Processing Ranked Queries with the Minimum Space

Yufei Tao[1] and Marios Hadjieleftheriou[2]

[1] Department of Computer Science, City University of Hong Kong,
Tat Chee Avenue, Hong Kong
taoyf@cs.cityu.edu.hk
[2] Department of Computer Science, Boston University, Boston, USA
marioh@cs.bu.edu

Abstract. Practical applications often need to rank multi-variate records by assigning various priorities to different attributes. Consider a relation that stores students' grades on two courses: *database* and *algorithm*. Student performance is evaluated by an "overall score" calculated as $w_1 \cdot g_{db} + w_2 \cdot g_{alg}$, where w_1, w_2 are two input "weights", and g_{db} (g_{alg}) is the student grade on *database* (*algorithm*). A "top-k ranked query" retrieves the k students with the best scores according to specific w_1 and w_2.

We focus on top-k queries whose k is bounded by a constant c, and present solutions that guarantee low worst-case query cost by using provably the minimum space. The core of our methods is a novel concept, "minimum covering subset", which contains only the necessary data for ensuring correct answers for all queries. Any 2D ranked search, for example, can be processed in $O(\log_B(m/B) + c/B)$ I/Os using $O(m/B)$ space, where m is the size of the minimum covering subset, and B the disk page capacity. Similar results are also derived for higher dimensionalities and approximate ranked retrieval.

1 Introduction

Practical applications often need to rank multi-variate records by assigning various priorities to different attributes. Consider a relation that stores students' grades on two courses: *database* and *algorithm*. Student performance is evaluated by an "overall score" calculated as $w_1 \cdot g_{db} + w_2 \cdot g_{alg}$, where w_1, w_2 are two input "weights", and g_{db} (g_{alg}) is the student's grade on *database* (*algorithm*). A common operation is to retrieve the best k students according to specific weights. For example, a top-k search with $w_1 = 1$ and $w_2 = 0$ returns students with the highest *database* grades, while a query with $w_1 = w_2 = 0.5$ selects students by the sum of their grades on the two courses. In this paper, we consider supporting ranked queries with low worst-case overhead for all user-defined weights.

1.1 Problem Statements

Consider a d-dimensional space, where each axis has a domain $[0, \infty)$. A *weight vector* $w = \{w[1], w[2], ..., w[d]\}$ specifies a positive weight $w[i]$ on each dimension $1 \leq i \leq d$. Given such a vector w, the *score* of a point p in the data space equals $\sum_{i=1}^{d}(w[i] \cdot p[i])$, where $p[i]$ is the coordinate of p on the i-th axis ($1 \leq i \leq d$).

J. Dix and S.J. Hegner (Eds.): FoIKS 2006, LNCS 3861, pp. 294–312, 2006.

Problem 1. Let $\mathcal{D}$ be a set of d-dimensional points. Given a weight vector w, a *top-k ranked query* returns the k points from $\mathcal{D}$ with the highest scores. Let c be a constant (by far) smaller than the cardinality of $\mathcal{D}$. The goal is to minimize the worst-case cost of processing any top-k query with $k \leq c$.

The motivation behind introducing the constant c is that a typical query in practice aims at finding only the "best-few" objects [11], e.g., the 10 best students from the whole university with a huge student population.

Many applications accept approximate answers with low (bounded) error, especially if computing such results requires less (time and space) overhead than the precise ones. Hence, we also consider a novel variation of ranked retrieval, called "top-(k,K) search". For example, a top-(1,10) query reports a single point whose score is at most the 10-th largest in the dataset $\mathcal{D}$. Hence, the query result is *not* unique, since it can be any of the objects whose scores are the 1st, 2nd, ..., 10th highest in $\mathcal{D}$. However, the quality of the result is guaranteed — in the worst case, the retrieved point is the "10-th best" in $\mathcal{D}$. Similarly, a legal outcome of a top-(3,10) query may consist of any 3 points in the top-10 set, and therefore, the number of permissible results equals $\binom{10}{3}$. We are ready to define the second problem tackled in this paper.

Problem 2. A *top-(k,K) ranked query* specifies a weight vector w, and two integers k, K with $k \leq K$. The query result includes any k objects in the top-K set for w. Let c and C be two constants (by far) smaller than the dataset cardinality, and $c \leq C$. The goal is to minimize the worst-case cost of any top-(k,C) query with $k \leq c$.

1.2 Previous Results

While Problem 1 has received considerable attention [10][4][5][7][8][9][11] in the database literature, the previous approaches mostly rely on heuristics which have poor worst-case performance. In particular, they require accessing the entire database to answer a single query [10][4][5][7], or consume space several times the dataset size [8][9].

The only exception is due to Tsaparas et al [11]. They propose an index that occupies $O(c^2 \cdot s/B)$ space and answers a 2D ranked query in $O(\log_B(s/B) + \log_B(c) + c/B)$ I/Os, where c is as defined in Problem 1, s is the size of the "c-skyline" of the dataset, and B is the disk page capacity. Specifically, a *c-skyline* consists of the objects that are not dominated by c other objects (an object p dominates another p' if the coordinates of p are larger on all dimensions). In the dataset of Figure 1, a 1-skyline contains p_2, p_3, p_7, p_4, p_5. Point p_1, for example, is not in the skyline because it is dominated by p_3.

The solutions of [11] are not applicable to higher dimensionalities. To the best of our knowledge, no previous results on Problem 2 exist.

1.3 Our Results

Any method that correctly solves Problem 1 must store a *minimum covering subset*, which is the result union of all the possible (an infinite number of) ranked queries with $k \leq c$. For example, in Figure 1, as clarified later the minimum covering subset for $c = 1$ contains p_2, p_3, p_4, p_5 — the top-1 object for any weight vector must be captured in this subset. Note that, the subset is smaller than the 1-skyline which, as mentioned earlier, also includes p_7 and p_6.

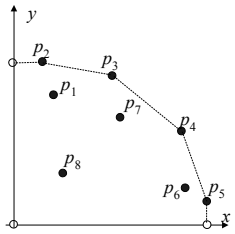


Fig. 1. An example

We propose polynomial-time algorithms for extracting minimum covering subsets in arbitrary dimensionalities. Notice that *the discovery of such a subset immediately improves the worst-case behavior of any previous approaches*. In particular, instead of applying a technique on the original dataset, we can deploy it on the minimum subset directly. Hence, any ranked query can be processed without considering other data not in the subset.

As a second step, we pre-process them into appropriate structures for performing ranked retrieval effectively. Specifically, any 2D ranked query can be answered in $O(\log_B\ (m/B) + c/B)$ I/Os using $O(m/B)$ space, where m is the size of the minimum covering subset. These bounds significantly improve those of [11]. For higher dimensionalities, a query can be solved in $O(m/B)$ I/Os by storing $O(m/B)$ information. Note that our methods in both scenarios require asymptotically the smallest amount $O(m/B)$ of space.

For Problem 2, there also exists a corresponding "minimum subset" containing the necessary data for ensuring correct results for all queries. If this subset has size m, we develop an index that occupies $O(m'/B)$ space, and processes an approximate 2D ranked query in $O(\log_B(m'/B)+c/B)$ I/Os, where m' is bounded by $(\ln m+1)\cdot m$. In higher-dimensional space, a query can be answered in $O(m'/B)$ I/Os with $O(m'/B)$ space.

The rest of the paper is organized as follows. Section 2 elaborates the definition of minimum covering subsets, and Section 3 discusses their computation in arbitrary dimensionality. Section 4 explains an "incremental" approach for deriving minimum covering subsets. Section 5 presents an index structure that optimizes exact ranked search, while Section 6 discusses approximate retrieval. Section 7 concludes the paper with directions for future work.

2 Minimum Covering Subsets

Let $\mathcal{D}_\subseteq$ be a subset of $\mathcal{D}$. We say that $\mathcal{D}_\subseteq$ *covers* an exact top-k query if $\mathcal{D}_\subseteq$ contains all the k points in its result. Given a constant c, $\mathcal{D}_\subseteq$ is a *c-covering subset* if it covers *all* possible top-k queries whose k is at most c. For instance, a "1-covering subset" includes the results of all top-1 queries, *regardless of their weight vectors*, while a "3-covering subset" covers all top-1, top-2, and top-3 queries. Among all the c-covering subsets, the one with the *smallest* size is called the *minimum c-covering subset*, represented as $min\mathcal{D}_\subseteq$.

As an example, consider a 1D dataset $\mathcal{D}$ with 10 records $\{1, 2, ..., 10\}$. Accordingly, a weight vector contains a single number w. A top-k query reports the k tuples $p \in \mathcal{D}$ that maximize $w \cdot p$. Clearly, the result simply consists of the k largest numbers in $\mathcal{D}$. Indeed, when the dimensionality equals 1, all top-k queries with the same k produce identical results, independent of w. Therefore, for any integer c, the minimum c-covering subset $min\mathcal{D}_{\subseteq}$ includes the c largest tuples. For instance, for $c = 1$, $min\mathcal{D}_{\subseteq} = \{10\}$, and for $c = 2$, $min\mathcal{D}_{\subseteq} = \{10, 9\}$, etc. Note that $\{10, 9\}$ is also a 1-covering subset, but not a minimal one.

The importance of $min\mathcal{D}_{\subseteq}$ lies in the fact that it can replace the original dataset $\mathcal{D}$ to correctly answer any top-k query, as long as the parameter k is not larger than c. This replacement significantly reduces the space consumption because, even if the cardinality of $\mathcal{D}$ is huge, $min\mathcal{D}_{\subseteq}$ may contain only a small number of points. Furthermore, notice that $min\mathcal{D}_{\subseteq}$ must be stored by any technique that aims at providing correct results to all top-k queries. *Hence, the size of $min\mathcal{D}_{\subseteq}$ corresponds to the lower bound for the space consumption of ranked retrieval.*

As stated in Problem 2, a top-(k,K) query returns k points in the top-K set for w. As mentioned in Section 1.1, the result is not unique — since any k objects in the top-K set forms a "legal" result, the number of possible results equals $\binom{K}{k}$. Given a subset $\mathcal{D}_{\subseteq}$ of the dataset $\mathcal{D}$, we say that $\mathcal{D}_{\subseteq}$ *covers* a top-(k,K) query, if there exist k points in $\mathcal{D}_{\subseteq}$ that constitute *one* of the $\binom{K}{k}$ legal results. We define the *(c,C)-covering subset* to be a subset of $\mathcal{D}$ that covers all top-(k,K) queries with $k \leq c$ and $K = C$. The (c,C)-covering subset with the smallest size is the *minimum (c,C)-covering subset* $min\mathcal{D}^*_{\subseteq}$, where the asterisk differentiates it from the notation $min\mathcal{D}_{\subseteq}$ of a minimum c-covering subset.

We illustrate these concepts using again a 1D dataset $\mathcal{D}$ storing data 1, 2, ..., 10. A possible minimum (1,5)-covering subset of $\mathcal{D}$ can involve a single tuple $\{6\}$. Indeed, for any top-(1,5) query, the tuple 6 is always a legal result since it is in the top-5 set. In fact, a minimum (1,5)-covering subset can involve any *single* record chosen from $\{6$, 7, 8, 9, 10$\}$. Similarly, a minimum (3,5)-covering subset can be $\{6$, 7, 8$\}$, or in general, any subset of $\{6$, 7, 8, 9, 10$\}$ with 3 elements.

The minimum (c,C)-covering subset $min\mathcal{D}^*_{\subseteq}$ can substitute the original database $\mathcal{D}$ to support top-(k,C) ranked search whose parameter k is at most c. Next, we discuss the computation of minimum c-covering subsets, while (c,C)-covering subsets are the topic of Section 6.

3 Finding Minimum C-Covering Subsets

In this section, we analyze extracting the minimum c-covering subset in arbitrary dimensionality. Section 3.1 first presents some fundamental results, based on which Section 3.2 elaborates the concrete algorithms.

3.1 Basic Results

Lemma 1. *Let $\mathcal{D}$ be a set of multi-dimensional points, and $\mathcal{D}_{\subseteq}$ be a subset of $\mathcal{D}$. If $\mathcal{D}_{\subseteq}$ covers all top-c queries on $\mathcal{D}$, then it also covers all top-k queries, for any $1 \leq k \leq c$.*

The lemma has an important corollary:

Corollary 1. *The minimum c-covering subset $min\mathcal{D}_{\subseteq}$ of $\mathcal{D}$ is the union of the results of all (exact) top-c queries.*

Hence, the computation of $min\mathcal{D}_{\subseteq}$ would be simple if we *were* able to execute an infinite number of top-c queries with all possible weight vectors w. In the sequel, we present polynomial-time approaches based on several inherent connections between $min\mathcal{D}_{\subseteq}$ and the "positive convex hull" $\mathcal{PCH}$ of dataset $\mathcal{D}$. To formally define $\mathcal{PCH}$, let us formulate a set $\mathcal{D}'$, which contains all the objects from $\mathcal{D}$, together with $d+1$ "dummy" points. The first one is the origin of the data space, and there is also a dummy point on each dimension, whose coordinate on this dimension equals the maximum coordinate of the points in $\mathcal{D}$ on this axis, and 0 on the others. All the $d+1$ dummy records must appear in the convex hull of $\mathcal{D}'$. The *positive convex hull* $\mathcal{PCH}$ of the original dataset $\mathcal{D}$ includes the non-dummy points in the convex hull of $\mathcal{D}'$.

Figure 1 illustrates an example where $\mathcal{D}$ contains 8 points $p_1, p_2, ..., p_8$. The augmented dataset $\mathcal{D}'$ involves 3 dummy points represented as white dots. The convex hull of $\mathcal{D}'$ contains all the dummy points, together with p_2, p_3, p_4, and p_5. Hence, the positive convex hull $\mathcal{PCH}$ of $\mathcal{D}$ consists of p_2, p_3, p_4, and p_5. Clearly, the time needed to compute the $\mathcal{PCH}$ of any dataset $\mathcal{D}$ is bounded by the cost of obtaining a complete convex hull of $\mathcal{D}$.

It is known [4] that the result of any top-1 query on $\mathcal{D}$ can be found in the $\mathcal{PCH}$ of $\mathcal{D}$, namely:

Lemma 2. *The minimum 1-covering subset of $\mathcal{D}$ is the positive convex hull of $\mathcal{D}$.*

It is natural to wonder whether this lemma can be trivially extended to capture minimum c-covering subsets for arbitrary c. Specifically, should the minimum 2-covering subset of $\mathcal{D}$ be the union of $\mathcal{PCH}(D)$ and $\mathcal{PCH}(D - \mathcal{PCH}(D))$? That is, can we obtain the minimum 2-covering subset by combining the minimum 1-covering subset, and the positive convex hull of the remaining data of $\mathcal{D}$ after excluding the points in $\mathcal{PCH}(D)$ (this corresponds to the Onion technique in [4])? Unfortunately, the answer is negative. For example, in Figure 1, $\mathcal{PCH}(D)$ equals $\{p_2, p_3, p_4, p_5\}$. After removing $\mathcal{PCH}(D)$, the positive convex hull of the remaining objects consists of p_1, p_7, p_6. However, as clarified in the next section, the minimum 2-covering subset of $\mathcal{D}$ has the same content as the minimum 1-covering subset.

3.2 Algorithm for Arbitrary Dimensionality

We prove a reduction that transforms the problem of discovering minimum c-covering subsets $min\mathcal{D}_{\subseteq}$ to finding positive convex hulls. As a result, $min\mathcal{D}_{\subseteq}$ can be obtained using any existing algorithm [3] for computing convex hulls. Given a value c and dataset $\mathcal{D}$, we represent the minimum c-covering subset as $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$. According to Lemma 2, $min\mathcal{D}_{\subseteq}(1, \mathcal{D})$ is equivalent to $\mathcal{PCH}(\mathcal{D})$.

Theorem 1. *For any $c \geq 2$, a minimum c-covering subset can be computed in a recursive manner:*

$$min\mathcal{D}_{\subseteq}(c, \mathcal{D}) = \mathcal{PCH}(D) \bigcup \left(\bigcup_{\forall p \in \mathcal{PCH}(\mathcal{D})} min\mathcal{D}_{\subseteq}(c-1, \mathcal{D} - \{p\}) \right) \quad (1)$$

Algorithm *find-minD* $(c, \mathcal{D})$
1. if $c = 1$
2. return $\mathcal{PCH}(\mathcal{D})$ //using any convex-hull method
3. else
4. $S = \mathcal{PCH}(\mathcal{D})$
5. for each point $p \in \mathcal{PCH}(\mathcal{D})$
6. $S = S \cup$ ***find-minD*** $(c-1, \mathcal{D} - \{p\})$
7. return S

Fig. 2. Finding $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$ in any dimensionality

Based on Theorem 1, Figure 2 describes the algorithm *find-minD* for retrieving $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$ in any dimensionality. We illustrate the idea of *find-minD* by using it to find the minimum-2-covering subset on the dataset $\mathcal{D}$ in Figure 1. First, *find-minD* invokes the selected convex-hull algorithm to extract $\mathcal{PCH}(\mathcal{D})$ = $\{p_2, p_3, p_4, p_5\}$, i.e., the content of $min\mathcal{D}_{\subseteq}(1, \mathcal{D})$. All the points in this set must belong to $min\mathcal{D}_{\subseteq}(2, \mathcal{D})$. To find the other objects in $min\mathcal{D}_{\subseteq}(2, \mathcal{D})$, *find-minD* removes a point, say p_2, from $\mathcal{D}$, and computes the $\mathcal{PCH}$ of the remaining data. The result of this computation is $\mathcal{PCH}(\mathcal{D} - \{p_2\}) = \{p_3, p_4, p_5\}$. Next, $min\mathcal{D}_{\subseteq}(2, \mathcal{D})$ is updated to the union of its current content and $\mathcal{PCH}(\mathcal{D} - \{p_2\})$, which incurs no change to $min\mathcal{D}_{\subseteq}(2, \mathcal{D})$. *Find-minD* performs the above operations with respect to every other point p_3, p_4, and p_5 of $\mathcal{PCH}(\mathcal{D})$ in turn. Namely, for each $i = 3, 4, 5$, it obtains $\mathcal{PCH}(\mathcal{D} - \{p_i\})$, and updates $min\mathcal{D}_{\subseteq}(2, \mathcal{D})$ by union-ing it with $\mathcal{PCH}(\mathcal{D} - \{p_i\})$. It can be easily verified that no modification to $min\mathcal{D}_{\subseteq}(2, \mathcal{D})$ happens, leaving the final result $min\mathcal{D}_{\subseteq}(2, \mathcal{D})$ = $\{p_2, p_3, p_4, p_5\}$, that is, same as $\mathcal{PCH}(\mathcal{D})$.

Note that *find-minD* needs to compute the positive convex hull $\mathcal{PCH}$ of several different datasets. For example, in our earlier example of computing $min\mathcal{D}_{\subseteq}(2, \mathcal{D})$ in Figure 1, we calculated the $\mathcal{PCH}$ of $\mathcal{D}$, and $\mathcal{D} - \{p_i\}$ for each integer i in the range $2 \leq i \leq 5$. Hence, we have:

Theorem 2. *Let α be the highest cost of each $\mathcal{PCH}$ computation in* find-minD *(line 4), and β be the maximum number of $\mathcal{PCH}$ points retrieved in each execution of line 4. Then, the cost of* find-minD *is $O(\alpha \cdot \beta^{c-1})$.*

The factor α in Theorem 2 corresponds to the efficiency of the algorithm used by *find-minD* to compute $\mathcal{PCH}$.

Thus, a direct corollary of the theorem is:

Corollary 2. *The execution time of* find-minD *is worse than that of the deployed algorithm for computing convex hulls by at most a polynomial factor β^{c-1}.*

Notice that β is $O(|\mathcal{D}|)$ (the database cardinality) in the worst case. This happens in the very rare case where almost all the points in $\mathcal{D}$ belong to the positive convex hull. For practical datasets, β is fairly small. For instance, for uniform data, β is at the order of $(\ln |\mathcal{D}|)^{d-1}/(d-1)!$ [2], where d is the dimensionality of the dataset. For anti-correlated data (where most points lie around the major diagonal of the data space), β is expected to be a constant. In this case, *find-minD* is asymptotically as fast as computing the convex hull.

4 Incremental Computation of $min\mathcal{D}_{\subseteq}$

Before computing the minimum c-covering subset $min\mathcal{D}_{\subseteq}$, the algorithm in Figure 2 requires that the value of c should be known in advance. In the sequel, we present an alternative method that obtains the $min\mathcal{D}_{\subseteq}$ in a faster "incremental" manner. Specifically, it first finds the minimum 1-covering subset, which is used to discover the 2-covering subset, then the 3-covering, and so on. Our discussion focuses on 2D space in this section, while the extension to higher dimensionalities is presented Section 5.2.

4.1 Slope Space and Its Decomposition

Given a 2D weight vector $w = \{w[1], w[2]\}$, we refer to $\frac{w[2]}{w[1]}$ as the *weight slope* λ. Clearly, λ ranges in the *slope space* $[0, \infty)$. The following lemma is proved in [11]:

Lemma 3. *For an arbitrary integer k, top-k queries with the same weight slope λ return identical results. In particular, if we project each data point onto a ray shot from the origin with slope λ, the query result consists of the k objects whose projected points are the farthest from the origin.*

For example, the top-k objects produced by a query with weight vector $\{10, 20\}$ are the same as those reported by a query with weight vector $\{1, 2\}$ — both queries have a weight slope 2. Figure 3 shows a dataset that contains 3 data points p_1, p_2, and p_3. Ray l has a slope 2, and the projection of point p_i onto l is p_i' for each integer $i \in [1, 3]$. Since p_1' (p_2') is the farthest (2nd farthest) from the origin among the 3 projections, object p_1 (p_2) has the highest (2nd highest) score for the weight slope 2.

In the sequel we characterize a top-k query by the value of k and its weight slope λ. By Corollary 1, $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$ corresponds to the union of the results of top-c queries for all $\lambda \in [0, \infty)$. Imagine that we slowly increase λ from 0, and meanwhile continuously monitor the corresponding top-c set. Since the size of $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$ is finite, there can be only a finite number of top-c changes as λ travels from 0 to ∞. Therefore, we can decompose the slope space $[0, \infty)$ into a set of *disjoint* intervals such that the results are identical for those queries whose weight slopes are in the same interval.

We call the set $\mathcal{HD}$ of intervals thus obtained the *top-c homogeneous decomposition* of the slope space. The size of $\mathcal{HD}$ equals the total number of times the top-c objects incur a "change" as λ grows from 0 to ∞. A change here means that a data point is

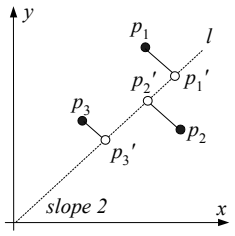


Fig. 3. Deciding score relationship from projections on l

removed from the current top-c set, and another point is added. Note that, no change is generated if only the scores of the existing top-c points switch their relative order. For instance, assume that the top-2 result for the current λ contains p_1, p_2, and the score of p_1 is higher. As λ increases to some value, the score of p_2 becomes larger, but p_1, p_2 still have the 2 largest scores among all the data points. This is *not* counted as a result change.

4.2 Computing Top-1 Homogeneous Decomposition

Let $I = [I_{\vdash}, I_{\dashv})$ be an interval in $\mathcal{HD}$, where $I_{\vdash}$ ($I_{\dashv}$) is the *starting* (*ending*) slope of I. For any weight slope in $[I_{\vdash}, I_{\dashv})$, the top-c set is the same, and is represented as $I.S$. To compute the top-1 homogeneous decomposition, we first obtain the minimum 1-covering subset $min\mathcal{D}_{\subseteq}(1, \mathcal{D})$ (i.e., the positive convex hull $\mathcal{PCH}$ of $\mathcal{D}$). Assume, without loss of generality, that the $\mathcal{PCH}$ contains m points p_1, p_2, ..., p_m sorted in descending order of their x-coordinates, where m is the size of $\mathcal{PCH}$. By the definition of the $\mathcal{PCH}$, if we start from p_1 and walk on the hull boundary (passing vertices p_2, ..., p_m in this order), we will make a left turn every time a vertex is encountered. This implies that the slopes of hull edges monotonically decrease in the order that we walk through them. Figure 4 shows an example where the $\mathcal{PCH}$ contains 4 points p_1, ..., p_4. The slope of edge p_1p_2 is larger than that of p_2p_3 (note that both slopes are negative), which in turn is greater than the slope of p_3p_4.

Let us shoot a ray l_i from the origin of the data space, vertically to the hull boundary p_ip_{i+1} for each $i \in [1, m-1]$. The slopes of the $m-1$ rays l_1, l_2, ..., l_{m-1} must increase monotonically. For example, in Figure 4 where $m = 3$, we obtain 3 rays l_1, l_2, and l_3. Since the edge p_1p_2 has a larger slope than p_2p_3, the slope of l_1 is smaller than that of l_2. This can be verified easily with the fact that the product of the slopes of two mutually orthogonal lines equals -1. Similarly, l_2 has a smaller slope than l_3.

Lemma 4. *Assume that the positive convex hull of $\mathcal{D}$ contains m points p_1, p_2, ..., p_m, sorted in descending order of their x-coordinates. Let i be any integer in the range $[1, m-1]$, and x be the slope of the ray shot from the origin perpendicular to the segment connecting p_i and p_{i+1}. Then, p_i has a larger score than p_{i+1} for any weight slope in $[0, x)$, while p_{i+1} has a higher score for any weight slope in (x, ∞).*

For example, let λ_1 be the slope of l_1 in Figure 4. The score of p_1 is larger than that of p_2 (for any weight slope) in the range $[0, \lambda_1)$, while p_2 has a greater score than p_1 in (λ_1, ∞). Similarly, if the slope of l_2 is λ_2, the score of p_2 is larger than that of p_3 for any weight slope smaller than λ_2, while p_3 has a higher score for slopes larger than λ_2.

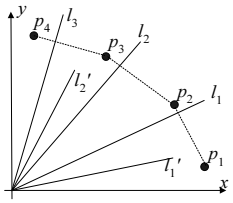


Fig. 4. Top-1 homogeneous decomposition

Based on Lemma 4, the top-1 homogeneous decomposition $\mathcal{HD}$ can be decided as follows. Given $\mathcal{PCH} = \{p_1, ..., p_m\}$ and $m-1$ rays l_1, ..., l_{m-1} as described earlier, the first interval I_1 in $\mathcal{HD}$ is $[I_{1\vdash}, I_{1\dashv})$, where $I_{1\vdash} = 0$, and $I_{1\dashv}$ equals the slope of l_1. The second interval I_2 starts at $I_{1\dashv}$, and terminates at the slope of ray l_2, and so on. The last interval I_m starts at the slope of I_{m-1}, and ends at ∞. The top-1 object for all weight slopes in interval I_i ($1 \leq i \leq m$) is point p_i, i.e., $I_i.S = \{p_i\}$.

Lemma 5. *Assume that the positive convex hull of $\mathcal{D}$ contains m points p_1, p_2, ..., p_m, sorted in descending order of their x-coordinates. θ_1, θ_2 are two arbitrary weight slopes, and $\theta_1 < \theta_2$. Let p_i (p_j) be the top-1 object for slope θ_1 (θ_2). Then, the union of top-1 objects for all weight slopes in $[\theta_1, \theta_2]$ equals $\{p_i, p_{i+1}, ..., p_j\}$.*

For example, let θ_1 (θ_2) be the slope of the ray l'_1 (l'_2) in Figure 4. Point p_1 (p_3) is the object that has the largest score at θ_1 (θ_2). Then, for any weight slope in the range $[\theta_1, \theta_2]$, the top-1 object must be found in the set $\{p_1, p_2, p_3\}$.

4.3 Computing Top-k Homogeneous Decomposition

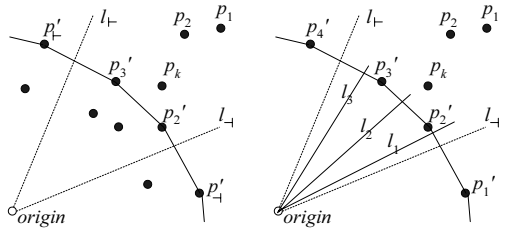
The subsequent analysis shows that the top-(k+1) homogeneous decomposition $\mathcal{HD}(k+1, \mathcal{D})$ can be derived efficiently from the top-k counterpart $\mathcal{HD}(k, \mathcal{D})$.

Tail Sets. Consider an arbitrary interval $I = [I_\vdash, I_\dashv) \in \mathcal{HD}(k, \mathcal{D})$. $I.S$ contains k points (in $\mathcal{D}$) with the highest scores for any weight slope in $[I_\vdash, I_\dashv)$. To derive $\mathcal{HD}(k+1, \mathcal{D})$, we need to decide the union of the top-$(k+1)$ objects produced by all weight slopes in $[I_\vdash, I_\dashv)$. Let $(\mathcal{D} - I.S)$ be the set of points in $\mathcal{D}$ after excluding those in $I.S$. We define $p'_\vdash$ ($p'_\dashv$) to be the object in $(\mathcal{D} - I.S)$ that has the largest score for the weight slope $I_\vdash$ ($I_\dashv$). By Lemma 2, both $p'_\vdash$ and $p'_\dashv$ appear on the positive convex hull of $(\mathcal{D} - I.S)$. Then, the *tail set* $I.TS$ of I contains the vertices of this hull *between (but including)* $p'_\vdash$ and $p'_\dashv$.

Figure 5a shows an example where the black dots represent the data points in $\mathcal{D}$. Assume an interval $I \in \mathcal{HD}(k, \mathcal{D})$ whose starting value $I_\vdash$ (ending value $I_\dashv$) equals the slope of ray $l_\vdash$ ($l_\dashv$). Objects p_1, p_2, ..., p_k are in the top-k set $I.S$ for all weight slopes in $[I_\vdash, I_\dashv)$ (p_3, ..., p_{k-1} are omitted from the figure). Point $p'_\vdash$ ($p'_\dashv$) is the object that has the highest score at weight slope $I_\vdash$ ($I_\dashv$) among the data in $(D - I.S)$. Objects $p'_\vdash$, p'_2, p'_3, $p'_\dashv$ are the vertices between $p'_\vdash$ and $p'_\dashv$ on the positive convex hull of $(D - I.S)$; hence, they constitute the tail set $I.TS$ of I. According to the following lemma, the result of any top-(k+1) query with weight slope in $[I_\vdash, I_\dashv)$ must be included in the union of $I.S$ and $\{p'_\vdash, p'_2, p'_3, p'_\dashv\}$.

Lemma 6. *Let I be an arbitrary interval in the top-k homogeneous decomposition $\mathcal{HD}(k, \mathcal{D})$. The union of the top-($k$+1) objects for weight slopes in $[I_\vdash, I_\dashv)$ equals $I.S \cup I.TS$, where $I.S$ is the top-k set in $[I_\vdash, I_\dashv)$, and $I.TS$ the tail set of I.*

An important step in extracting the tail set of I is to identify points $p'_\vdash$ and $p'_\dashv$, which are the top-1 objects in $(\mathcal{D} - I.S)$ at slopes $I_\vdash$ and $I_\dashv$, respectively. Both points can be efficiently obtained as follows. First, we sort all the intervals in the top-k homogeneous decomposition $\mathcal{HD}(k, \mathcal{D})$ in ascending order of their starting slopes. To decide $p'_\vdash$ ($p'_\dashv$),



(a) Obtaining $min\mathcal{D}_{\subseteq}(k, \mathcal{D})$ (b) Obtaining $\mathcal{HD}(k, \mathcal{D})$

Fig. 5. The tail set and its application

we check whether I is the first (last) interval in the sorted list. If the answer is positive, $p'_{\vdash}$ ($p'_{\dashv}$) corresponds to the point in $\mathcal{D}$ with the (k+1)-st largest x- (y-) coordinate. Otherwise, we take the interval I' that ranks just before (after) I in the sorted list. By the definition of $\mathcal{HD}(k, \mathcal{D})$ (see Section 4.1), $I'.S$ involves one object that is not in $I.S$, i.e., the top-k result change at the boundary between I and I'. Then, $p'_{\vdash}$ ($p'_{\dashv}$) is set to this object. Figure 6 shows the formal procedures of computing the tail set.

After computing the tail sets of all intervals in $\mathcal{HD}(k, \mathcal{D})$, we can obtain the minimum (k+1)-covering subset of the dataset $\mathcal{D}$ immediately:

Theorem 3. *Given the top-k homogeneous decomposition $\mathcal{HD}(k, \mathcal{D})$, the minimum ($k$+1)-covering subset can be decided as:*

$$min\mathcal{D}_{\subseteq}(k+1, \mathcal{D}) = \left(\bigcup_{I \in \mathcal{HD}(k,\mathcal{D})} I.S \cup I.TS \right) \tag{2}$$

Computing $\mathcal{HD}(k+1, \mathcal{D})$. Next we clarify the derivation of $\mathcal{HD}(k+1, \mathcal{D})$ from the tail sets of the intervals in $\mathcal{HD}(k, \mathcal{D})$. Before presenting the detailed algorithm, we first discuss the general idea using a concrete example in Figure 5b, which is based on Figure 5a. As mentioned earlier, $I = [I_{\vdash}, I_{\dashv})$ is an interval in $\mathcal{HD}(k, \mathcal{D})$, where $I_{\vdash}$ ($I_{\dashv}$) equals the slope of ray $l_{\vdash}$ ($l_{\dashv}$). The tail set $I.TS$ consists of points p'_1, p'_2, ..., p'_4, sorted in descending order of their x-coordinates (note that p'_1 and p'_4 are equivalent to $p'_{\vdash}$ and $p'_{\dashv}$ in Figure 5a, respectively). For each segment $p'_i p'_{i+1}$ ($1 \leq i \leq 3$), we shoot a ray l_i from the origin perpendicularly to it. The slope of λ_i of l_i must be larger than $I_{\vdash}$ and but smaller than $I_{\dashv}$ (note that l lies between $l_{\vdash}$ and $l_{\dashv}$). The 3 numbers λ_1, λ_2, and λ_3 divide $[I_{\vdash}, I_{\dashv})$ into 4 pieces with different top-(k+1) results. To facilitate illustration, let us denote $\lambda_0 = I_{\vdash}$ and $\lambda_4 = I_{\dashv}$. Then, for $i \in [0, 3]$, the top-(k+1) set contains p'_i and the objects in $I.S$ at any weight slope in $[\lambda_i, \lambda_{i+1})$. Recall that $I.S$ = $\{p_1, ..., p_k\}$ is the top-k result at any weight slope in I.

Formally, given an interval $I \in \mathcal{HD}(k, \mathcal{D})$, the algorithm *break-interval* in Figure 7 divides I into a set $\mathcal{HD}(\mathcal{I})$ of disjoint pieces. Each interval $I' \in \mathcal{HD}(\mathcal{I})$ is associated with a set $I'.S$ that is the top-(k+1) result for any weight slope in I'. In order to obtain $\mathcal{HD}(k+1, \mathcal{D})$, we execute *break-interval* for every interval $I \in \mathcal{HD}(k, \mathcal{D})$, after which $\mathcal{HD}(k+1, \mathcal{D})$ corresponds to the union of all $\mathcal{HD}(\mathcal{I})$ produced.

There is one minor detail worth mentioning. As will be shown in an example, the $\mathcal{HD}(k+1, \mathcal{D})$ thus decided may contain multiple intervals I whose associated top-(k+1) sets $I.S$ are the same. These intervals should be combined into a single one

Algorithm *tail-set* (I, $\mathcal{HD}(k, \mathcal{D})$)
/* I is an interval in $\mathcal{HD}(k, \mathcal{D})$, whose intervals have been sorted by their starting values */
1. if $I_{\vdash} = 0$ //I is the first interval in $\mathcal{HD}(k, \mathcal{D})$
2. $p'_{\vdash}$ = the point in $\mathcal{D}$ with the (k+1)-st highest x-coordinate
3. else
4. I' = the interval that ranks before I in $\mathcal{HD}(k, \mathcal{D})$
5. $p'_{\vdash}$ = $I'.S - I.S$
6. if $I_{\dashv} = \infty$ //I is the last interval in $\mathcal{HD}(k, \mathcal{D})$
7. $p'_{\dashv}$ = the point in $\mathcal{D}$ with the (k+1)-st highest y-coordinate
8. else
9. I' = the interval that ranks after I in $\mathcal{HD}(k, \mathcal{D})$
10. $p'_{\dashv}$ = $I'.S - I.S$
11. $I.TS$ = the set of vertices between (including) $p'_{\vdash}$ and $p'_{\dashv}$ on the convex hull of $\mathcal{D} - I.S$

Fig. 6. Algorithm for computing the tail set

Algorithm *break-interval* (I, $I.S$, $I.TS$)
/* I is an interval in $\mathcal{HD}(k, \mathcal{D})$; $I.S$ is the top-k set at any weight slope in I; $I.TS$ is the tail set of I */
1. $\theta_{last} = I_{\vdash}$; $\mathcal{HD}(\mathcal{I}) = \emptyset$; m = number of points in $I.TS$
2. sort the points in $I.TS$ in descending order of their x-coordinates; let the sorted order be $\{p'_1, p'_2, ..., p'_m\}$
3. for i = 1 to $m - 1$
4. shoot a ray from the origin perpendicularly to the segment connecting p'_i and p'_{i+1}
5. λ = the slope of the ray
6. create an interval $I_i = [I_{i\vdash}, I_{i\dashv})$ with $I_{i\vdash} = \theta_{last}$, $I_{i\dashv} = \lambda$, and $I_i.S = p'_i \cup I.S$
7. $\mathcal{HD}(\mathcal{I}) = \mathcal{HD}(\mathcal{I}) \cup \{I_i\}$, and $\theta_{last} = \lambda$
8. add to $\mathcal{HD}(\mathcal{I})$ $I_m = [\theta_{last}, I_{\dashv})$ with $I_m.S = p'_m \cup I.S$
9. return $\mathcal{HD}(\mathcal{I})$

Fig. 7. Algorithm for breaking an interval in $\mathcal{HD}(k, \mathcal{D})$ into ones in $\mathcal{HD}(k + 1, \mathcal{D})$

which spans all their respective slope ranges. This *duplicate-removal* process, as well as the overall algorithm for computing $\mathcal{HD}(k + 1, \mathcal{D})$ is shown in Figure 8.

An Example. Consider Figure 9, where p_1 and p_2 are the vertices on the positive convex hull of a dataset $\mathcal{D}$. Hence, they constitute the minimum 1-covering subset $min\mathcal{D}_{\subseteq}(1, \mathcal{D})$. Let λ_1 be the slope of ray l_1, which passes the origin and is vertical to segment p_1p_2. The top-1 homogeneous decomposition $\mathcal{HD}(1, \mathcal{D})$ contains two intervals $I_1 = [0, \lambda_1)$ and $I_2 = [\lambda_1, \infty)$. The top-1 object (for any weight slope) in I_1 is $I_1.S = \{p_1\}$, and that in I_2 is $I_2.S = \{p_2\}$.

Next we compute $min\mathcal{D}_{\subseteq}(2, \mathcal{D})$ and $\mathcal{HD}(2, \mathcal{D})$, by considering each interval of $\mathcal{HD}(1, \mathcal{D})$ in turn. For the first interval $I_1 = [0, \lambda_1)$, its tail set $I_1.TS$ consists of points p_4 and p_1. Accordingly, the algorithm *break-interval* in Figure 7 divides I_1 into two

Algorithm *ho-decomp* ($\mathcal{HD}(k, \mathcal{D})$)
1. $minD_{\subseteq}(k+1, \mathcal{D}) = \emptyset$; $\mathcal{HD}(k+1, \mathcal{D}) = \emptyset$
2. assume that $\mathcal{HD}(k, \mathcal{D})$ contains m intervals I_1, ..., I_m sorted in ascending order of their starting values
3. for i = 1 to m
4. $I.TS$ = *tail-set* (I_i, $\mathcal{HD}(k, \mathcal{D})$)
5. $minD_{\subseteq}(k+1, \mathcal{D}) \cup = I.S \cup I.TS$
6. $\mathcal{HD}(k+1, \mathcal{D}) \cup =$ *break-interval* (I, $I.S$, $I.TS$)
7. $\mathcal{HD}(k+1, \mathcal{D})$ = *remove-duplicate* ($\mathcal{HD}(k+1, \mathcal{D})$)
8. return $\mathcal{HD}(k+1, \mathcal{D})$

Algorithm *remove-duplicate* ($\mathcal{HD}(k+1, \mathcal{D})$)
//assume that $\mathcal{HD}(k+1, \mathcal{D})$ contains m' intervals I'_1, ..., $I'_{m'}$
1. $\delta\mathcal{HD}_- = \delta\mathcal{HD}_+ = \emptyset$
2. sort all intervals in $\mathcal{HD}(k+1, \mathcal{D})$ in ascending order of their starting slopes;
3. $i = 1$
4. while $i \leq m-1$
5. j = the largest integer s.t. $I'_i.S = I'_{i+1}.S = ... = I'_j.S$
6. add I'_i, I'_{i+1}, ..., I'_j into $\delta\mathcal{HD}_-$
7. create a new interval $I = \cup_{x=i}^{j} I'_x$ with $I.S = I'_i.S$
8. $i = j + 1$
9. $\mathcal{HD}(k+1, \mathcal{D}) = \mathcal{HD}(k+1, \mathcal{D}) - (\delta\mathcal{HD}_-) \cup (\delta\mathcal{HD}_+)$

Fig. 8. Algorithm for computing $\mathcal{HD}(k+1, \mathcal{D})$

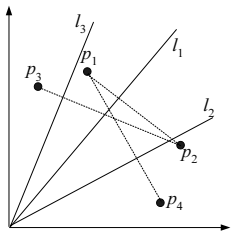


Fig. 9. Illustration of the algorithm in Figure 8

intervals $I'_1 = [0, \lambda_2)$ and $I'_2 = [\lambda_2, \lambda_1)$, where λ_2 is the slope of ray l_2 perpendicular to segment p_1p_4. The top-2 result in I'_1 (I'_2) is $I'_1.S$ = $\{p_2, p_4\}$ ($I'_2.S$ = $\{p_2, p_1\}$). Let $\mathcal{HD}(I_1)$ be the set $\{I'_1, I'_2\}$.

Similarly, we examine the second interval $I_2 = [\lambda_1, \infty)$ of $\mathcal{HD}(1, \mathcal{D})$. Its tail set $I_2.TS$ includes objects p_2, p_3. *Break-interval* divides I_2 at the slope λ_3 of ray l_3 orthogonal to segment p_2p_3. Specifically, I_2 is broken into $\mathcal{HD}(I_2)$ = $\{I'_3, I'_4\}$, where $I'_3 = [\lambda_1, \lambda_3)$ and $I'_4 = [\lambda_3, \infty)$. The top-2 result in I'_3 is $I'_3.S$ = $\{p_2, p_1\}$, and that in I'_4 is $I'_4.S$ = $\{p_3, p_1\}$.

Theorem 3 shows that the minimum 2-covering subset involves all the data in Figure 9, which equals the union of $I_1.S$, $I_1.TS$, $I_2.TS$ and $I_2.TS$. Furthermore, by merging $\mathcal{HD}(I_1)$ and $\mathcal{HD}(I_2)$, we obtain a top-2 homogeneous decomposition $\mathcal{HD}(2, \mathcal{D})$ with 4 intervals I'_1, ..., I'_4. The top-2 results in I'_2 and I'_3, however, are both

$\{p_2, p_1\}$. Therefore, algorithm *remove-duplicate* in Figure 8 combines these two intervals into one $I'_5 = I'_2 \cup I'_3 = [\lambda_2, \lambda_3)$, with $I'_5.S = \{p_2, p_1\}$. Therefore, the final $\mathcal{HD}(2, \mathcal{D})$ involves only 3 elements: I'_1, I'_5, and I'_3.

4.4 Analysis

The subsequent discussion aims at bounding (i) the size of the top-c homogeneous decomposition $\mathcal{HD}(c, \mathcal{D})$, and (ii) the time of computing $\mathcal{HD}(c, \mathcal{D})$. We have:

Theorem 4. *The number of intervals in the top-c homogeneous decomposition $\mathcal{HD}(c, \mathcal{D})$ is asymptotically the same as the number of points in the corresponding minimum c-covering subset $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$.*

Now we analyze the cost of computing top-c homogeneous decompositions.

Theorem 5. *Let α be the time of computing the convex hull of $\mathcal{D}$, and α_i the highest cost of computing the tail set of an interval in $\mathcal{HD}(i, \mathcal{D})$ ($2 \leq i \leq c$). $\mathcal{HD}(c, \mathcal{D})$ and $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$ can be computed in $O(\alpha + \sum_{i=2}^{c}(\alpha_i \cdot |min\mathcal{D}_{\subseteq}(i-1, \mathcal{D})|))$ time, where $|min\mathcal{D}_{\subseteq}(i-1, \mathcal{D})|$ is the number of objects in $min\mathcal{D}_{\subseteq}(i-1, \mathcal{D})$.*

Factor α_i ($1 \leq i \leq c$) depends on the concrete convex-hull algorithm for calculating the tail set. The theorem indicates that $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$ can be computed in shorter time in the 2D space than the algorithm presented in Section 3 (which applies to any dimensionality). To better illustrate this, we utilize the fact that $|min\mathcal{D}_{\subseteq}(i, \mathcal{D})|$ is bounded by $|min\mathcal{D}_{\subseteq}(c, \mathcal{D})|$ for any $i < c$, which results in:

Corollary 3. *The 2D minimum c-covering subset can be computed in $O(\alpha + \alpha_{max} \cdot |min\mathcal{D}_{\subseteq}(c-1, \mathcal{D})| \cdot (c-1))$ time, where α_{max} equals the maximum of α_2, α_3, ..., α_c defined in Theorem 5.*

5 Ranked Indexes

Once the minimum c-covering subset $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$ has been discovered, we can correctly answer any top-k query, with arbitrary weight slope and $k \leq c$, by performing $O(m/B)$ I/Os, where m is the number of points in $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$, and B is the size of a disk page. Although being relatively straightforward, this method constitutes the first solution for ranked retrieval in any dimensionality that does not require examining the entire dataset in the worst case.

In the next section, we show how to pre-process $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$ to further reduce query cost in the 2D space. In Section 5.2, we present pessimistic results that explain why a similar approach is not feasible for dimensionalities $d \geq 3$.

5.1 A 2D Solution

As an obvious approach, we could extract the top-c homogeneous decomposition $\mathcal{HD}(c, \mathcal{D})$. Assume that it contains m intervals I_1, I_2, ..., I_m, sorted in ascending order of their starting values $I_{1\vdash}$, ... , $I_{m\vdash}$, which are indexed by a B-tree. The B-tree entry for

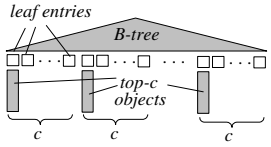


Fig. 10. Indexing the top-c homog. decomp.

each $I_{i\vdash}$ ($1 \leq i \leq m$) is associated with a pointer to a sequence of $O(c/B)$ pages, storing the c objects in $I_i.S$. Given a top-k query with weight slope λ, we first locate the leaf entry I_{i-} in the B-tree that is the largest among all the leaf entries smaller than λ. (i.e., λ falls in the corresponding interval I_i). This requires $O(\log_B m)$ I/Os. Following the pointer stored with I_{i-}, we retrieve $I_i.S$ (using $O(c/B)$ I/Os), and report the k records in $I_i.S$ with the highest scores at λ. The total query cost is $O(\log_B m + c/B)$ I/Os.

The space consumption of this approach is $O(c \cdot m/B)$ pages. We present an alternative solution that achieves the same query time but with only $O(m/B)$ space. Recall that for two consecutive intervals I_i and I_{i+1}, $I_i.S$ differs from $I_{i+1}.S$ by exactly one point. Motivated by this, we store $I.S$ in a compact way as illustrated in Figure 10. For the first interval I_1, c/B pages are used to record the c objects of $I_1.S$ in the same way as the previous method. For the next $c-1$ intervals I_i ($2 \leq i \leq c$), we do not materialize the full $I_i.S$. Instead, we keep only the difference between $I_i.S$ and $I_{i-1}.S$ in the (B-tree) leaf entry I_i using $O(1)$ space. The above process is repeated for the next c intervals I_{c+1}, I_{c+2}, ..., I_{2c}. Specifically, we write to the disk the complete $I_{c+1}.S$. For I_i with $c+2 \leq i \leq 2c$, only the $O(1)$ result changes (between $I_i.S$ and $I_{i-1}.S$) are kept. Then, a similar process is performed for still the next c intervals, and so on. Since $O(c/B)$ space is allocated for every c intervals, the total space consumption is $O(\frac{m}{c} \cdot c/B) = O(m/B)$ pages.

Given a top-k query with weight slope λ, we first identify the largest leaf entry $I_{i\vdash}$ smaller than λ in the same way as in the previous solution. First, we scan, at the leaf level, the preceding leaf entries $I_{(i-1)\vdash}$, $I_{(i-2)\vdash}$, ... (in this order), until finding the first entry $I_{i'\vdash}$ ($i' \leq i$) whose $I_{i'\vdash}.S$ is completely stored. Then, we create a copy S of $I_{i'\vdash}.S$ in the memory, and re-visit the leaf entries (that were just scanned) in a *reverse* order: $I_{(i'+1)\vdash}$, $I_{(i'+2)\vdash}$, ..., $I_{i\vdash}$. At each $I_{j\vdash}$ for $i'+1 \leq j \leq i$, we update S to the top-c result $I_j.S$ (of any weight slope) in interval I_j by applying the $O(1)$ result changes recorded in $I_{j\vdash}$. Hence, when we arrive at $I_{i\vdash}$, the content of S becomes $I_i.S$, from which the k records with the highest scores at the query slope λ are returned. Since we need to trace (from $I_{i\vdash}$ to $I_{i'\vdash}$) at most $O(c)$ leaf entries (by accessing $O(c/B)$ pages), the query cost is bounded by $O(\log_B m + c/B)$ I/Os.

By Theorem 4, the size m of $\mathcal{HD}(c, \mathcal{D})$ is asymptotically the same as the number of points in $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$. Therefore, we have:

Theorem 6. *Given a constant c, we can pre-process a 2D dataset $\mathcal{D}$ into a structure that consumes $O(m/B)$ space, and answers any top-k query with $k \leq c$ in $O(\log_B (m/B) + c/B)$ I/Os, where m is the size of the minimum c-covering subset.*

Evidently, the 2D ranked index presented earlier consumes the smallest amount of space, since any solution that ensures correct answers for all ranked queries must store

at least $min\mathcal{D}_{\subseteq}(c, \mathcal{D})$. In practice the disk page size B is usually fairly large (a typical value is 4k bytes), and is asymptotically comparable to c (i.e., $c = O(B)$). In this case, the query performance of our solution becomes $\log_B(m/B)$.

5.2 Discussion on Higher Dimensionalities

The technique in the previous section can be extended to dimensionality $d > 2$ for achieving logarithmic query cost. Such an extension, however, turns out to be purely of theoretical interests because it requires expensive space consumption for large c (and hence violates our goal of using the minimum space to support ranked retrieval). In the sequel, we discuss this in the 3D space, since the analysis for even higher dimensionalities is similar.

Given a 3D weight vector $w = \{w[1], w[2], w[3]\}$, we define its *slope vector* λ as a 2D vector $\{\frac{w[1]}{w[3]}, \frac{w[2]}{w[3]}\}$. Hence, λ can be regarded as a point in a 2D *slope space*, where both axes have domain $[0, \infty)$. Similar to Lemma 3, two weight vectors with the same slope vector have identical top-k sets (for any k). Therefore, in the sequel, we represent a top-k query equivalently using k and its slope vector λ.

Lemma 7. *Given two 3D points p_1 and p_2, let pl be a function of 2D slope vector $\lambda = \{\lambda[1], \lambda[2]\}$:*

$$pl(\lambda) = \sum_{i=1}^{2} (\lambda[i] \cdot (p_1[i] - p_2[i])) + (p_1[3] - p_2[3]) \quad (3)$$

Then, p_1 has a higher score than p_2 for all slope vectors λ in a half-plane $pl(\lambda) > 0$, while the score of p_2 is higher in $pl(\lambda) < 0$. The scores of the two points are identical for λ on the line $pl(\lambda) = 0$.

Given the c-covering minimum subset $min\mathcal{D}_{\subseteq}$ of a 3D dataset $\mathcal{D}$, the slope space can be divided into a set of disjoint regions, such that the top-c sets for all slope vectors in a region are the same. Following the terminology in Section 4, we call such a division a *top-c homogeneous decomposition*, and denote it as $\mathcal{HD}(c, \mathcal{D})$. In the sequel, we first analyze the top-1 homogeneous decomposition $\mathcal{HD}(1, \mathcal{D})$, and then generalize the discussion to top-c.

Let $min\mathcal{D}_{\subseteq}$ contain m points p_1, p_2, ..., p_m. To compute $\mathcal{HD}(1, \mathcal{D})$, for each p_i ($1 \leq i \leq m$), we obtain $m - 1$ half-planes in the form of $pl(\lambda) > 0$ by defining pl as in equation 3 using p_i and every other point in $min\mathcal{D}_{\subseteq}$. Then, the intersection of these $m - 1$ half-planes includes all the slope vectors for which p_i has the highest score among all points in the original dataset $\mathcal{D}$. We refer to this intersection area as the *valid region* of p_i. Observe that the m valid regions of all objects in $min\mathcal{D}_{\subseteq}$ are disjoint and cover the entire slope space. Since each region is bounded by at most m edges, the total complexity of all regions equals $O(m^2)$.

Therefore, a 3D top-1 search is reduced to a 2D point-location problem. Specifically, we first obtain the 2D slope vector λ of the query, and identify the valid region that contains λ. Then, the point associated with the region is the query result. Using an efficient point-location structure [1], we can answer a top-1 query in $O(log_B(m/B))$ I/Os using $O(m^2/B)$ space.

The problem with this technique is that the space consumption is no longer minimum (i.e., size m of $min\mathcal{D}_{\subseteq}$). The situation is even worse for top-c queries with $c > 1$. In this case, we can generalize the above derivation and prove that any top-c query in the 3D space can be answered in $O(c \cdot log_B(m/B))$ I/Os using $O(m^{c+1}/B)$ space, where m is the size of the minimum c-covering subset $min\mathcal{D}_{\subseteq}$. As c increases, the space consumption may become prohibitive, rendering this solution impractical. In this case, a better method is to answer a query by simply scanning $min\mathcal{D}_{\subseteq}$ without using any structure, as mentioned at the beginning of Section 5.

6 Approximate Ranked Retrieval

In this section, we consider approximate ranked queries formulated in Problem 2. Specifically, given a weight vector w, a top-(k,K) query returns k objects among the K data points with the highest scores for w. Section 6.1 first elaborates the characteristics of (c,C)-covering subsets, based on which Section 6.2 discusses their computation in 2D space. Section 6.3 presents query processing algorithms for all dimensionalities.

6.1 Properties of (c,C)-Covering Subsets

As discussed in Section 4.1, in the 2D space, the relative order of objects' scores is determined by the slope of the corresponding weight vector. Hence, we will characterize a top-(k,K) query using three values: k, K, and the weight slope. Recall that a (c,C)-covering subset covers any top-(k,C) query with $k \leq c$. Specifically, for any value k and weight slope λ, the subset always contains c objects in the top-C set at λ. Among all these subsets, the smallest one is called the *minimum* (c,C)*-covering subset*, represented as $min\mathcal{D}^*_{\subseteq}(c, C, \mathcal{D})$.

It is easy to see that the minimum C-covering subset $min\mathcal{D}_{\subseteq}$ $(C, \mathcal{D})$ (see the previous sections) is a (c,C)-covering subset. In fact, for each weight slope λ, $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$ includes the corresponding top-C set. Hence, a top-(k,C) query with slope λ can be answered by returning any k points in the top-C set. The following lemma reveals the relationship between $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$ and $min\mathcal{D}^*_{\subseteq}(c, C, \mathcal{D})$.

Lemma 8. $min\mathcal{D}^*_{\subseteq}(c, C, \mathcal{D}) \subseteq min\mathcal{D}_{\subseteq}(C, \mathcal{D})$.

The lemma motivates a strategy for computing $min\mathcal{D}^*_{\subseteq}(c, C, \mathcal{D})$. Specifically, we first retrieve $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$, and then eliminate the points of $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$ that are not needed for top-(c,C) processing. The remaining data constitute $min\mathcal{D}^*_{\subseteq}(c, C, \mathcal{D})$. In order to identify the objects necessary for top-(c,C) queries, we resort to an interesting connection between (c,C)-covering subsets and the top-C homogeneous decomposition $\mathcal{HD}(C, \mathcal{D})$, illustrated below.

Assume that $\mathcal{HD}(C, \mathcal{D})$ contains m intervals I_1, I_2, ..., I_m. Each $I_i = [I_{i\vdash}, I_{i\dashv})$, for $1 \leq i \leq m$, is associated with a set $I_i.S$ consisting of the C objects having the highest scores at any weight slope in $[I_{i\vdash}, I_{i\dashv})$. Therefore, any c points in $I_i.S$ form a legal result for a top-(c,C) query whose weight slope falls in $[I_{i\vdash}, I_{i\dashv})$.

For each object $p \in min\mathcal{D}_{\subseteq}(C, \mathcal{D})$, we construct a *legal set* $p.LS$, containing the intervals of $\mathcal{HD}(C, \mathcal{D})$ whose top-C sets involve p. For example, consider Figure 9

where, as discussed at the end of Section 4.3, $min\mathcal{D}_{\subseteq}(2, \mathcal{D})$ involves all the points shown in the figure. $\mathcal{HD}(2, \mathcal{D})$ includes three intervals $I_1 = [0, \lambda_2)$, $I_2 = [\lambda_2, \lambda_3)$, and $I_3 = [\lambda_3, \infty)$, where λ_2 (λ_3) is the slope of ray l_2 (l_3). The top-2 set $I_1.S$ of I_1 equals $\{p_2, p_4\}$, while $I_2.S$ = $\{p_2, p_1\}$, and $I_3.S$ = $\{p_3, p_1\}$. Hence, the legal set $p_1.LS$ of p_1 contains intervals I_2 and I_3 since their top-2 sets include p_1. Similarly, $p_2.LS$ = $\{I_1, I_2\}$, $p_3.LS$ = $\{I_3\}$, and $p_4.LS$ = $\{I_1\}$.

Lemma 9. *Let $\mathcal{D}_{\subseteq}$ be a subset of $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$, and also a (c,C)-covering subset. Then, each interval in $\mathcal{HD}(C, \mathcal{D})$ is included in the legal sets of at least c points in $\mathcal{D}_{\subseteq}$.*

In Figure 9, for instance, $\{p_1\}$ is *not* a (1,2)-covering subset since interval I_3, which is in $\mathcal{HD}(2, \mathcal{D})$, does not belong to $p_1.LS$. Set $\{p_1, p_2\}$, on the other hand, is a (1,2)-covering subset because each interval in $\mathcal{HD}(2, \mathcal{D})$ is in the legal set of either p_1 or p_2 — I_1 is in $p_2.LS$, while I_2 and I_3 belong to $p_1.LS$. Similarly, $\{p_2, p_3\}$ is also a (1,2)-covering subset.

6.2 Computing (c,C)-Covering Subsets

Lemmas 8 and 9 indicate that finding the minimum (c,C)-covering subset is equivalent to extracting the smallest number of points from $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$, such that each interval in $\mathcal{HD}(C, \mathcal{D})$ is included in the legal sets of at least c retrieved points. This is an instance of the *minimum set cover* problem which, unfortunately, is NP-hard [6].

We provide a greedy solution which computes a (c,C)-covering subset, whose cardinality is larger than the minimum size by only a small factor. The pseudo-code of the algorithm, called *find-cCsub*, is presented in Figure 11, which assumes that the legal sets of all points in $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$ have been obtained. *Find-cCsub* maintains a set $\mathcal{D}_{\subseteq}$, which is empty at the beginning of the algorithm, and becomes the produced (c,C)-covering subset at termination. For each interval I in $\mathcal{HD}(c, \mathcal{D})$, we keep a counter $I.cnt$, which equals the number of points in the current $\mathcal{D}_{\subseteq}$ whose legal sets include I. The counter is set to 0 initially.

Find-cCsub executes in iterations. In each iteration, it identifies the point p in $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$ with the largest legal set $p.LS$, and incorporates p into $\mathcal{D}_{\subseteq}$. For each interval I in $P.LS$, the counter $I.cnt$ is increased by 1, reflecting the fact that a point whose legal set includes I has been newly added to $\mathcal{D}_{\subseteq}$. Once $I.cnt$ reaches c, I is removed from the legal set of every point in $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$ — it does not need to be considered in the remaining execution. Finally, the object p inserted to $\mathcal{D}_{\subseteq}$ in this iteration is eliminated from $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$. The algorithm terminates if the legal sets of all remaining data in $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$ are empty. Otherwise, it performs another iteration.

Theorem 7. *The size of the subset returned by the algorithm in Figure 11 is at most $(\ln \gamma + 1)$ times larger than that of the minimum (c,C)-covering subset, where γ is the maximum cardinality of the legal sets of the points in $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$.*

Note that $\gamma = |\mathcal{HD}(C, \mathcal{D})|$ in the worst case, when the legal set of an object in $min\mathcal{D}_{\subseteq}(C, \mathcal{D})$ contains all the intervals in $\mathcal{HD}(C, \mathcal{D})$. Since $|\mathcal{HD}(C, \mathcal{D})| = O(|min\mathcal{D}_{\subseteq}(C, \mathcal{D})|)$ (Theorem 4), the size of the subset produced by *find-cCsub* is larger than that of the minimum (c,C)-covering subset by a factor of at most

Algorithm *find-cCsub* (c, $min\mathcal{D}_{\subseteq}(C,\mathcal{D})$, $\mathcal{HD}(C,\mathcal{D})$)
/* the legal set of every point in $min\mathcal{D}_{\subseteq}(C,\mathcal{D})$ has been computed */
1. $\mathcal{D}_{\subseteq} = \emptyset$; $I.cnt = 0$ for each interval $I \in \mathcal{HD}(C,\mathcal{D})$
2. while (a point in $min\mathcal{D}_{\subseteq}(C,\mathcal{D})$ has a non-empty legal set)
3. p = point in $min\mathcal{D}_{\subseteq}(C,\mathcal{D})$ with the largest legal set
4. $\mathcal{D}_{\subseteq} = \mathcal{D}_{\subseteq} \cup \{p\}$
5. for each interval I in the legal set $p.LS$
6. $I.cnt = I.cnt + 1$
7. if $I.cnt = c$ then remove I from the legal set of every point in $min\mathcal{D}_{\subseteq}(C,\mathcal{D})$
8. $min\mathcal{D}_{\subseteq}(C,\mathcal{D}) = min\mathcal{D}_{\subseteq}(C,\mathcal{D}) - \{p\}$
9. return $\mathcal{D}_{\subseteq}$

Fig. 11. Finding a (c,C)-covering subset

$\ln |min\mathcal{D}_{\subseteq}(C,\mathcal{D})| + 1$. In practice, we expect γ to be much smaller than $|\mathcal{HD}(C,\mathcal{D})|$, and as a result, the subset obtained by *find-cCsub* has a cardinality close to the theoretically minimum value.

Theorem 8. *The algorithm in Figure 11 finishes $O(m^2)$ time, where m is the number of points in $min\mathcal{D}_{\subseteq}(C,\mathcal{D})$.*

6.3 Query Processing

Let $\mathcal{D}_{\subseteq}$ be a (c,C)-covering subset computed by the method in Figure 11. Given a top-(k,C) query ($k \leq c$) with weight slope λ, we simply return the k points in $\mathcal{D}_{\subseteq}$ having the highest scores at λ. Due to the properties of (c,C)-covering subsets, these k objects are guaranteed to be a legal result. The query performance can be optimized using exactly the same techniques as in Section 4, by replacing the original dataset $\mathcal{D}$ with $\mathcal{D}_{\subseteq}$. All the bounds on the execution cost and space consumption in Section 4 are still valid.

The above analysis applies to dimensionality 2. For higher-dimensional space, we can compute the minimum c-covering subset $min\mathcal{D}_{\subseteq}(c,\mathcal{D})$ using the algorithm in Figure 2. Note that $min\mathcal{D}_{\subseteq}(c,\mathcal{D})$ is a (c,C)-covering subset, and hence, can be deployed to substitute the original dataset $\mathcal{D}$ to support top-(k,C) queries ($k \leq c$). This leads to an approach that solves any such query in $O(|min\mathcal{D}_{\subseteq}(c,\mathcal{D})|\ /B)$ I/Os. It is important to note that we computed (c,C)-covering subset in the 2D space based on $min\mathcal{D}_{\subseteq}(C,\mathcal{D})$ (note the capitalized C), as is required by Lemma 8.

7 Conclusions

This paper introduced the concept of "minimum covering subset", which is the smallest subset of the database that must be stored by any ranked-retrieval algorithms to ensure correct results for all queries. For 2D space, we developed a technique that consumes $O(m/B)$ space and solves any top-k query in $O(\log_B(m/B) + c/B)$ I/Os, where c is the upper bound of k, m the size of the minimum c-covering subset, and B the disk page capacity. For higher dimensionality, our approach requires $O(m/B)$ space and query time.

As a second step, we provided the first solutions for approximate ranked retrieval that do not require inspecting the entire database in the worst case. In the 2D scenario, our method occupies $O(m'/B)$ space and solves a top-(k,C) query with $k \leq c$ in $O(\log_B(m'/B)+c/B)$ where m' is larger than the size of the minimum (c,C)-covering subset by a small bounded factor. For higher dimensionality, we showed that a top-(k,C) ($k \leq c$) query can be answered in $O(m/B)$ I/Os and space, where m is the size of the minimum c-covering subset.

This work lays down a foundation for continued investigation of ranked queries. A promising direction for future work is to study faster algorithms for computing the minimum covering subsets, utilizing the properties presented earlier. Another interesting problem is the dynamic maintenance of minimum subsets at the presence of data updates. Specialized approaches may be developed to explore the tradeoff between update efficiency and the space overhead.

Acknowledgement

This work was fully supported by RGC Grant CityU 1163/04E from the government of HKSAR.

References

1. L. Arge, A. Danner, and S.-M. Teh. I/o-efficient point location using persistent b-trees. In *ALENEX*, pages 82–92, 2003.
2. J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, 1978.
3. M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2000.
4. Y.-C. Chang, L. D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The onion technique: Indexing for linear optimization queries. In *SIGMOD*, pages 391–402, 2000.
5. S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *VLDB*, pages 397–410, 1999.
6. T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.
7. D. Donjerkovic and R. Ramakrishnan. Probabilistic optimization of top n queries. In *VLDB*, pages 411–422, 1999.
8. V. Hristidis, N. Koudas, and Y. Papakonstantinou. Prefer: a system for the efficient execution of multi-parametric ranked queries. In *SIGMOD*, pages 259–270, 2001.
9. V. Hristidis and Y. Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. *The VLDB Journal*, 13(1):49–70, 2004.
10. A. Marian, N. Bruno, and L. Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, 2004.
11. P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava. Ranked join indices. In *ICDE*, 2003.