k-Regret Minimizing Set: Efficient Algorithms and Hardness

Wei Cao[1], Jian Li[1], Haitao Wang[2], Kangning Wang[1], Ruosong Wang[1], Raymond Chi-Wing Wong[3], and Wei Zhan[1]

1 **Institute for Interdisciplinary Information Sciences, Tsinghua University, China**
{cao-w13,wkn13,wrs13,zhan-w13}@mails.tsinghua.edu.cn,lapordge@gmail.com
2 **Utah State University, Logon, Utah, USA, haitao.wang@usu.edu**
3 **The Hong Kong University of Science and Technology, Hong Kong, China**
raywong@cse.ust.hk

Abstract

We study the k-regret minimizing query (k-RMS), which is a useful operator for supporting multi-criteria decision-making. Given two integers k and r, a k-RMS returns r tuples from the database which minimize the *k-regret ratio*, defined as one minus the worst ratio between the k-th maximum utility score among all tuples in the database and the maximum utility score of the r tuples returned. A solution set contains only r tuples, enjoying the benefits of both top-k queries and skyline queries. Proposed in 2012, the query has been studied extensively in recent years. In this paper, we advance the theory and the practice of k-RMS in the following aspects. First, we develop efficient algorithms for k-RMS (and its decision version) when the dimensionality is 2. The running time of our algorithms outperforms those of previous ones. Second, we show that k-RMS is NP-hard even when the dimensionality is 3. This provides a complete characterization of the complexity of k-RMS, and answers an open question in previous studies. In addition, we present approximation algorithms for the problem when the dimensionality is 3 or larger.

1998 ACM Subject Classification H.2.4 Query processing.

Keywords and phrases multi-criteria decision-making, regret minimizing set, top-k query

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.23

1 Introduction

One major task of a database system is to return "representative" records to a user. Usually, there are two goals in the system. The first goal is to return a *limited* number of records to a user when the utility function of this user is *known*. One query type achieving this goal is top-k queries [14, 15, 20, 29, 30, 34], each returning k records that have the greatest scores calculated based on these k records and the utility function of a user where k is a positive integer. Interested readers may refer to [18] for a survey of top-k queries. The second goal is to return a set of records which are interesting to a user even though his/her utility function is *unknown*. One example of a query type for this goal is skyline queries [3, 4, 15, 20, 22, 24, 31], each returning a set of records from the database each of which is not *dominated* by other records in the database. Here, a record x is said to *dominate* another record x' if and only if each attribute value of x is not worse than that of x' and at least one attribute value of x is better than that of x'. Interested readers may also refer to [9] for a survey of skyline queries.

However, as described in [25, 26, 27], the above two popular queries could not achieve these two goals simultaneously. First, a top-k query does not achieve the second goal since

The 20th International Conference on Database Theory (ICDT 2017).
Editors: John Q. Open and Joan R. Acces; Article No. 23; pp. 23:1–23:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

it requires that a user is given an *exact* utility function indicating his/her preference, which is not reasonable in some cases because in many situations, the user does not know how to specify his/her exact utility function. Second, a skyline query does not meet the first goal because it returns an *uncontrolled* number of records. In the worst case, all records in the database are returned as an output in a skyline query.

Recently, *r-regret queries* and *k-regret minimizing set (k-RMS)* queries, two new types of queries meeting the above two goals, were proposed [8, 25, 26, 27] and studied extensively due to its usefulness and its wide applicability, where r and k are two positive integers. All applications originally applied to top-k queries and skyline queries could also be applied to r-regret queries and k-RMS queries. Some typical applications are choosing hotels for vacation and choosing items (e.g., cars) for purchase.

The purpose of an r-regret query is to return a set of r records in the database, minimizing the "unhappiness" level of a user when seeing only these r records instead of all records in the database, even though the utility function of this user is unknown. Given a positive integer r and a database D containing a number of records, an r-regret query is to return a set R of r records from D such that the greatest "unhappiness" level of a user, formally called the *maximum regret ratio* of a user, is minimized when the user sees only records in R. Here, the "unhappiness" level of a user, called the *regret ratio* of a user, ranging from 0 to 1, refers to how unhappy the user would be when seeing only the records in R, instead of all records in D. Consider the user with his/her utility function f. The *score* of a record x in D with respect to the utility function f is denoted by $f(x)$. The greater the score of a record is, the better the record is. Given a set R of records, the *best* record in R with respect to the utility function f is defined to be the record in R with its greatest score with respect to the utility function f. The regret ratio of this user is equal to 0 if the score of the best record in the selection set R is equal to the one in the whole database D. It becomes larger if the score of the best record in R is smaller than the one in D. The maximum regret ratio of a user refers to the greatest possible regret ratio of a user (since different users can have different utility functions).

Recently, Chester et al. [8] proposed a *generalized* version of r-regret queries called the *k-regret minimizing set (k-RMS)* problem (or queries) relaxing the concept of the "best" record to the concept of *the best k records.* The original form of an r-regret query assumes that a user must be satisfied with only the "best" record in D. Chester et al. [8] relaxed this assumption and considered that a user is already satisfied and "happy" with one of the best k records in D. Specifically, given two positive integers r and k, and a database D containing a number of records, the k-RMS problem is to return a set R of r records from D such that the *maximum k-regret ratio* of a user is minimized. Here, the *k-regret ratio* of a user, ranging from 0 to 1, is equal to 0 if the score of the best record in R is at least the score of the k-th best record in D. It becomes larger if the score of the best record in R is smaller than the score of the k-th best record in D. Clearly, when $k = 1$, k-RMS becomes the r-regret query (called 1-RMS, or simply the RMS problem). In this paper, we have the following contributions.

1. For RMS in $\mathbb{R}^2$ (i.e., the dimensionality is 2), we propose an $O(n \log n)$ time exact algorithm, where n is the number of records in the dataset D. The time complexity is better than the previous best-known time complexity of $O(rn^2 + n^2 \log n)$ [8].
2. For k-RMS in $\mathbb{R}^2$, we present an $O(n^2 \log n)$ time algorithm, which improves the $O(rn^2 k^{\frac{1}{3}} + n^2 \log n)$ time complexity result in [8]. We also propose an approximation algorithm of $O(nk^{\frac{1}{3}} \log(1/\epsilon) + n \log n + nk^{\frac{1}{3}} \log^{1+\delta} n)$ time, where ϵ is the additive approximation error and δ is any positive constant. For typical parameters, it performs

much faster than the previous best-known algorithm [8]. To solve the problem, we also give an efficient algorithm for the decision version of the problem, which is interesting in its own right both theoretically and practically. A summary of our results is in Table 1.

3. We show that for any positive integer k, the k-RMS problem is NP-hard when the dimensionality of the database is 3 (or larger). This is the first-known hardness result for the k-RMS problem in a fixed dimensional database. Although Chester et al. [8] prove the NP-hardness of the RMS problem, it states that the hardness is due to both the *high* dimensionality of the dataset and the *large* number of records in the dataset. It has been open whether the problem is still NP-hard for fixed dimensional cases. Our result settles the open problem and thus provides a complete characterization of the computational complexity of the problem (together with our algorithms in $\mathbb{R}^2$).
4. For RMS in $\mathbb{R}^d$, we show it is closely connected to the notion of *ϵ-kernel*, introduced by Agarwal et al. [2]. Based on the connection, we derive an upper bound $r^{-2/(d-1)}$ of the maximum regret ratio, improving the previous bound $r^{-1/(d-1)}$ in [26]. We also provide an approximation algorithm for k-RMS when $d \geq 3$.

Outline: The rest of the paper is organized as follows. In Section 2, we formally define the problem. Section 3 gives our algorithms for k-RMS when the dimensionality is 2. Section 4 presents the NP-hardness result. Section 5 gives our algorithms in high-dimensional cases. Section 6 discusses the related work. Due to the page limitation, many details and proofs are omitted but can be found in the full version of this paper.

2 Problem Formulation

Let D be a database containing n records/points[1] with d attributes/dimensions. Given a point x in D, for each $i \in [1, d]$, the i-th dimensional value of point x is denoted by $x[i]$. We assume that the values $x[i]$ in the database are all non-negative real numbers, which is the common assumption in related literatures [8, 26]. Each user is associated with a utility function f denoted by a d-dimensional non-negative vector ω called a *weight vector*. Let W be the set of all possible weight vectors.

Given a point x in D and a weight vector ω, the *score* of x with respect to ω is the dot product of x and ω, denoted by $\langle x, \omega \rangle$. That is, $\langle x, \omega \rangle$ is equal to $\sum_{i=1}^{d} x[i]\omega[i]$. If we know the utility function f with the weight vector ω, this score $\langle x, \omega \rangle$ is also written as $f_\omega(x)$. Given an integer $k \geq 1$, we denote the k-th largest score among $x \in D$ with respect to weight vector ω by $\max^{(k)}_{x\in D}\langle x, \omega \rangle$.

Given a non-empty subset R of D and a weight vector ω, the *k-regret ratio* of set R with respect to weight vector ω, denoted by k-regratio(R, ω), is defined to be

$$k\text{-regratio}(R, \omega) = \max\left\{0, 1 - \frac{\max_{x\in R}\langle x, \omega \rangle}{\max^{(k)}_{x\in D}\langle x, \omega \rangle}\right\}.$$

If k-regratio(R, ω) is 0, the best score in R is at least as good as the k-th largest score with respect to ω in the original dataset D. The *maximum k-regret ratio* of R, denoted by k-regratio(R), is defined to be k-regratio$(R) = \sup_{\omega\in W} k$-regratio$(R, \omega)$.

▶ Problem 1 (k-RMS [8]). Given two positive integers k and r, we want to find a set R of r points from D such that k-regratio(R) is minimized.

[1] In the following, we use term "records" and "points" interchangeably since they refer to the same concept.

Problem	Algorithm	Time Complexity	Det	Ex	Source
Dec-RMS	D-IntCov-1	$O(n\log n)$	yes	yes	Sect. 3.1.1
Dec-k-RMS	D-Greedy-k	$O(n+m)$ [†]	yes	yes	Sect. 3.2
RMS	E-Pre-1	$O(rn^2+n^2\log n)$ [††]	yes	yes	[8]
	A-IntCov-1	$O(n\log n\log(1/\epsilon))$	yes	no	Sect. 3.1.2
	A-Greedy-k	$O(n\log n+n\log(1/\epsilon))$	yes	no	Sect. 3.2
	E-Greedy-1	$O(n\log n)$	no	yes	Sect. 3.3.2
k-RMS	E-Pre-k	$O(rn^2k^{\frac{1}{3}}+n^2\log n)$ [††]	yes	yes	[8]
	A-Greedy-k	$O((n+m)\log(1/\epsilon)+n\log n+m\log^{1+\delta} n)$	yes	no	Sect. 3.2
	E-Greedy-k	$O(n^2\log n)$	yes	yes	Sect. 3.3.1

Table 1 The running times of the previous algorithms and our new algorithms in $\mathbb{R}^2$. Det: Deterministic/randomized algorithms (yes/no); Ex: Exact/approximation algorithms (yes/no); The naming of the algorithms: D- means the decision version, E- means an exact algorithm and A- means an approximation algorithm. $n=|D|$, $m=|\mathsf{LS}_k|$, $r=|R|$. ϵ is the additive error of the approximate regret ratio. δ can be any positive constant. [†] D-Greedy-k requires $O(n\log n+m\log^{1+\delta} n)$ preprocessing time, and runs in $O(n+m)$ time for any threshold θ. [††] In [8], the authors claim their algorithm runs in $O(rn^2)$ time, with the factor depending on k omitted as a constant. However, a more careful examination shows their algorithm runs in $O(rn^2+n^2\log n)$ time for RMS and $O(rn^2k^{\frac{1}{3}}+n^2\log n)$ time for k-RMS instead: The priority queue requires $O(n^2\log n)$ time; the best known upper bound of the size of the k-level set is $O(nk^{\frac{1}{3}})$.

This is an *optimization* problem. The following defines its *decision* version, called Dec-k-RMS (we also use Dec-RMS to refer to the case $k=1$).

▶ Problem 2 (Dec-k-RMS). Given two positive integers k and r, and a real value $\theta\in[0,1]$, determine whether there exists a set R of r points from D such that k-regratio(R) is at most $1-\theta$ (if yes, find such a solution set R).

We will also give algorithms for Dec-k-RMS since they will be used as subroutines for solving the optimization version (i.e., Problem 1). On the other hand, in some applications where there is a pre-specified error threshold of k-regratio$(\cdot)$, it would be more suitable to solve the decision version, and thus the decision problem may be interesting in its own right.

3 Efficient Algorithms in $\mathbb{R}^2$

In this section, we develop several algorithms for RMS and k-RMS in $\mathbb{R}^2$. Table 1 summarizes our results. We assume that $\|\omega\|_1=\omega[1]+\omega[2]=1$ for any weight vector $\omega\in W$ as scaling does not change the k-regratio. Hence, we can write $\omega=(\lambda,1-\lambda)$ for some $\lambda\in[0,1]$.

For each point $p=(x,y)$ in D, we define a linear function $f_p(\lambda)=\lambda x+(1-\lambda)y$ with $\lambda\in[0,1]$. We reformulate both the decision version and the optimization version as follows.

1. (The decision version) In the decision version Dec-k-RMS, we are given a constant θ, and we need to decide whether there is some set $R\subseteq D$ of cardinality r such that the following holds:

$$\forall\lambda\in[0,1],\ \max_{p\in R} f_p(\lambda)\geq\theta\cdot\max{}^{(k)}_{p\in D} f_p(\lambda), \tag{1}$$

   where $\max^{(k)}$ is the operator that returns the k-th largest value.
2. (The optimization version) The optimization version k-RMS is to maximize θ in (1).

It is convenient to view the problem from a geometric perspective as follows. Each record $p\in D$ corresponds to a line $f_p(\lambda)$ ($\lambda\in[0,1]$). All such lines form a line arrangement $\mathbb{A}(D)$.

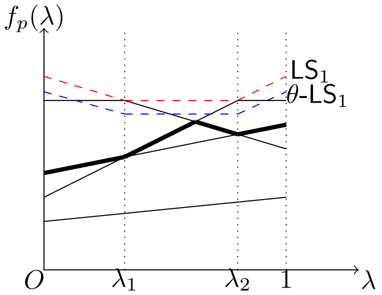


Figure 1 The arrangement $\mathbb{A}(D)$ (black lines), its 1-level set LS_1 (red dashed line), θ-scaled 1-level set (blue lines), and 3-level set (thick black line)

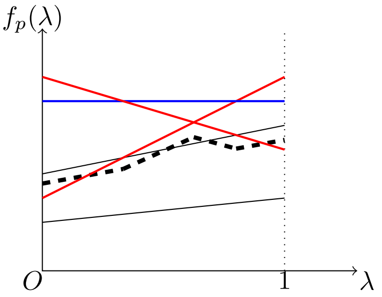


Figure 2 Illustrating Example 2: θ-LS_k is the dashed thick curve

The *k-level set* of $\mathbb{A}(D)$ [5] is a piecewise linear curve (see Figure 1 for an example)

$$\mathsf{LS}_k(\lambda) = \max\nolimits^{(k)}_{p\in D} f_p(\lambda), \text{ for } \lambda \in [0,1].$$

A *segment* is a maximal linear piece in the k-level set. Let m denote the number of segments of $\mathsf{LS}_k(\lambda)$. It is known that m is bounded by $O(nk^{\frac{1}{3}})$ and $\mathsf{LS}_k(\lambda)$ can be computed in $O(n\log n + nk^{\frac{1}{3}})$ expected time by a randomized algorithm [5] or in $O(n\log m + m\log^{1+\delta} k)$ time by a deterministic algorithm for any constant $\delta > 0$ [5]. Note that the 1-level set LS_1 is always *convex* since it is the *upper envelop* of $\mathbb{A}(D)$ (see the red-dashed curve in Figure 1). However, the convexity property does not necessarily hold for any $k > 1$.

We introduce *scaled level sets*, which generalizes the notion of k-level sets.

▶ **Definition 1.** (Scaled Level Set) Given a threshold $\theta > 0$, define the θ-scaled k-level set as the function θ-$\mathsf{LS}_k(\lambda) = \theta \cdot \mathsf{LS}_k(\lambda) = \theta \cdot \max^{(k)}_{p\in D} f_p(\lambda)$, for $\lambda \in [0,1]$.

We can reformulate the decision problem Dec-k-RMS as follows: Decide whether there exists a subset R of r lines, such that the upper envelop of R covers the scaled level set (i.e., the function $\max_{p\in R} f_p(\lambda)$ is above θ-LS_k).

▶ **Example 2.** As an example of the decision problem Dec-k-RMS shown in Fig. 2, where $k = 3$, $\theta = 0.9$ and the dashed thick curve is θ-LS_k, we aim to find r lines such that they collectively cover the dashed thick curve from above. When $r = 2$, the two thick lines shown in red form a solution. When $r = 1$, the thick blue line is the only valid solution.

In the sequel, we solve the decision problem in Section 3.2. In Section 3.3, we solve the optimization problem, using the algorithms for the decision problem as subroutines. But as warm-ups, we first give some simple but practical algorithms.

3.1 The Warm-up Algorithms

In this section we present algorithms for Dec-RMS and RMS. These algorithms are theoretically not as efficient as the algorithms given later, but they are very simple and practical, and may also provide some directions for the later improved ones.

3.1.1 Reducing Dec-RMS to Interval Coverage

Note that since LS_1 (which is actually the upper envelop of $\mathbb{A}(D)$) is convex, the scaled level set θ-LS_1 is also convex. Also note that $m \le n$ in this case. We can compute LS_1 (and thus θ-LS_1) in $O(n\log n)$ time [16]. We use $\lambda_0 = 0, \lambda_1, \ldots, \lambda_{m-1}, \lambda_m = 1$ to denote all breaking points of θ-LS_1 (see Fig. 1). Our task is to cover θ-LS_1 using at most r lines. For a line f_p, we let $I(p) = \{\lambda \mid f_p(\lambda) \geq \theta\text{-}\mathsf{LS}_1(\lambda)\}$. The convexity of θ-LS_1 implies that $I(p)$ is a closed interval (which may be empty). Moreover, the interval can be computed in $O(\log n)$ time by binary search.

Hence, we can compute the set of intervals $\{I(p)\}_{p \in D}$ in $O(n \log n)$ time.

To solve our problem Dec-RMS, it is sufficient to find a minimum number of intervals in the above set whose union covers the range $[0, 1]$. This can be easily done in $O(n)$ time by a greedy method after the endpoints of the intervals of $\{I(p)\}_{p \in D}$ are sorted [1].

We call the above algorithm D-IntCov-1.

▶ **Theorem 3.** *D-IntCov-*1 *solves the* Dec-RMS *problem in* $O(n \log n)$ *time and* $O(n)$ *space deterministically.*

3.1.2 An Approximating Algorithm for RMS

To solve the optimization problem RMS, the high-level idea is to perform binary search on a set of "candidate values" for the optimal regret ratio θ, and use our decision algorithms to check whether θ-LS_1 can be covered by r lines from D.

We simply perform binary search directly on the interval $[0, 1]$. Initially, the candidate range of θ is $[0, 1]$. Given $\theta \in [0, 1]$, we run the decision algorithm for Dec-RMS to check whether the regret ratio of $1 - \theta$ is achievable (i.e., whether θ-LS_1 can be covered). If the answer is yes (resp., no), then we say that θ is *feasible* and the optimal value is at least θ (resp., smaller than θ). We stop until the interval for the candidate θ values is shorter than ϵ, a given tolerable error. The decision procedure is evoked for $O(\log \frac{1}{\epsilon})$ times and the regret ratio of solution is at most the optimal regret plus ϵ (we call such a solution an ϵ-approximation). We refer to the algorithm as A-IntCov-1 (using D-IntCov-1 as the decision procedure). We have the following theorem.

▶ **Theorem 4.** *For any* $\epsilon > 0$*, A-IntCov-*1 *can find an* ϵ*-approximation for RMS in* $O(n \log n \log \frac{1}{\epsilon})$ *time.*

3.2 The Decision Algorithm for Dec-k-RMS

In this section, we present an algorithm for the problem Dec-k-RMS (and thus also for Dec-RMS). We call our algorithm D-Greedy-k. We will prove the following theorem.

▶ **Theorem 5.** *After* $O(n \log n + m \log^{1+\delta} k)$*-time preprocessing for any* $\delta > 0$*, given any* $\theta \in [0, 1]$*, our algorithm D-Greedy-k solves the problem* Dec-k-RMS *in* $O(n+m)$ *time, where* n *is the number of lines of* D *and* m *is the number of segments in the* k*-level set* LS_k*.*

Due to the page limitation, we ignored proofs for some lemmas and some implementation details, which can be found in the full version of this paper.

Given any $\theta > 0$, D-Greedy-k first finds a smallest subset $R \subseteq D$ of lines such that the upper envelop of R is above θ-LS_k for $\lambda \in [0, 1]$ and then solves Dec-k-RMS by comparing $|R|$ with the given maximum cardinality r.

As preprocessing, we sort all lines of D by their intersections with the vertical line $\lambda = 0$ from top to bottom. This step can be done in $O(n \log n)$ time. Then we compute LS_k in $O(n \log m + m \log^{1+\delta} k)$ time for any $\delta > 0$, using the algorithm in [5]. The total preprocessing time is $O(n \log n + m \log^{1+\delta} k)$ (since $m = O(nk^{\frac{1}{3}})$ and $k \leq n$).

After the preprocessing, for any given θ, we first compute θ-LS_k, which can be done in $O(m)$ time since LS_k has been computed in the preprocessing step. Let $l_1, l_2, \ldots, l_n$ be the

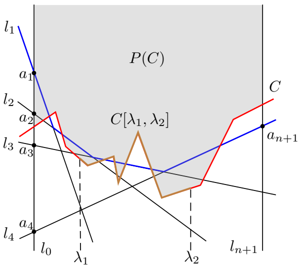


Figure 3 The blue curve denotes $U(\{l_1, l_2, l_3, l_4\})$. The brown curve denotes $C[\lambda_1, \lambda_2]$. The gray area denotes $P(C)$.

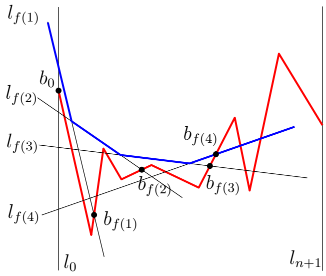


Figure 4 Illustrating the set $R_i = \{l_0, l_{f(1)}, l_{f(2)}, l_{f(3)}, l_{f(4)}\}$ with $g_i = 4$. The red curve is C and the blue curve is $U(R_i)$. The point p is at $b_{f(4)}$.

lines of D sorted by their intersections with the vertical line $\lambda = 0$ from top to bottom. For each $i \in [1, n]$, let a_i denote the intersection of l_i and the vertical line $\lambda = 0$, and thus a_i is also from top to bottom. For convenience, we use l_0 to denote the vertical line $\lambda = 0$ and l_{n+1} to denote the vertical line $\lambda = 1$. A simple observation, as formalized in Lemma 6, is that for two lines l and l', if l "dominates" l', then l' can be directly discarded.

▶ **Lemma 6.** *For any $1 \le i < j \le n$, if the slope of l_i is larger than or equal to that of l_j, then there exists an optimal solution R that does not contain l_j (and thus l_j can be ignored for solving the problem).*

Notice that Lemma 6 essentially states that points that are not in the skyline (in the original space) can be dropped. This has already been known in previous works (e.g., [8]).

We run a *pruning procedure* on D to remove such lines l_j as specified in the preceding lemma. This can be done by scanning the lines of D in their index order in $O(n)$ time. The following algorithm will work on the remaining lines of D. Hence, after the pruning procedure, we can assume that the remaining lines of D following their index order are also sorted by their slopes in strictly ascending order (renamed as $\{l_1, l_2, ...l_n\}$).

To simplify the notation, we use C to refer to θ-LS_k. For any two values λ_1 and λ_2 with $\lambda_1 \le \lambda_2$, we use $C[\lambda_1, \lambda_2]$ to denote the portion of C defined on the interval $\lambda \in [\lambda_1, \lambda_2]$. For any two points q_1 and q_2 on C, we also use $C[q_1, q_2]$ to refer to the portion of C between q_1 and q_2. Let $P(C)$ denote the region of the plane above C and between l_0 and l_{n+1}. For any set D' of lines, we use $U(D')$ to denote its upper envelop. For any point q in the plane, let $\lambda(q)$ denote its λ-coordinate. Note that C is λ-monotone, i.e., any vertical line intersects C at most once. Therefore, we can say something like "move a point on C from left to right". Let C' be another λ-monotone curve in the plane. We say that C' is *above* $C[\lambda_1, \lambda_2]$ for some $\lambda_1 \le \lambda_2$ if for any value $\lambda' \in [\lambda_1, \lambda_2]$, the intersection of the vertical line $\lambda = \lambda'$ and C' is not lower than that of the vertical line $\lambda = \lambda'$ and C. For two points $p_1 \ne p_2$, we use $\overline{p_1 p_2}$ to denote the line segment with endpoints p_1 and p_2. See Figure 3 for an illustration of the definitions given above.

Our algorithm processes the lines of $l_0, l_1, \ldots, l_{n+1}$ in their index order from l_0 to l_{n+1}. In general, suppose line l_i has just been processed and we are about to process l_{i+1} for some i with $0 \le i \le n$. Our algorithm maintains a set $R_i = \{l_{f(0)}, l_{f(1)}, \ldots, l_{f(g_i)}\}$ of $g_i + 1$ lines in $\{l_0, l_1, l_2 \ldots, l_i\}$ and a set $B_i = \{b_{f(0)}, b_{f(1)}, \ldots, b_{f(g_i)}\}$ of $g_i + 1$ points with $f(0) < f(1) < \cdots < f(g_i)$, for some integer $g_i \ge 0$, such that R_i and B_i have the following properties. Refer to Figure 4 for an example.

1. $f(0) = 0$, i.e., $l_{f(0)}$ is l_0. Since l_0 is vertical, we tilt it slightly such that it has a negative slope so that the definition of the upper envelop $U(R_i)$ is clear. Similarly, we tilt l_{n+1}

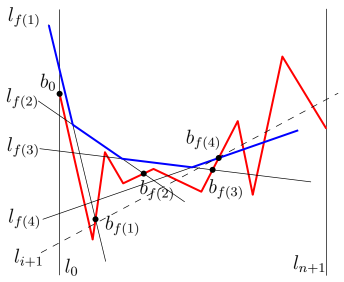


Figure 5 Illustrating the *special case*. Notice that $l_{i+1} \cap \overline{a_{f(4)}b_{f(4)}} = b_{f(4)}$.

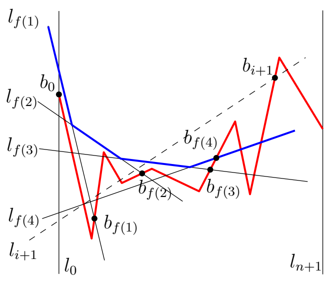


Figure 6 Illustrating the case where l_{i+1} (the dashed line) intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$ with $g_i = 4$. In this example, $R_{i+1} = \{l_0, l_{f(1)}, l_{f(2)}, l_{i+1}\}$ because l_{i+1} intersects $\overline{a_{f(1)}b_{f(1)}}$ but is not above $C[b_{f(1)}, b_{f(2)}]$.

slightly such that it has a positive slope.

2. Each line of R_i has a segment that appears in $U(R_i)$. The segments of lines of R_i appear on $U(R_i)$ from left to right following the index order.
3. $0 = \lambda(b_{f(0)}) < \lambda(b_{f(1)}) < \ldots < \lambda(b_{f(g_i)})$. For $0 \leq t \leq g_i$, $b_{f(t)} \in l_{f(t)} \cap C$. Recall that a_i denote the intersection of l_i and the vertical line $\lambda = 0$, thus the line segment $\overline{a_{f(t)}b_{f(t)}}$ is segment of the line $l_{f(t)}$. For $0 < t \leq g_i$, l_t is above $C[b_{f(t-1)}, b_{f(t)}]$. Thus, $U(R_i)$ is above $C[0, \lambda(b_{f(g_i)})]$.
4. For each line $l_{f(t)} \in R_i$ with $0 \leq t \leq i$, the point $b_{f(t)}$ is defined as follows.
When $t = 0$, $b_{f(t)}$ (i.e., b_0) is defined as the intersection of C and l_0. Here, for convenience of discussion, we also let C include the half-line of l_0 above b_0 and the half-line of l_{n+1} above a_{n+1} (i.e., the intersection of C and l_{n+1}). In this way, C is the boundary of the region $P(C)$.
When $t > 0$, suppose point $b_{f(t-1)}$ has been defined on C. Denote q to be the intersection of $l_{f(t)}$ and the vertical line through $b_{f(t-1)}$, it holds that q is above $b_{f(t-1)}$. If we move q rightwards on $l_{f(t)}$, then $b_{f(t)}$ is defined as the first point of $P(C)$ we encounter after which q will move out of $P(C)$.
5. $\overline{a_{f(t)}b_{f(t)}}$ intersects $\overline{a_{f(t-1)}b_{f(t-1)}}$ for any t with $1 \leq t \leq i$. For any $2 \leq t \leq i$, either $\overline{a_{f(t)}b_{f(t)}}$ does not intersect $\overline{a_{f(t-2)}b_{f(t-2)}}$, or they intersect but $l_{f(t)}$ is not completely above $C[b_{f(t-2)}, b_{f(t-1)}]$. In Figure 4, although $\overline{a_{f(1)}b_{f(1)}}$ intersects with $\overline{a_{f(3)}b_{f(3)}}$, $l_{f(3)}$ is not completely above $C[b_{f(1)}, b_{f(2)}]$.
6. For each $1 \leq t \leq i$, the upper hull of the convex hull of $C[b_{f(t-1)}, b_{f(t)}]$ is maintained in a linked list $L(b_{f(t-1)}, b_{f(t)})$. More specifically, $L(b_{f(t-1)}, b_{f(t)})$ stores the edges of the upper hull of $C[b_{f(t-1)}, b_{f(t)}]$ from left to right.

3.2.1 The Algorithm

Initially, for $i = 0$, we let $R_0 = \{l_0\}$ and $B_0 = \{b_0\}$. In general, suppose we have processed the line l_i and obtained R_i and B_i. In the following, we describe the algorithm for processing the next line l_{i+1} and obtain the set R_{i+1} and B_{i+1}.

We first check whether l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$. If not, we simply ignore l_{i+1} and let $R_{i+1} = R_i$.

If l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$, we consider a *special case* where $l_{i+1} \cap \overline{a_{f(g_i)}b_{f(g_i)}} = b_{f(g_i)}$ and b_{i+1} is $b_{f(g_i)}$ (e.g., see Fig. 5). If this case happens, then we simply ignore l_{i+1} and let $R_{i+1} = R_i$. To determine whether b_{i+1} is $b_{f(g_i)}$, we check whether we will go outside $P(C)$ after we cross $b_{f(g_i)}$ if we move on l_{i+1} rightwards. Since $b_{f(g_i)}$ is known, we can determine whether this special case happens in $O(1)$ time.

If l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$ and the special case does not happen (e.g., see Figure 6), then we proceed to compute the point b_{i+1} as follows.

As $f(g_i) \leq i < i+1$, a_{i+1} is below $a_{f(g_i)}$. Since l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$ and the special case does not happen, if q is the intersection of l_{i+1} and the vertical line through $b_{f(g_i)}$, then q must be above $b_{f(g_i)}$. Imagine that we move q rightwards on l_{i+1}. As we defined earlier, b_{i+1} is the first point of $P(C)$ we encounter after which q will move out of $P(c)$ (e.g., see Figure 6). As the special case does not happen, $\lambda(b_{i+1}) > \lambda(b_{f(g_i)})$ holds. To find b_{i+1}, we simply move q rightwards on C until we meet an intersection between l_{i+1} and an edge of C.

Next we compute the upper hull of (the convex hull of) $C[b_{f(g_i)}, b_{i+1}]$ and store it in a linked list $L(b_{f(g_i)}, b_{i+1})$. The list $L(b_{f(g_i)}, b_{i+1})$ can be constructed when we compute b_{i+1} by moving q from $b_{f(g_i)}$ to b_{i+1}. Since C is λ-monotone, $L(b_{f(g_i)}, b_{i+1})$ can be constructed in linear time in the number of vertices of $C[b_{f(g_i)}, b_{i+1}]$ (e.g., by Graham's scan).

Finally, we determine the set R_{i+1} as follows. We consider the lines of R_i in the reverse order of their indices. Consider $l_{f(g_i)}$ first. If l_{i+1} does not intersect the line segment $\overline{a_{f(g_i-1)}b_{f(g_i-1)}}$ of $l_{f(g_i-1)}$, we stop the procedure with $R_{i+1} = R_i \cup \{l_{i+1}\}$.

Otherwise, there are further two subcases. We check whether l_{i+1} is above $C[b_{f(g_i-1)}, b_{f(g_i)}]$. To this end, observe that l_{i+1} is above $C[b_{f(g_i-1)}, b_{f(g_i)}]$ if and only if l_{i+1} is above the upper hull of $C[b_{f(g_i-1)}, b_{f(g_i)}]$, which is stored in the list $L(b_{f(g_i-1)}, b_{f(g_i)})$. As we will formalize later in Lemma 7, we can use a *upper hull walking procedure* to efficiently determine whether l_{i+1} is above the upper hull of $C[b_{f(g_i-1)}, b_{f(g_i)}]$.

If l_{i+1} is not above $C[b_{f(g_i-1)}, b_{f(g_i)}]$, then we stop the procedure with $R_{i+1} = R_i \cup \{l_{i+1}\}$. Otherwise, we remove $l_{f(g_i)}$ from R_i and proceed on considering the next line $l_{f(g_i-1)}$. In addition, we perform a *upper hull merge procedure* to merge the two lists $L(b_{f(g_i-1)}, b_{f(g_i)})$ and $L(b_{f(g_i)}, b_{i+1})$ to obtain a single list $L(b_{f(g_i-1)}, b_{i+1})$, representing the upper hull of $C[b_{f(g_i-1)}, b_{i+1}]$. As formalized later in Lemma 7, the merge procedure can be efficiently implemented.

The above processes the line $l_{f(g_i)}$. Processing the next line $l_{f(g_i-1)}$ (and other lines) is done similarly, and we omit the details. Refer to Figure 6 for an example.

The above algorithm may remove some lines from R_i. For ease of reference, we let R'_i be the remaining R_i after the above algorithm and we still use R_i to refer to the original set. After the above algorithm, we have $R_{i+1} = R'_i \cup \{l_{i+1}\}$.

The algorithm finishes once l_{n+1} is processed, after which we will obtain the set R_{n+1}. In the full version of this paper, we show that $R_{n+1} \setminus \{l_0, l_{n+1}\}$ is the optimal solution set R for the problem Dec-k-RMS and the whole algorithm runs in $O(n+m)$ time (excluding the preprocessing). The subsequent lemma state that the upper hull merge and walking procedures can be efficiently implemented. See the full version for details of these two procedures. Note that the efficiency of Lemma 7 relies on that the slopes of the lines of D in their index order are sorted increasingly.

▶ **Lemma 7.** *We can implement the upper hull merge procedure and the upper hull walking procedure such that the total time of the procedure in the entire algorithm is $O(m+n)$.*

By the similar binary search approach as in Section 3.1.2 with D-Greedy-k as the decision procedure instead, we can obtain an approximation algorithm for k-RMS. We refer to this algorithm as A-Greedy-k, whose performance is summarized below.

▶ **Theorem 8.** *For any $\epsilon > 0$, A-Greedy-k can find an ϵ-approximation for k-RMS in $O((n+m)\log(1/\epsilon) + n\log n + m\log^{1+\delta} n)$ time.*

3.3 Optimization Algorithms

In this section, we solve the optimization problems RMS and k-RMS. As in Section 3.1.2, the idea is to perform binary search on the candidate values of the optimal θ, with the regret ratio $1-\theta$. Unlike the algorithm there that only gives approximating result, here we present two exact algorithms. The first algorithm (Section 3.3.1) determines all candidate values implicitly (there are too many such values so we cannot afford to compute them explicitly), and performs binary search on them to find an optimal solution for k-RMS. The second algorithm (Section 3.3.2) exploits the convexity of θ-LS_1 and performs randomized binary search over the candidate values, and it works quite efficiently but only on RMS.

3.3.1 An Exact Algorithm for k-RMS

▶ **Lemma 9.** *The following statements are equivalent:*

1. *A set R covers θ-LS_k for all $\lambda \in [0,1]$.*
2. *A set R covers θ-LS_k for all $\lambda \in X(D) := \{0,1\} \cup \{\lambda \mid \exists\, l_1, l_2 \in D, l_1(\lambda) = l_2(\lambda)\}$.*

Obviously statement (1) implies (2). On the other hand, note that both θ-LS_k and the upper envelop of R are piecewise linear, with all breaking points contained in $X(D)$. Therefore if (2) holds, the upper envelop of R must be above θ-LS_k. Hence, statement (2) implies (1) as well. The lemma thus follows.

The following lemma is a consequence of Lemma 9.

▶ **Lemma 10.** *For k-RMS, the optimal θ is $0, 1$ or in*

$$\mathsf{Cand}(D) := \left\{ \frac{l(\lambda)}{\mathsf{LS}_k(\lambda)} \,\middle|\, l \in D, \lambda \in X(D) \right\}.$$

Proof. Notice that by Lemma 9, θ is optimized so that R covers θ-LS_k within $X(D)$. This implies θ-LS_k and the upper envelop of R coincide at some $\lambda \in X(D)$, so the lemma holds. ◀

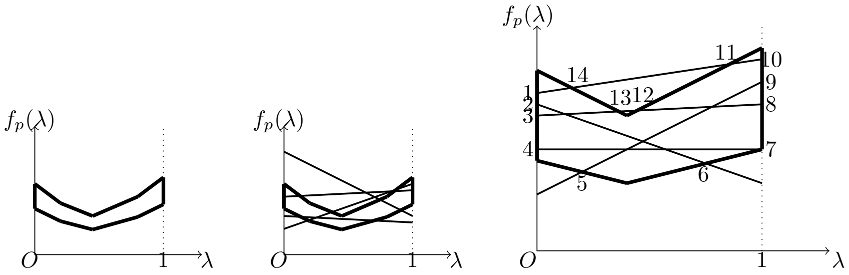
Clearly, the set $\mathsf{Cand}(D)$ consists of at most $|D| \cdot |X(D)| = O(n^3)$ values. To solve the problem k-RMS, we can call the decision algorithm D-Greedy-k to find the largest feasible $\theta \in \mathsf{Cand}(D)$. Computing the set $\mathsf{Cand}(D)$ explicitly would take $\Omega(n^3)$ time. Instead, we present an approach that only constructs $\mathsf{Cand}(D)$ implicitly.

First we compute and sort the set $X(D)$. For each $\lambda \in X(D)$, our algorithm maintains an interval of indices $I_\lambda \subseteq [1,n]$ so that if the optimal value θ is equal to the j-th largest value of $l(\lambda)/\mathsf{LS}_k(\lambda)$ for all $l \in D$, then $j \in I_\lambda$ must hold. Initially, I_λ is set to $[1,n]$ for each $\lambda \in X(D)$, and the interval will shrink during the algorithm.

The algorithm consists of multiple stages. In each stage, we use a line sweeping algorithm on λ, keeping track of the lines l of D ordered by $l(\lambda)$. For the i-th time the sweeping line hits a $\lambda_i \in X(D)$, we compute a value θ_i of θ (according to Lemma 10) determined by the line ranked at the median of the interval I_{λ_i}, and assign it a weight $w_i = |I_{\lambda_i}|$. In this way, we compute a weighted subset $S = \{\theta_i\}_i \subseteq \mathsf{Cand}(D)$ of size $O(n^2)$. Next we compute the weighted median of S: that is, a value $\theta_m \in S$ such that

$$\sum\{w_i \mid \theta_i < \theta_m\} \le \frac{1}{2}\sum_i w_i < \sum\{w_i \mid \theta_i \le \theta_m\}.$$

The weighted median can be found in $O(|S|)$ time using the linear-time selection algorithm [21]. Then we use D-Greedy-k to determine whether θ_m is feasible. If yes, we update



(a) An example of the band $\mathcal{B}$

(b) Each line may provide at most two segments in the band

(c) The sampling method

Figure 7 Illustration of the band, the segments in it, and the sampling method

each I_{λ_i} with $\theta_i \geq \theta_m$ to be its lower-half interval; otherwise, we update each I_{λ_i} with $\theta_i \leq \theta$ to be its upper-half interval. Hence, the reduced weight of all intervals of S is $\frac{1}{2}\sum\{w_i \mid \theta_i \geq \theta\}$ or $\frac{1}{2}\sum\{w_i \mid \theta_i \leq \theta\}$, which is larger than $\frac{1}{4}\sum_i w_i$ in either case. Therefore, each stage will reduce the total weight by at least $1/4$. Since the initial total weight, that is, the total size of all intervals I_λ is $O(n^3)$, we conclude that there are $O(\log n)$ stages (the algorithm stops once the remaining total weight is $O(1)$, after which we can use D-Greedy-k to find the optimal θ from the remaining $O(1)$ candidate values).

For the running time, each stage is comprised of an $O(n^2)$-time line sweeping algorithm, an $O(n^2)$-time weighted median algorithm [21], and one call of D-Greedy-k taking $O(n + m) = O(n^2)$ time. Thus, the total time of the algorithm is $O(n^2 \log n)$. We refer to the algorithm is as E-Greedy-k (for Exact Greedy Algorithm).

▶ **Theorem 11.** *E-Greedy-k can compute an optimal solution for the k-RMS problem in $O(n^2 \log n)$ time.*

3.3.2 An $O(n \log n)$ Time Exact Algorithm for RMS

For RMS, we derive a more efficient randomized algorithm, whose expected running time is $O(n \log n)$. Lemma 10 still applies here, but we have a stronger Lemma 12 (which is not applicable to k-RMS),

▶ **Lemma 12.** *For RMS, the optimal value θ is in the set $\mathsf{Cand}(D)$ defined as*

$$\{0,1\} \cup \left\{ \frac{l(\lambda)}{\mathsf{LS}_1(\lambda)} \,\middle|\, l \in D \textit{ and } \lambda \in \{0,1\}, \textit{ or } \exists\, l' \in D, l(\lambda) = l'(\lambda) \right\}.$$

Proof. For the optimal θ such that a set R covers θ-LS_1, the upper envelop of R and θ-LS_1 must coincide at some point. Let $l \in R$ to be the line whose segment in the upper envelop contains such a coincidence point. Suppose two endpoints of this segment are at λ_1, λ_2. Then at least one of them is also a coincidence point, since otherwise θ-LS_1 would be strictly above the segment at one of λ_1 and λ_2, incurring contradiction. ◀

We partition $\mathsf{Cand}(D)$ into two subsets:

$$\mathsf{Cand}_1(D) = \{0,1\} \cup \left\{ \tfrac{l(\lambda)}{\mathsf{LS}_1(\lambda)} \mid l \in D \text{ and } \lambda \in \{0,1\} \right\}, \mathsf{Cand}_2(D) = \left\{ \tfrac{l(\lambda)}{\mathsf{LS}_1(\lambda)} \mid \exists\, l' \in D, l(\lambda) = l'(\lambda) \right\}.$$

Notice that $|\mathsf{Cand}_1(D)| \leq 2n + 2 = O(n)$. In the following, we first process $\mathsf{Cand}_2(D)$. Our approach is inspired by the random sampling technique of Matoušek [23].

We perform a randomized search over the values of $\mathsf{Cand}_2(D)$, without constructing $\mathsf{Cand}_2(D)$ explicitly. The algorithm consists of multiple rounds and each round shrinks the search range significantly. Initially, the search range of θ is $[0, 1]$. In general, suppose the search range is $[\theta_0, \theta_1]$ in the current round. We define a band $\mathcal{B}$ as the region bounded by θ_0-LS_1, θ_1-LS_1, and the two vertical lines $\lambda = 0$ and $\lambda = 1$ (See Figure 7a). A candidate value $\frac{l(\lambda)}{\mathsf{LS}_1(\lambda)}$ in $[\theta_0, \theta_1]$ corresponds to an intersection $(\lambda, l(\lambda))$ of two lines of D lying in the band $\mathcal{B}$. The current round of the algorithm works as follows.

We first compute the number of line intersections in $\mathcal{B}$ and then sample at most n out of them. Note that the intersection of each line $l \in D$ and $\mathcal{B}$ consists of at most two (maximal) segments, and each segment has two endpoints on the boundary of $\mathcal{B}$. Thus there are at most 4 such endpoints on each line $l \in D$, and the total number of endpoints is $O(n)$. (See Fig. 7b). These endpoints can be calculated in $O(\log n)$ time for each line due to the convexity of LS_1. We then sort the $O(n)$ endpoints on the boundary $\mathcal{B}$ in counterclockwise order (See Fig. 7c), and for the i-th endpoint, we use s_i to denote the segment ending at it. We traverse these endpoints in order along the boundary of $\mathcal{B}$. During the traversal, we maintain an ordered list L, and a counter N. For the i-th endpoint, let i' be the index such that $s_i = s_{i'}$. If i' is not in L, then we add i to L; otherwise we delete i' from L and increase N by the size of the set $\{j \in L \mid j > i'\}$.

▶ **Lemma 13.** *After the traversal, the counter N is the number of candidates of $\theta = \frac{l(\lambda)}{\mathsf{LS}_1(\lambda)}$ in $[\theta_0, \theta_1]$.*

Proof. For each $\lambda \in \mathsf{Cand}_2(D)$ which is determined by an intersection of two lines $l(\lambda) = l'(\lambda)$ for $l, l' \in D$, the corresponding segments in $\mathcal{B}$ must have four endpoints with indices $i' < j < i < j'$ where i and i' belong to l while j and j' belong to l'. Thus, we will increase the counter N for exactly one time, i.e., when i' is deleted from L and we find that $j < i'$. Therefore, there exists a bijection between all candidate values and all increments of the counter N. So the lemma holds. ◀

▶ **Example 14.** As an example, consider the instance presented in Figure 7c. The endpoints of segments are labeled counterclockwise. Starting from endpoint 1, we in turn add $1, 2, 3, 4, 5$ into the list L. Then for endpoint 6, since $s_2 = s_6$ the list becomes $\{1, 3, 4, 5\}$, and N increases by 3. For endpoint 7, we delete 4 from the list and increase N by 1.

The above computes the number of candidates N. We assume $N > n$. Next, we uniformly and randomly pick n candidates out of the N candidate values in $[\theta_0, \theta_1]$. To this end, we first uniformly (with replacement) pick a set S of n indices from $\{1, \ldots, N\}$. Then we compute the candidate values corresponding to these picked indices, which can be done by doing the above traversal again. Specifically, during the traversal, suppose that after processing an endpoint i, the counter N grows from N_0 to N_1. Then we add the following candidate values: for every index $k \in (N_0, N_1] \cap S$, we add the candidate value determined by the intersection of the segment s_i and the segment corresponding to the $(k - N_0)$-th largest endpoint in the current ordered list L maintained during the traversal.

The above (at most) n candidate values of θ can be regarded as uniform samples from all candidate values in the search range $[\theta_0, \theta_1]$. We sort and perform a binary search on these values using the decision procedure D-Greedy-k (with $k = 1$) to find two adjacent values θ'_1 and θ'_2 in the sorted list such that the optimal θ value is in the range $(\theta'_0, \theta'_1]$. Then, we shrink the range $[\theta_0, \theta_1]$ by updating $\theta_0 = \theta'_0$ and $\theta_1 = \theta'_1$, and proceed to the next round.

We proceed as above until there are at most n candidate values in the search range (i.e., $N \leq n$). Finally, we run the following *post-processing step.* Let U be the union of the set of these at most n candidate values in the search range and $\mathsf{Cand}_1(D)$. By the above algorithm, the optimal value θ is in U. Since $|\mathsf{Cand}_1(D)| = O(n)$, $|U| = O(n)$. We sort and perform a binary search on the values of U using the decision procedure D-Greedy-k to eventually compute the optimal θ value. This finishes our algorithm.

To analyze the running time, it is easy to see that the post-processing step takes $O(n \log n)$ time. Below, we analyze the algorithm before the post-processing step.

We first consider the running time for each round. In each round, the algorithm computes and sorts the line segment endpoints on the boundary of $\mathcal{B}$, in $O(n \log n)$ time. Maintaining the list L for a traversal of $O(n)$ endpoints can be done in $O(n \log n)$ time using a balanced binary search tree. Sorting and doing the binary search on the sampled n values takes $O(n \log n)$ time, since the procedure D-Greedy-k is called $O(\log n)$ times. Thus, the total running time of each round is $O(n \log n)$.

Next, we show that the algorithm has a constant $p^* > 0$ probability to proceed into post-processing within two rounds. Indeed, let $p(n_0, n_1)$ denote the probability of reducing the size of the candidate set from n_0 to at most n_1 in one round. We can obtain that $p(n_0, n_1) \geq 1 - n_0 \cdot \left(\frac{n_0 - n_1}{n_0}\right)^n$, because the size after the round is larger than n_1 only if our algorithm did not pick any indices from some interval $[i, i + n_1)$ (here the indices i refer to the θ values of the candidate set after they are sorted). For large enough n, we can see that $p^* = p(n^2, n^{3/2}) \cdot p(n^{3/2}, n) \geq (1 - n^2 e^{-\sqrt{n}})^2$ is close to 1. Suppose the algorithm terminates at round t (t is a random variable). For any $r \geq 3$, it holds that $\Pr[t \geq r] \leq (1 - p^*)^{r-2}$. Overall, the expected number of rounds is $\mathbb{E}[t] \leq 2 + \sum_{r=3}^{\infty} \Pr[t \geq r] \leq 2 + \sum_{r=3}^{\infty} (1 - p^*)^{r-2} = O(1)$.

By the above analysis, our algorithm, named E-Greedy-1, has the following performance.

▶ **Theorem 15.** *E-Greedy-1 solves RMS in $O(n \log n)$ expected time.*

4 NP-Hardness

The main results of this section are the NP-hardness results in Theorem 16 and Theorem 17.

▶ **Theorem 16.** *Dec-RMS is* NP*-hard even in $\mathbb{R}^3$.*

Note that Dec-RMS in $\mathbb{R}^d$ for $d > 3$ generalizes Dec-RMS in $\mathbb{R}^3$ (by adding a few dummy dimensions). Further, by duplicating each point k times in an Dec-RMS instance, we can create a Dec-k-RMS instance with exactly the same optimal solution as the Dec-RMS instance. This implies the following hardness result.

▶ **Theorem 17.** *Dec-k-RMS is* NP*-hard for any fixed $k \geq 1$ and $d \geq 3$.*

In the following, we will prove Theorem 16, by a reduction from the vertex cover problem on a special planar graph, defined as follows.

A *planar straight-line graph (PSLG)* [28] is a graph $G = (V, E)$ where V is a finite subset of $\mathbb{R}^2$, and E is a subset of mutually disjoint open line segments with both endpoints in V. A *face* of a PSLG is a connected component of $\mathbb{R}^2 \setminus (V \cup E)$. The unique unbounded face is called the *outer face*, and all others are called *inner faces*. Similarly, vertices on the boundary of the outer face are called *outer vertices*, and all others are called *inner vertices*. Notice that in a PSLG, every inner face is an open polygon, and thus for every inner vertex with degree t, there are t interior angles attached to it. We say that a PSLG is *convex* if every inner face is convex, and the complement of the outer face is also convex.

Given an undirected graph $G = (V, E)$, a *vertex cover* is a subset of vertices $S \subseteq V$ that cover all edges in E (i.e., every edge in E is incident on some vertex in S). The *vertex cover*

(VC) problem asks for a vertex cover of minimum cardinality. Das and Goodrich [10] showed that the VC problem on a convex PSLG is NP-hard, and below we do the reduction to prove Theorem 16.

We first define an intermediate problem, called the *inner vertex cover problem* (IVC), on a class of so-called *normalized PSLG graphs*, and provide a reduction from VC convex PSLG to it in Section 4.1. Then, we further reduce the problem to Dec-RMS in Section 4.2.

4.1 IVC on Normalized PSLG

We define a PSLG to be *normalized* if it satisfies the following properties:

- (*convex*) Every inner face is convex, and the complement of the outer face is also convex;
- (*low-degree*) Every inner vertex has degree 2 or 3, and every outer vertex has degree 3.
- (*bounded-angle*) Every interior angle attached to an inner vertex of degree 3 is in the range $[\pi/4, \pi)$.
- (α*-isometric*) There exist a standard length l and a constant $\alpha \in (0, 1)$ such that every edge with at least one endpoint being inner vertex has a length in range $[l(1-\alpha), l(1+\alpha)]$, and such an edge must also have at least one endpoint with degree 2.

Given a normalized PSLG, the *inner vertex cover (IVC)* problem on PSLG asks for a vertex cover which contains all the outer vertices of G. Given an instance of convex PSLG $G_0 = (V_0, E_0)$, we first construct a normalized PSLG $G_4 = (V_4, E_4)$ through other three intermediate graphs, such that VC on G_0 is reduced to IVC on G_4. The construction can be found in the full version, and we also decide the value of the parameter $\alpha = \Omega(1/|V_0|)$ there.

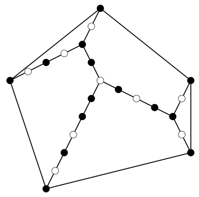


Figure 8 A normalized PSLG with an inner vertex cover labelled by black vertices.

▶ **Lemma 18.** *There is a polynomial reduction from VC on a convex PSLG to IVC on a normalized PSLG.*

4.2 Reduction to Dec-RMS

Then we construct a 3D point set D for Dec-RMS from the above IVC instance on the normalized PSLG $G_4 = (V_4, E_4)$. Consider the eighth sphere $\mathcal{S} = \{(x, y, z) \in \mathbb{R}^3_+ \mid x^2 + y^2 + z^2 = \rho^2\}$, where ρ is a constant depending on the parameters of G_4. and we draw the PSLG G_4 in the unit disk $\mathcal{D}$ inscribed in $\mathcal{S}$. From the origin O, we project the vertices and the edges $V_4 \cup E_4$ onto $\mathcal{S}$, and denote the projection mapping by η. Note that straight lines in E_4 are projected to arcs of great-circles, and thus the faces in the projection image are still convex. In fact, when ρ is large enough, the image of the graph does not deform a lot. Formally, we have Lemma 19, whose proof is omitted and can be found in the full version.

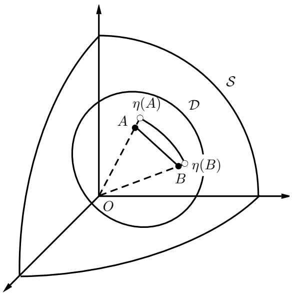


Figure 9 Sphere projection for G_4 from the disk $\mathcal{D}$

▶ **Lemma 19.** *For any two points $A, B \in \mathcal{D}$, we have*

$$\rho^2 \sqrt{1 - \frac{AB^2}{\rho^2 - 1}} \le \langle \eta(A), \eta(B) \rangle \le \rho^2 \left(1 - \frac{AB^2}{2\rho^2}\right).$$

Let D be the projection image of V_4. Thus, $|D| = |V_4|$, and D can be computed in polynomial time. The convexity of the sphere $\mathcal{S}$ ensures that D forms the 1-level of itself. Intuitively, the normalizing constraints on the degrees, angles, and edges of G_4 make sure that a point of D outside a subset $R \subseteq D$ could keep a small regret ratio if and only if all its neighbors are in R. We claim that by properly choosing θ in the Dec-RMS problem, a subset $R \subseteq D$ has 1-regret ratio at most $1 - \theta$ if and only if $\eta^{-1}(R)$ is an inner vertex cover of G_4, and this is implied by the following two lemmas.

▶ **Lemma 20.** *There exists a constant θ such that for any subset $R \subseteq D$, if $\eta^{-1}(R)$ is an inner vertex cover of G_4, then 1-regratio$(R) \leq 1 - \theta$.*

By careful computations in the proof of Lemma 20 in the full version, we can set $\theta = \sqrt{1 - l^2(1+\alpha)^2/(\rho^2 - 1)}$. For carefully chosen values of l, α and ρ, we can prove:

▶ **Lemma 21.** *For any subset $R \subseteq D$, if $\eta^{-1}(R)$ is not an inner vertex cover of G_4, then 1-regratio$(R) > 1 - \theta$.*

Combining Lemmas 20 and 21 leads to Lemma 22.

▶ **Lemma 22.** *For any integer r, G_4 has an inner vertex cover of size r if and only if D has an 1-regret set of size r with ratio $1 - \theta$, where D, θ can be obtained from G_4 in polynomial time.*

Theorem 16 thus follows.

5 Algorithms in High Dimensions

5.1 The Problem RMS

The concept of *ϵ-kernel* was introduced by Agarwal et al. [2]. By showing that RMS is closely connected to ϵ-kernel, we obtain an approximation algorithm for RMS and an upper bound of the maximum regret ratio, which improves the previous result [26].

A subset R of D is an ϵ-kernel if $\frac{\max_{x\in R}\langle x,\omega\rangle - \min_{y\in R}\langle y,\omega\rangle}{\max_{x\in D}\langle x,\omega\rangle - \min_{y\in D}\langle y,\omega\rangle} \geq 1 - \epsilon$, for any non-zero real vector ω. Roughly speaking, an ϵ-kernel is a subset that approximately preserves the *width* of the data set in every direction. It is well known that an ϵ-kernel of constant size can be computed in linear time (when $d = O(1)$).

▶ **Theorem 23.** [2, 6, 35] *Given D in $\mathbb{R}^d$, one can compute an ϵ-kernel of D of size $O(\epsilon^{-(d-1)/2})$ in $O(|D| + 1/\epsilon^d)$ time.*

We reduce the RMS problem to the ϵ-kernel problem as follows. Recall that due to our assumption, all points of D are in the first orthant. We make 2^d copies of D in every orthant as follows. Define $D^{\pm} = \{(p[1]x[1], \ldots, p[d]x[d]) \mid (x[1], \ldots, x[d]) \in D, p[i] \in \{\pm 1\})\}$. Suppose we have already found an ϵ-kernel R' of $D^{\pm}$; then we can project the subset back to D by taking the absolute value in each coordinate, as follows. Define

$$\mathsf{abs}(x) := (|x[1]|, \ldots, |x[d]|), \qquad R = \mathsf{abs}(R') := \{\mathsf{abs}(x) \mid x \in R'\}.$$

▶ **Lemma 24.** *If R' is an ϵ-kernel of $D^{\pm}$, then R must have regret ratio at most ϵ in D.*

Using the above reduction and observing that $|R| \leq |R'|$, it is immediate to translate Theorem 23 to an approximation algorithm for RMS in high dimensions.

▶ **Corollary 25.** *Fixing the dimension d, one can compute a subset $R \subseteq D$ of size r with maximum regret ratio $O(r^{-2/(d-1)})$ in $O(2^d n + r^{2d/(d-1)})$ time.*

The upper bound on the regret ratio is better than the previous upper bound $O(r^{-1/(d-1)})$ given in [26].

5.2 The Problem k-RMS

Given r and k, the goal of k-RMS is to compute a set R of r points from D such that the maximum regret ratio is minimized. Let $1-\theta^*$ denote the maximum regret ratio in the optimal solution, for some $\theta^* \in [0,1]$. Viewing each point in D as a $(d-1)$-dimensional hyperplane, Theorem 26 follows from a reduction to the set cover problem over the arrangement of these hyperplanes. See the full version for a complete proof.

▶ **Theorem 26.** *There exists a polynomial time algorithm that can compute a set R of at most $r \cdot (d \cdot \ln(2n) + 1)$ points from D with maximum regret ratio at most $1-\theta^*$.*

6 Related Work

Due to the individual drawback of top-k queries and skyline queries, there exist a variety of ways to combine these two queries in the literature. Top-k skyline select and top-k skyline join were proposed in [15]. ϵ-skyline [33] controls the output size with respect to ϵ after the utility function specified by the user is known. However, these studies still require that the utility function should be known beforehand.

The RMS query we study in this paper has the attractive property that no information on the utility function has to be provided by the user. Since its introduction in [26], it has been extended and generalized to the interactive setting in [25] and the k-RMS problem in [8]. [27] proposed an efficient algorithm for 1-RMS. [19] generalized the notion of RMS to include nonlinear utility functions.

Computing k-level sets and obtaining tight size bounds are of fundamental importance in computational geometry. For the two-dimension case, [12] provided the best-known upper bound $O(nk^{1/3})$. However, the best-known lower bound is $\Omega(ne^{c\sqrt{\log k}})$ [32], which is still far from the upper bound, and closing the gap is an open problem for years. For algorithms that compute the k-level sets, we refer the interested readers to [5] and the references therein. Note that any improvement on computing the k-level sets may lead to improvement of the time bounds of our algorithms for k-RMS.

The notion of ϵ-kernel coreset was introduced in the seminal paper by Agarwal et al. [2]. They applied their algorithm for constructing ϵ-kernel to several shape fitting problems. Since then, the idea has been extended to many other settings such as clustering (e.g., [7, 13]), matrix approximation [11, 13] and stochastic points [17].

Acknowledgement

Wei Cao, Jian Li, Kangning Wang, Ruosong Wang and Wei Zhan are supported in part by the National Basic Research Program of China Grant 2015CB358700, 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61202009, 61033001, 61361136003. The research of Raymond Chi-Wing Wong is supported by the grant GRF 16219816. Haitao Wang's research was supported in part by NSF under Grant CCF-1317143. We are grateful to the anonymous reviewers for their constructive comments on this paper.

References

1 A greedy algorithm — the interval point cover problem. `http://www.cs.yorku.ca/~andy/courses/3101/lecture-notes/IntervalCover.html`.

2 Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Approximating extent measures of points. *Journal of the ACM (JACM)*, 51(4):606–635, 2004.

3 S Borzsony, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

4 C.Y. Chan, HV Jagadish, K.L. Tan, A.K.H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, 2006.

5 Timothy M Chan. Remarks on k-level algorithms in the plane. *Manuscript, Department of Computer Science, University of Waterloo, Waterloo, Canada*, 1999.

6 Timothy M Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*, pages 152–159, 2004.

7 K. Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

8 Sean Chester, Alex Thomo, S Venkatesh, and Sue Whitesides. Computing k-regret minimizing sets. *Proceedings of VLDB*, 7(5), 2014.

9 Jan Chomicki, Paolo Ciaccia, and Niccolo' Meneghetti. Skyline queries, front and back. *SIGMOD Rec.*, 42(3):6–18, October 2013.

10 Gautam Das and Michael T Goodrich. On the complexity of approximating and illuminating three-dimensional convex polyhedra. In *Algorithms and Data Structures*, pages 74–85. Springer, 1995.

11 A. Deshpande, L. Rademacher, S. Vempala, and G. Wang. Matrix approximation and projective clustering via volume sampling. In *Proceedings of the 17th ACM-SIAM symposium on Discrete algorithm*, pages 1117–1126, 2006.

12 Tamal K Dey. Improved bounds on planar k-sets and k-levels. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 156–161. IEEE, 1997.

13 D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 569–578, 2011.

14 P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k bounded diversification. In *SIGMOD*, 2012.

15 M. Goncalves and M.E. Vidal. Top-k skyline: A unified approach. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, pages 790–799. Springer, 2005.

16 John Hershberger. Finding the upper envelope of n line segments in o (n log n) time. *Information Processing Letters*, 33(4):169–174, 1989.

17 Lingxiao. Huang, Jian. Li, Jeff. Phillips, and Haitao. Wang. ϵ-kernel coresets for stochastic points. In *ESA*, 2016.

18 Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, October 2008.

19 Taylor Kessler Faulkner, Will Brackenbury, and Ashwin Lall. K-regret queries with nonlinear utilities. *Proc. VLDB Endow.*, 8(13):2098–2109, September 2015.

20 J. Lee, G. You, and S. Hwang. Personalized top-k skyline queries in high-dimensional space. *Information Systems*, 2009.

21 C.E. Leiserson, C. Stein, R. Rivest, and T.H. Cormen. *Introduction to algorithms*. MIT press, 2009.

22 X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, 2007.

23 J. Matoušek. Randomized optimal algorithm for slope selection. *Information Processing Letters*, 39(4):183 − 187, 1991.

24 D. Mindolin and J. Chomicki. Discovering relative importance of skyline attributes. *VLDB*, 2009.

25 D. Nanongkai, A. Lall, A. D. Sarma, and K. Makino. Interactive regret minimization. In *SIGMOD*, 2012.

26 Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J Lipton, and Jun Xu. Regret-minimizing representative databases. *Proceedings of the VLDB Endowment*, 3(1-2):1114–1124, 2010.

27 P. Peng and R. C.-W. Wong. Geometry approach for k-regret query. In *ICDE*, 2014.

28 Franco P Preparata, Michael Ian Shamos, and Franco P Preparata. *Computational geometry: an introduction*, volume 5. Springer-Verlag New York, 1985.

29 L. Qin, J.X. Yu, and L. Chang. Diversifying top-k results. *VLDB*, 2012.

30 M.A. Soliman, I.F. Ilyas, and K. Chen-Chuan Chang. Top-k query processing in uncertain databases. In *IEEE 23rd International Conference on Data Engineering.*, pages 896–905. IEEE, 2007.

31 Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, 2009.

32 Géza Tóth. Point sets with many k-sets. *Discrete & Computational Geometry*, 26(2):187–194, 2001.

33 T. Xia, D. Zhang, and Y. Tao. On skylining with flexible dominance relation. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1397–1399. IEEE, 2008.

34 M.L. Yiu and N. Mamoulis. Multi-dimensional top-k dominating queries. *The VLDB Journal*, 18(3):695–718, 2009.

35 Hai Yu, Pankaj K Agarwal, Raghunath Poreddy, and Kasturi R Varadarajan. Practical methods for shape fitting and kinetic data structures using coresets. *Algorithmica*, 52(3):378–402, 2008.