the unit-vector assumption, the scoring function cos computes the dot product $\mathbf{q} \cdot \mathbf{s}$. Without the unit-vector assumption, the problem is equivalent to *inner product threshold querying*, which is of interest in its own right. Related work on cosine and inner product similarity search is summarized in Section 5.

In this paper we develop novel techniques for the efficient evaluation of cosine threshold queries. We take as a starting point the well-known Threshold Algorithm (TA), by Fagin et al. [16], because of its simplicity, wide applicability, and optimality guarantees. A review of the classic TA is provided in the full version of the paper.

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 $\texttt{cos}(\mathbf{q}, \mathbf{s})$, 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.

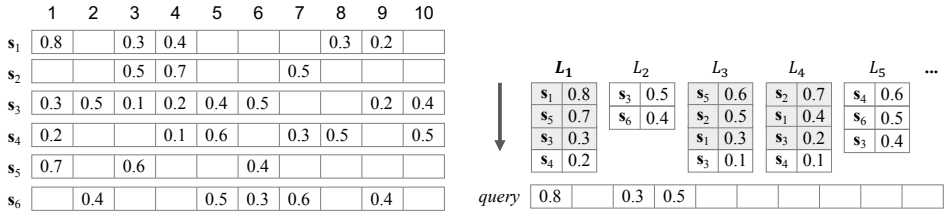


Figure 1 An example of cosine threshold query with six 10-dimensional vectors. The missing values are 0's. We only need to scan the lists L_1, L_3, and L_4 since the query vector has non-zero values in dimension 1, 3 and 4. For $\theta = 0.6$, the gathering phase terminates after each list has examined 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.

A *baseline* index and algorithm inspired by TA can answer cosine threshold queries exactly without a full scan of the vector database for each query. In addition, the baseline algorithm enjoys the same instance optimality guarantee as the original TA. This baseline is created as follows. First, 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 $b = 0$.) 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 $j < b$ 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 k dimensions that have non-zero values in the query vector $\mathbf{q}$ should participate in query processing – this leads to a $k \cdot \mathsf{OPT}$ guarantee for inner product queries.[2] But even this guarantee loses its practical value when k is a large

[1] There is no need to include pairs with zero values in the list.
[2] This optimization is equally applicable to the TA's problem: Scan only the lists that correspond to

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	$\mathsf{OPT}(\theta - \epsilon) + c$

number. In the mass spectrometry scenario k is $\sim$100. In document similarity and image similarity cases it is even higher.

For cosine threshold queries, the $k \cdot \mathsf{OPT}$ 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 Appendix C of the full version).[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 lightweight additions to the baseline index.
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 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 7). In contrast, the stopping condition of the (TA-inspired) baseline is complete but not tight (Theorem 27 in the full version). The proposed stopping condition takes into account that all database vectors

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.

are normalized and reduces the problem to solving a special quadratic program (Equation 1) that guarantees both completeness and tightness. While the new stopping condition prunes further the set of candidates, it can also be efficiently computed in $\mathcal{O}(\log d)$ time using incremental maintenance techniques.

- We introduce the *hull-based* traversal strategies that exploit the skewness of the data (Section 4). In particular, skewness implies 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 16 and Corollary 18), where OPT is the number of accesses by the optimal strategy and c is the max distance between two vertices in the lower convex hull. Experiments show that in multiple real-world cases, c is a very small fraction of OPT.
- 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 20) 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 adjustment ϵ is very small in practice, e.g., 0.1. We summarize these new results in Table 1.

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 as we mentioned above. Finally, we discuss related work in Section 5 and conclude in Section 6.

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 [11]) 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 $\{(\mathsf{ref}(\mathbf{s}), \mathbf{s}[i]) \mid \mathbf{s} \in \mathcal{D} \wedge \mathbf{s}[i] > 0\}$ sorted in descending order of $\mathbf{s}[i]$ 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. In the interest of brevity, we will often write $(\mathbf{s}, \mathbf{s}[i])$ instead of $(\mathsf{ref}(\mathbf{s}), \mathbf{s}[i])$. As an example in Figure 1, the list L_1 is built for the first 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.

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

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

Table 2 Notation.

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

Algorithm 1: Gathering-Verification Framework.

```
   input  : (D, {L_i}_{1≤i≤d}, q, θ)
   output : R the set of θ-similar vectors
   /* Gathering phase                                   */
1  Initialize b = (b_1, ..., b_d) = (0, ..., 0);
   // φ(·) is the stopping condition
2  while φ(b) = false do
       // T(·) is the traversal strategy to
          determine which list to access
          next
3      i ← T(b);
4      b_i ← b_i + 1;
5      Put the vector s in L_i[b_i] to the candidate
         pool C;
   /* Verification phase                                */
6  R ← {s | s ∈ C ∧ cos(q, s) ≥ θ};
7  return 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 2.** *Let $\mathbf{b}$ be a position vector on the inverted index $\{L_i\}_{1\le i\le d}$ of a database $\mathcal{D}$. The partial observation at $\mathbf{b}$, denoted as $\mathcal{L}(\mathbf{b})$, is a collection of lists $\{\hat{L}_i\}_{1\le i\le d}$ where for every $1\le i\le d$, $\hat{L}_i=[L_i[1],\ldots,L_i[b_i]]$.*

▶ **Definition 3.** *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 $\mathtt{cos}(\mathbf{q},\mathbf{s})\ge\theta$ by accessing the database. Various

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

techniques [31, 4, 26] 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 Appendix B. Additionally, as a novel contribution, we show that in the presence of data skewness, partial verification can have a near-constant performance guarantee (Theorem 25 of the full version) for each candidate.

▶ **Theorem 4** (Informal). *For most skewed vectors, θ-similarity can be computed at constant time.*

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. This reason behind 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.

Remark on the suitability of TA-like algorithms. One may wonder whether algorithms that start the gathering phase NOT from the top of the inverted lists may outperform the best TA-like algorithm. In particular, it appears tempting to start the gathering phase from the point closest to q_i in each inverted list and traverse towards the two ends of each list. Appendix E of the full version proves why this idea can lead to poor performance. In particular, we prove that in a general setting, the computation of a tight and complete stopping condition (formally defined in Definition 5 and 6) becomes NP-HARD since it needs to take into account constraints from two pointers (forward and backward) for each inverted list. Furthermore, in many applications, the data skewing leads to small savings from pruning the top area of each list, since the top area is sparsely populated - unlike the densely populated bottom area of each list. Thus it is not justified to use an expensive gathering phase algorithm for small savings.

Section 5.1 reviews additional prior work ideas [31, 32] that avoid traversing some top/bottom regions of the inverted index. Such ideas may provide additional optimizations to TA-like algorithms in variations and/or restrictions of the problem (e.g., a restriction that the threshold is very high) and thus they present future work opportunities in closely related problems.

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 $\{L_i\}_{1\leq i\leq d}$ 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 $\mathbf{b}$, $\varphi(\mathbf{b}) = \texttt{True}$.*

It is desirable that a stopping condition be both complete and tight. However, as we shown in Appendix C of the full version, 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 $\mathbf{s}\cdot\mathbf{q} \geq \theta$. 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:

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

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 [10]. We address this challenge by deriving an equivalently strong stopping condition that guarantees tightness and is efficiently testable:

▶ **Theorem 7.** *Let τ be the solution of the equation $\sum_{i=1}^{d} \min\{q_i\cdot\tau, L_i[b_i]\}^2 = 1$ and*

$$\mathsf{MS}(L[\mathbf{b}]) = \sum_{i=1}^{d} \min\{q_i\cdot\tau, L_i[b_i]\}\cdot q_i \tag{2}$$

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

Proof. The tight and complete stopping condition is obtained by applying the Karush-Kuhn-Tucker (KKT) conditions [23] for solving nonlinear programs. We first formulate the set of constraints in (1) as an optimization problem over $\mathbf{s}$:

$$\text{maximize}\quad \sum_{i=1}^{d} s_i\cdot q_i \qquad \text{subject to}\quad \sum_{i=1}^{d} s_i^2 = 1 \quad \text{and} \quad s_i \leq L_i[b_i], \quad \forall i \in [d] \tag{3}$$

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

▶ **Lemma 8** (derived from KKT). *The optimal* $\mathbf{s}$ *in (3) satisfies the following conditions:*

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

in addition to the constraints in (3) (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 $s_i = L_i[b_i]$. For the remaining dimensions, 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.

Testing the tight and complete condition φ_{TC} requires solving τ in Theorem (7), for which a direct application of the bisection method takes $\mathcal{O}(d)$ time. We show a novel efficient algorithm (Appendix D) in the full version of the paper based on incremental maintenance which takes only $\mathcal{O}(\log d)$ time for each test of φ_{TC}.

▶ **Theorem 9.** *The stopping condition* $\varphi_{\mathsf{TC}}(\mathbf{b})$ *can be incrementally computed in* $\mathcal{O}(\log d)$ *time.*

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 10** (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 $\varphi_{\mathsf{TC}}(\mathbf{b}_k) = \texttt{True}$. Note that $\mathsf{cost}(\mathcal{T})$ also equals $|\mathbf{b}_k|$.*

▶ **Definition 11** (Instance Optimality). *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 $\varphi_{\mathsf{TC}}(\mathbf{b}) = \texttt{True}$. 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 18) 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 16).

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 20). 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 12** (Decomposable Function). *A decomposable function $F(\mathbf{s})$ is a d-dimensional real 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 13** (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. 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

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

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.

It is obvious that there are instances where the max-reduction strategy can be far from optimal, but 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.

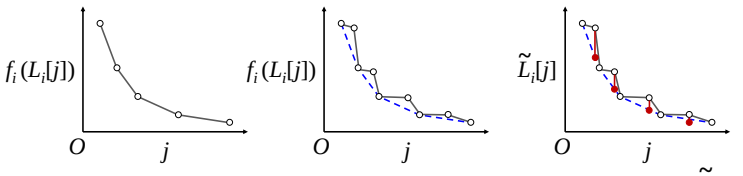
▶ **Assumption 1** (Ideal Convexity). *For every inverted list L_i, let $\Delta_i[j] = f_i(L_i[j]) - f_i(L_i[j+1])$ for $0 \le j < |L_i|$.[6] The list L_i is ideally convex if the sequence Δ_i is non-increasing, i.e., $\Delta_i[j+1] \le \Delta_i[j]$ for every j. Equivalently, the piecewise linear function passing through the points $\{(j, f_i(L_i[j]))\}_{0\le j\le |L_i|}$ 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 2(a). The max-reduction strategy $\mathcal{T}_{\mathsf{MR}}$ is optimal under the ideal convexity assumption:

▶ **Theorem 14** (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.

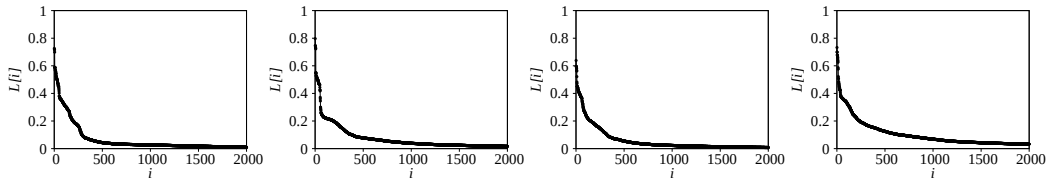
We prove Theorem 14 with a simple greedy argument (see Appendix F for more details): each move of $\mathcal{T}_{\mathsf{MR}}$ always results in the globally maximal reduction in the scoring function as guaranteed by the convexity condition.

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



(a) Convex (b) Near-convex (c) Constructing $\tilde{L}_i[j]$

Figure 2 Convexity and near-convexity.



(a) list L_0 (b) list L_1 (c) list L_2 (d) list L_3

Figure 3 The skewed inverted lists in mass spectrometry.

4.3 The hull-based traversal strategy

Theorem 14 provides a strong performance guarantee but the ideal convexity assumption is usually not true on real datasets. Without the ideal convexity assumption, 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. As a result, the $\mathcal{T}_{\mathsf{MR}}$ strategy has no performance guarantee in general.

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* [14] 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 3, a document dataset, and an image dataset (see Appendix I of the full version for details).

▶ **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 15.** 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 2(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 2(a), the constant is 1 when the decrease between successive values is strictly decelerating.

[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.

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, each sequence $\tilde{\Delta}_i$ is non-decreasing. Then the *hull-based* traversal strategy is simply defined as

$$\mathcal{T}_{\mathsf{HL}}(\mathbf{b}) = \operatorname*{argmax}_{1 \le i \le d} (\tilde{\Delta}_i[b_i]). \tag{4}$$

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 [14], 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 16.** *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 [16].

Proof. Let $\mathcal{B} = \{\mathbf{b}_i\}_{i \ge 0}$ 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 in between every pair of consecutive boundary positions $\mathbf{b}$ and $\mathbf{b}'$ in $\mathcal{B}$, $\mathcal{T}_{\mathsf{HL}}(\mathbf{b})$ will always be the same index. We call the subsequence positions $\{\mathbf{b}_i\}_{l \le i < r}$ 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 17.** *For every boundary position vector $\mathbf{b}$ generated by $\mathcal{T}_{\mathsf{HL}}$, we have $F(L[\mathbf{b}]) \le F(L[\mathbf{b}^*])$ 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. We prove Lemma 17 by generalizing the greedy argument in the proof of Theorem 14. More details can be found in Appendix G of the full version.

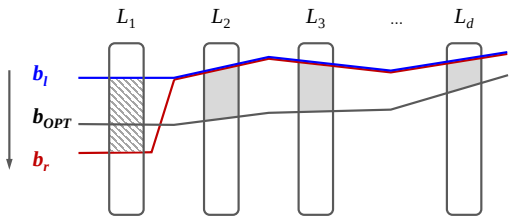


Figure 4 ($\mathbf{b}_l$, $\mathbf{b}_r$): the two boundary positions surrounding the stopping position $\mathbf{b}_{\mathsf{stop}}$ of $\mathcal{T}_{\mathsf{HL}}$; $\mathbf{b}_{\mathsf{OPT}}$: the optimal stopping position; It is guaranteed that (1) $|\mathbf{b}_{\mathsf{stop}}| - |\mathbf{b}_l| < |\mathbf{b}_r| - |\mathbf{b}_l| \leq c$ and (2) $|\mathbf{b}_l| < |\mathbf{b}_{\mathsf{OPT}}|$.

Lemma 17 is sufficient for Theorem 16 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 17, Theorem 16 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 17, $\mathbf{b}_l$ 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 4. ◀

Since the baseline stopping condition φ_{BL} is tight and complete for inner product queries, one immediate implication of Theorem 16 is that

▶ **Corollary 18** (Informal). *The hull-based strategy $\mathcal{T}_{\mathsf{HL}}$ for inner product queries is near-optimal.*

Verifying the assumption. We demonstrate the practical impact of the near-optimality result in real mass spectrometry datasets. The same experiment is repeated on a document and an image dataset (see Appendix I of the full version). 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 16, 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 4). 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[8]. 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. Similar results are obtained in a document and an image dataset, where the gap size takes only 7.9% and 0.4% of the overall access cost respectively.

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

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

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 16 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 19.** *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 $\tilde{F}(\mathbf{s}) - F(\mathbf{s}) \in [-\epsilon_1, \epsilon_2]$ for every vector $\mathbf{s}$.*

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 16:

▶ **Theorem 20.** *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. Recall that $\mathbf{b}_l$ is 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 $F(L[\mathbf{b}_l]) \ge \theta$. It is sufficient to show that for every vector $\mathbf{b}^*$ where $|\mathbf{b}^*| = |\mathbf{b}_l|$, $F(L[\mathbf{b}^*]) \ge \theta - \epsilon_1 - \epsilon_2$ so no traversal can stop within $|\mathbf{b}_l|$ steps, implying that the final access cost is no more than $|\mathbf{b}_l| + c$ which is bounded by $\mathsf{OPT}(\theta - \epsilon_1 - \epsilon_2) + c$.

By Lemma 17, we know that for every such $\mathbf{b}^*$, $\tilde{F}(L[\mathbf{b}^*]) \ge \tilde{F}(L[\mathbf{b}_l])$. By definition of the approximation $\tilde{F}$, we know that $F(L[\mathbf{b}^*]) \ge \tilde{F}(L[\mathbf{b}^*]) - \epsilon_1$ and $\tilde{F}(L[\mathbf{b}_l]) \ge F(L[\mathbf{b}_l]) - \epsilon_2$. Combined together, for every $\mathbf{b}^*$ where $|\mathbf{b}^*| = |\mathbf{b}_l|$, we have

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

This completes the proof of Theorem 20. ◀

Choosing the decomposable approximation. By Theorem 20, 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 (2) of MS, one reasonable choice of $\tilde{F}$ can be obtained by replacing the term τ with a fixed constant $\tilde{\tau}$. Formally, let

$$\tilde{F}(L[\mathbf{b}]) = \sum_{i=1}^{d} \min\{q_i \cdot \tilde{\tau}, L_i[b_i]\} \cdot q_i \tag{5}$$

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 a detailed analysis in Appendix H, we obtain the following upper bound of ϵ:

$$\epsilon \le \max\{0, \tilde{\tau} - 1/\mathsf{MS}(L[\mathbf{b}_l])\} + \mathsf{MS}(L[\mathbf{b}_l]) - \tilde{F}(L[\mathbf{b}_l]). \tag{6}$$

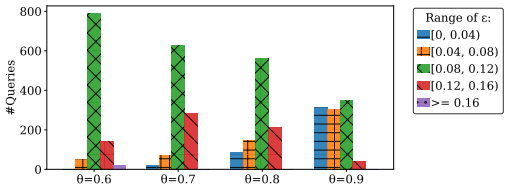


Figure 5 The distribution of ϵ.

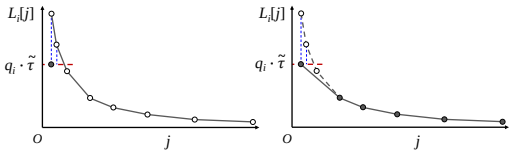


Figure 6 The construction of convex hull $\tilde{\mathsf{H}}_i$.

Verifying the near-optimality. Next, we verify that the above upper bound of ϵ is small in practice. We ran the same set of queries as in Section 4.3 and show the distribution of ϵ's upper bounds in Figure 5. We set $\tilde{\tau} = 1/\theta$ for all queries so the first term of (6) becomes zero. Note that more aggressive pruning 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%. 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.

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 $L_i[b_i] \geq q_i \cdot \tilde{\tau}$, $f_i(L_i[b_i])$ equals a fixed value $q_i^2 \cdot \tilde{\tau}$ otherwise is proportional to $L_i[b_i]$. As illustrated in Figure 6 (left), the list of values $\{f_i(L_i[j])\}_{j \geq 0}$ is essentially obtained by replacing the $L_i[j]$'s greater than $q_i \cdot \tilde{\tau}$ with $q_i \cdot \tilde{\tau}$.

The following can be shown using properties of convex hulls:

▶ **Lemma 21.** *For every $i \in [d]$, the convex hull $\tilde{\mathsf{H}}_i$ is a subset of H_i where an index j_k of H_i is in $\tilde{\mathsf{H}}_i$ iff $k = 1$ or*

$$\big(q_i \cdot \tilde{\tau} - L_i[j_k]\big) \;/\; j_k \geq \big(L_i[j_k] - L_i[j_{k+1}]\big) \;/\; (j_{k+1} - j_k). \tag{7}$$

Lemma 21 provides an efficient way to obtain each convex hull $\tilde{\mathsf{H}}_i$ from the pre-computed H_i's. When a query $\mathbf{q}$ is given, we perform a binary search on each H_i to find the first $j_k \in \mathsf{H}_i$ that satisfies (7). Then $\tilde{\mathsf{H}}_i$ is the set of indices $\{0, j_k, j_{k+1} \dots\}$. We illustrate the construction in Figure 6 (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.

5 Related work

In this section, we present the main related work and defer the additional related work (e.g., dimensionality reduction, mass spectrometry search, inner product queries) to the full version.

5.1 Cosine similarity search

The cosine threshold querying studied in this work is a special case of the *cosine similarity search* (CSS) problem [7, 3, 4] mentioned in Section 1. We first survey the techniques developed for CSS.

LSH. A widely used technique for cosine similarity search is locality-sensitive hash (LSH) [27, 5, 20, 22, 30]. 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 [21] for a survey). We have summarized the classic TA algorithm [16], 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., [18, 6, 15, 12]) 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 [18]. 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 [17] 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.

COORD. Teflioudi et al. proposed the COORD algorithm based on inverted lists for CSS [32, 31]. 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 are not always the requirement. 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 [3] without a full scan of the two vectors. We can apply the same optimization to the verification phase of the Gathering-Verification framework. Additionally, we prove that it has a novel near-constant performance guarantee in the presence of data skewness.

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 [7, 3, 4]. However, this work is different since the focus is comparing to a given query vector $\mathbf{q}$ rather than join. As a result, the techniques in [7, 3, 4] 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. (2) The index in [7, 3, 4] 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, 20].

5.2 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 Euclidean distance is within r. Next, we review exact approaches for distance queries while leaving the discussion of approximate approaches in the full version.

Tree-based indexing. Several tree-based indexing techniques (such as R-tree, KD-tree, Cover-tree [8]) were developed for range queries (so they can also be applied to distance queries), see [9] 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 [34].

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 [13, 19]. However, it does not scale in high-dimensional space as shown in [13] 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 [29, 28]. 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.

6 Conclusion

In this work, we proposed optimizations to the index-based, TA-like algorithms for answering the cosine threshold 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.

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 David C. Anastasiu and George Karypis. L2AP: Fast cosine similarity search with prefix L-2 norm bounds. In *ICDE*, pages 784–795, 2014.

4 David C. Anastasiu and George Karypis. PL2AP: Fast parallel cosine similarity search. In *IA3*, pages 8:1–8:8, 2015.

5 Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and Optimal LSH for Angular Distance. In *NIPS*, pages 1225–1233, 2015.

6 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.

7 Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling Up All Pairs Similarity Search. In *WWW*, pages 131–140, 2007.

8 Alina Beygelzimer, Sham Kakade, and John Langford. Cover Trees for Nearest Neighbor. In *ICML*, pages 97–104, 2006.

9 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.

10 Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

11 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.

12 Nicolas Bruno, Luis Gravano, and Amélie Marian. Evaluating top-k queries over Web-accessible databases. In *ICDE*, pages 369–380, 2002.

13 Lu Chen, Yunjun Gao, Baihua Zheng, Christian S. Jensen, Hanyu Yang, and Keyu Yang. Pivot-based Metric Indexing. *PVLDB*, 10(10):1058–1069, 2017.

14 Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Introduction*. Springer, 2008.

15 Prasad M Deshpande, Deepak P, and Krishna Kummamuru. Efficient Online top-K Retrieval with Arbitrary Similarity Measures. In *EDBT*, pages 356–367, 2008.

16 Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. In *PODS*, pages 102–113, 2001.

17 Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.

18 Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kiebling. Optimizing Multi-Feature Queries for Image Databases. In *VLDB*, pages 419–428, 2000.

19 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.

20 Xiao Hu, Yufei Tao, and Ke Yi. Output-optimal Parallel Algorithms for Similarity Joins. In *PODS*, pages 79–90, 2017.

21 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.

22 Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *ICDT*, pages 604–613, 1998.

23 Harold W Kuhn and Albert W Tucker. Nonlinear programming. In *Traces and Emergence of Nonlinear Programming*, pages 247–258. Springer, 2014.

24 B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137, 2009.

25 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.

26 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.

27 Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.

28 Sharadh Ramaswamy and Kenneth Rose. Adaptive Cluster Distance Bounding for High-Dimensional Indexing. *TKDE*, 23(6):815–830, 2011.

29 Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

30 Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and Efficiency in High Dimensional Nearest Neighbor Search. In *SIGMOD*, pages 563–576, 2009.

31 Christina Teflioudi and Rainer Gemulla. Exact and Approximate Maximum Inner Product Search with LEMP. *TODS*, 42(1):5:1–5:49, 2016.

32 Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. LEMP: Fast Retrieval of Large Entries in a Matrix Product. In *SIGMOD*, pages 107–122, 2015.

33 Mingxun Wang and Nuno Bandeira. Spectral Library Generating Function for Assessing Spectrum-Spectrum Match Significance. *Journal of Proteome Research*, 12(9):3944–3951, 2013.

34 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.