k-Regret Minimizing Set: Efficient Algorithms and Hardness

Wei Cao[1], Jian Li[1], Haitao Wang[3], Kangning Wang[1], Ruosong Wang[1], Raymond Chi-Wing Wong[6], and Wei Zhan[1]

1 **Institute for Interdisciplinary Information Sciences, Tsinghua University, China**
{cao-w13,wkn13,wrs13,zhan-w13}@mails.tsinghua.edu.cn,lijian83@mail.tsinghua.edu.cn
3 **Utah State University, Logon, Utah, USA, haitao.wang@usu.edu**
6 **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. Since 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. Our experimental results show that our algorithms are more efficient than previous ones on both synthetic and real datasets up to three orders of magnitude. 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.

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 [15, 17, 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. 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 [4, 5, 17, 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'.

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 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*, a new type of queries meeting the above two goals, were proposed [9, 25, 26, 27] and studied extensively due to its usefulness and its wide applicability, where r is a positive integer. All applications originally applied to top-k queries and skyline queries could also be applied to r-regret 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. [9] proposed a new version of r-regret queries called the *k-regret minimizing set (k-RMS)* problem generalizing 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. [9] relaxed this assumption and considered that a user is already satisfied 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 $O(rn^2 + n^2 \log n)$ [9].
2. For k-RMS in $\mathbb{R}^2$, we present an $O(n^2 \log n)$ time algorithm, which improves the $O(rn^2k^{\frac{1}{3}} + n^2 \log n)$ time complexity result in [9]. 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 [9]. 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. Our extensive experimental results show that our algorithms are consistently faster than

the previous work [9] up to three orders of magnitude.

4. 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. [9] 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 $\mathbb{R}^2$).
5. 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 limit, many details and proofs are omitted but can be found in the appendix. In particular, Appendix D reports the experimental results for our 2-dimensional case algorithms.

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 [9, 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 $f(x)$. Given an integer $k \geq 1$, we denote the k-th largest score of x with respect to weight vector ω by $\mathrm{kmax}_{x\in D}\langle x, \omega\rangle$.

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 $k\text{-regratio}(R, \omega)$, is defined to be

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

If $k\text{-regratio}(R, \omega)$ 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 $k\text{-regratio}(R)$, is defined to be $k\text{-regratio}(R) = \sup_{\omega\in W} k\text{-regratio}(R, \omega)$.

▶ Problem 1 (k-RMS [9]). Given two positive integers k and r, we want to find a set R of r points from D such that $k\text{-regratio}(R)$ is minimized.

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

Problem	Algorithm	Time Complexity	Deterministic	Exact	Related Material
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	[9]
	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	$O(rn^2 k^{\frac{1}{3}} + n^2 \log n)$ ††	yes	yes	[9]
	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. 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 $O(n \log m + m \log^{1+\delta} n)$ preprocessing time, and runs in $O(n+m)$ time for any threshold θ. †† In [9], the authors claim their algorithm runs in $O(rn^2)$ time. However, a more careful examination shows their algorithm runs in $O(rn^2 + n^2 \log n)$ time for RMS and $O(rn^2 k^{\frac{1}{3}} + n^2 \log n)$ 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 $O(nk^{\frac{1}{3}})$ (rather than n, as [9] assumed).

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. The experimental results for these algorithms are in Appendix D. Without loss of generality, we assume that $\|\omega\|_1 = \omega[1] + \omega[2] = 1$ for any weight vector $\omega \in W$ (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:
$$\forall \lambda \in [0, 1], \ \max_{p \in R} f_p(\lambda) \geq \theta \cdot \mathrm{kmax}_{p \in D} f_p(\lambda), \tag{1}$$
where kmax 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 $\mathbb{A}(D)$.

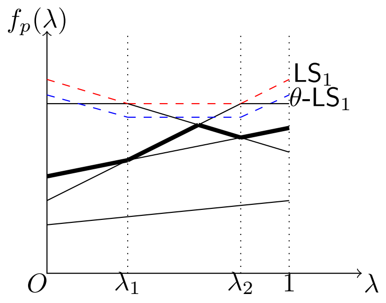


Figure 1 The arrangement $\mathbb{A}(D)$ (black lines), its 1-level set LS$_1$ (red dashed line), θ-scaled 1-level set (blue lines), and 3-level set (thick black line)

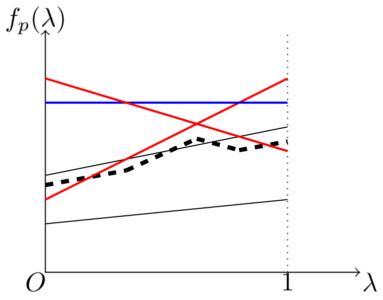


Figure 2 Illustrating Example 2: θ-LS$_k$ is the dashed thick curve

The *k-level set* of $\mathbb{A}(D)$ [6] is a piecewise linear curve (see Figure 1 for an example)

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

Let m denote the number of segments of LS$_k(\lambda)$. It is known that m is bounded by $O(nk^{\frac{1}{3}})$ and LS$_k(\lambda)$ can be computed in $O(n\log n + nk^{\frac{1}{3}})$ expected time by a randomized algorithm [6] or in $O(n\log m + m\log^{1+\delta} k)$ time by a deterministic algorithm for any constant $\delta > 0$ [6]. 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-LS$_k(\lambda) = \theta \cdot \mathsf{LS}_k(\lambda) = \theta \cdot \text{kmax}_{p\in D} f_p(\lambda)$, 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 [18]. 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_k 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-k-RMS to check whether the regret ratio of $1 - \theta$ is achievable (i.e., whether θ-LS_k 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 actually solve the following more general problem. Given any $\theta \in [0, 1]$, the problem is to compute a smallest subset $R \subseteq D$ of lines such that the upper envelop of R is above θ-LS_k. We call our algorithm D-Greedy-k. We will prove the following theorem.

▶ **Theorem 5.** *After $O(n \log m + m \log^{1+\delta} k)$-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.*

Below we describe our algorithm. Our algorithm is quite elegant, simple, and easy to implement (e.g., the most "complicated" data structure used in the algorithm is linked lists), although the correctness proof is somewhat involved. As will be seen later, the algorithm is interesting both theoretically and practically.

3.2.1 Preliminaries

Let l_0 denote the vertical line $\lambda = 0$. As preprocessing, we sort in $O(n \log n)$ time all lines of D by their intersections with l_0 from top to bottom. Then we compute LS_k in $O(n \log m + m \log^{1+\delta} k)$ time for any $\delta > 0$ [6]. The total preprocessing time is $O(n \log n + m \log^{1+\delta} k)$ (since $m = O(nk^{\frac{1}{3}})$ and $k \leq n$).

Given any $\theta > 0$, the goal is to find a smallest subset $R \subseteq D$ of lines such that the upper envelop of R is above θ-LS_k for $\lambda \in [0, 1]$. In the following, we give an $O(n + m)$ time algorithm for the problem (excluding the time of the preprocessing).

We first compute θ-LS_k, which can be done in $O(m)$ time since LS_k has been computed in the preprocessing. To simplify the notation, we use C to refer to θ-LS_k. Let $l_1, l_2, \ldots, l_n$ be the lines of D sorted by their intersections with l_0 from top to bottom. For each $i \in [1, n]$, let a_i denote the intersection of l_i and l_0. Let l_{n+1} denote the vertical line $\lambda = 1$.

▶ **Lemma 6.** *For any $1 \leq i < j \leq n$, 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).*

Proof. See Appendix A for the proof. ◀

We run a *pruning procedure* on D to remove such lines l_j as specified in the preceding lemma. This can be done in $O(n)$ time by scanning the lines of D in their index order. We omit the details. The following algorithm will work on the remaining lines of D. But to simplify the notation, we assume no lines have been pruned from D, and thus, the lines of D following their index order are also sorted by their slopes in strictly ascending order.

For any two values λ_1 and λ_2 with $\lambda_1 \leq \lambda_2$, 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 it 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 $\lambda_1 \leq \lambda_2$ if for any value $\lambda' \in [\lambda_1, \lambda_2]$, the intersection of $l_{\lambda'}$ and C' is not lower than that of $l_{\lambda'}$ and C, where $l_{\lambda'}$ is the vertical line $\lambda = \lambda'$. By this definition, C is above C itself.

For any $0 \leq i \leq n + 1$, let $D_i = \{l_0, l_1, \ldots, l_i\}$. Our algorithm processes the lines of $D_{n+1} = \{l_0, l_1, \ldots, l_{n+1}\}$ 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 $0 \leq i \leq n$. Our algorithm maintains a set $R_i = \{l_{f(0)}, l_{f(1)}, \ldots, l_{f(g_i)}\}$ of $g_i + 1$ lines of D_i with $f(0) < f(1) < \cdots < f(g_i)$, for some integer $g_i \geq 0$, and a particular point p at $b_{f(g_i)}$, such that R_i and p have the following properties. Refer to Figure 3 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, after which it is ambiguous to talk about the upper envelop of $U(R_i)$. Similarly, we tilt l_{n+1} 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. The portion of $U(R_i)$ in the interval $[0, \lambda(p)]$ is above $C[0, \lambda(p)]$.
4. For each line $l_{f(t)}$ of R_i with $0 \leq t \leq i$, it has a specific point $b_{f(t)}$ defined as follows. If $t = 0$, then $b_{f(t)}$ (i.e., b_0) is the intersection of C and l_0. 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 $P(C)$. If $t > 0$, suppose point $b_{f(t-1)}$ has been defined on C. It holds that q is above $b_{f(t-1)}$, where q is the intersection of $l_{f(t)}$ and the vertical line through $b_{f(t-1)}$. 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 $P(C)$ (note that $b_{f(t)}$ is necessarily on C).

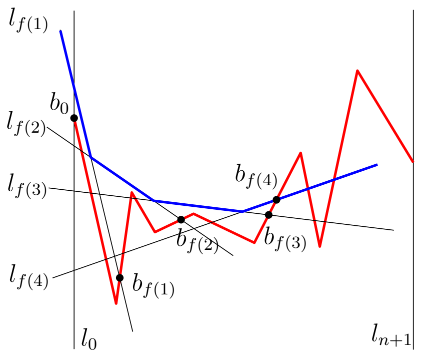


Figure 3 Illustrating the set $R_i = \{l_0, l_{f(1)}, l_{f(2)}, l_{f(3)}, l_{f(4)}\}$ with $g_i = 4$. The red curve is C and the blue curve is $U(R_i)$. The point p is at $b_{f(4)}$.

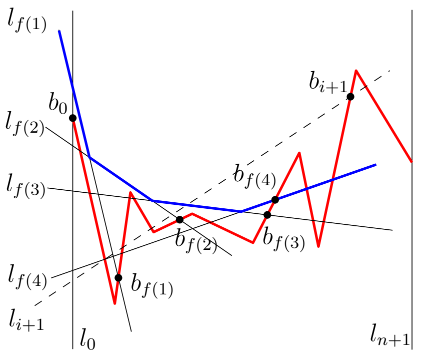


Figure 4 Illustrating the case where l_{i+1} (the dashed line) intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$ with $g_i = 4$. In this example, $R_{i+1} = \{l_0, l_{f(1)}, l_{f(2)}, l_{i+1}\}$ because l_{i+1} intersects $\overline{a_{f(1)}b_{f(1)}}$ but is not above $C[b_{f(1)}, b_{f(2)}]$.

For each line $l_{f(t)}$ of R_i with $0 \le t \le i$, $b_{f(t)}$ has been computed, and further, the (at most two) edges of C that contain $b_{f(t)}$ are also maintained.

5. $\overline{a_{f(t)}b_{f(t)}}$ intersects $\overline{a_{f(t-1)}b_{f(t-1)}}$ for any t with $1 \le t \le i$.
6. For any $2 \le t \le i$, 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 on the interval $[\lambda(b_{f(t-2)}), \lambda(b_{f(t-1)})]$ (i.e., there is some value $\lambda' \in [\lambda(b_{f(t-2)}), \lambda(b_{f(t-1)})]$ such that $l_{f(t)}$ is strictly below C at λ').
7. For each $1 \le t \le i$, 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.
8. $\lambda(b_{f(0)}) < \lambda(b_{f(1)}) < \ldots < \lambda(b_{f(g_i)})$.

3.2.2 The Algorithm

Initially, for $i = 0$, we let $R_0 = \{l_0\}$ and $p = b_0$. In general, suppose we have processed the line l_i and obtained R_i. In the following, we give the algorithm for processing the next line l_{i+1} and obtain the set R_{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. 7 in Appendix A). 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 and the edges of C that contain $b_{f(g_i)}$ are also 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 4), then we proceed to compute the point b_{i+1} as follows.

As $f(g_i) \le i < i+1$, 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}. We define b_{i+1} as the first point of $P(C)$ we encounter after which q will move out of $P(c)$ (e.g., see Figure 4). 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 p rightwards on C until we meet an intersection between l_{i+1} and an edge of C. Hence, the running time for finding b_{i+1} is linear in the number of edges of C between $b_{f(g_i)}$ and b_{i+1}. After b_{i+1} is computed, we set p to b_{i+1}.

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 p 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 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 $L(b_{f(g_i-1)}, b_{f(g_i)})$. Later in Lemma 8 we will give an *upper hull walking procedure* to 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 $C[b_{f(g_i-1)}, b_{i+1}]$. The merge procedure will be given later in Lemma 7.

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 4 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 Lemma 29 of Appendix A, we show that $R_{n+1} \setminus \{l_0, l_{n+1}\}$ is the optimal solution set R for the problem Dec-k-RMS.

The subsequent two lemmas give the upper hull merge and walking procedures.

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

Proof. See Appendix A for the proof. ◀

▶ **Lemma 8.** *We can implement the upper hull walking procedure such that the total time of the procedure in the entire algorithm is $O(m + n)$.*

Proof. See Appendix A for the proof. Note that the algorithm efficiency relies on that the slopes of the lines of D in their index order are sorted increasingly. ◀

Lemma 28 in Appendix A shows that the algorithm runs in $O(m + n)$ time. This proves Theorem 5. The pseudocode is also in Appendix A.

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 9.** *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 10.** *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 $\lambda \in X(D) := \{0,1\} \cup \{\lambda \mid \exists\, l_1, l_2 \in D, l_1(\lambda) = l_2(\lambda)\}$.*

Proof. See Appendix C for the proof. ◀

The following lemma is a consequence of Lemma 10, and the proof is in Appendix C.

▶ **Lemma 11.** *For k-RMS, the optimal θ is $0, 1$ or in*

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

Clearly, the set $\mathsf{Cand}(D)$ consists of $O(n^3)$ 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.

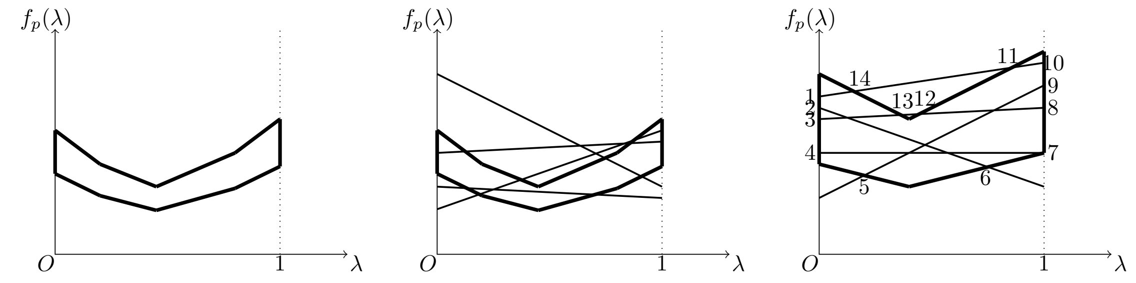
First we compute and sort the set $X(D)$. For each $\lambda \in X(D)$, our algorithm maintains an interval of indices $I_\lambda \subseteq [1, n]$ so that if the optimal value θ is equal to the j-th largest value of $l(\lambda)/\mathsf{LS}_k(\lambda)$ for all $l \in D$, then $j \in I_\lambda$ must hold. 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 11) 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 each I_{λ_i} with $\theta_i \ge \theta_m$ to be its lower-half interval; otherwise, we update each I_{λ_i} with $\theta_i \le \theta$ to be its upper-half interval. Hence, the reduced weight of all intervals of S is $\frac{1}{2}\sum\{w_i \mid \theta_i \ge \theta\}$ or $\frac{1}{2}\sum\{w_i \mid \theta_i \le \theta\}$, 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).



(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 5 Illustration of the band, the segments in it, and the sampling method

▶ **Theorem 12.** *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 11 still applies here, but we have a stronger Lemma 13 (which is not applicable to k-RMS), whose proof is in Appendix C.

▶ **Lemma 13.** *For RMS, the optimal value θ is in the set $\mathsf{Cand}(D)$ defined as*

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

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 $|\mathsf{Cand}_1(D)| \le 2n + 2 = O(n)$. 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 5a). 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. 5b). 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. 5c), 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 14.** *After the traversal, the counter N is the number of candidates of $\theta = \frac{l(\lambda)}{\mathsf{LS}_1(\lambda)}$ in $[\theta_0, \theta_1]$.*

▶ **Example 15.** As an example, consider the instance presented in Figure 5c. 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 $\{1, \dots, N\}$. 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., $N \le n$). 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 $p(n_0, n_1) \ge 1 - n_0 \cdot \left(\frac{n_0 - n_1}{n_0}\right)^n$, 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 $p^* = p(n^2, n^{3/2}) \cdot p(n^{3/2}, n) \ge (1 - n^2 e^{-\sqrt{n}})^2$ is close to 1. Suppose the algorithm terminates at round t (t is a random variable). For any $r \ge 3$, it holds that $\Pr[t \ge r] \le (1 - p^*)^{r-2}$. Overall, the expected number of rounds is $\mathbb{E}[t] \le 2 + \sum_{r=3}^{\infty} \Pr[t \ge r] \le 2 + \sum_{r=3}^{\infty} (1 - p^*)^{r-2} = O(1)$.

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

▶ **Theorem 16.** *E-Greedy-1 computes an optimal solution for the problem RMS in $O(n \log n)$ expected time.*

4 NP-Hardness

The main results of this section are the NP-hardness results in Theorem 17 and Theorem 18.

▶ **Theorem 17.** *RMS is* NP*-hard even in* $\mathbb{R}^3$*.*

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

▶ **Theorem 18.** *k-RMS is* NP*-hard for any fixed $k \geq 1$ and $d \geq 3$.*

In the following, we will prove Theorem 17, 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 PLSG is NP-hard, and below we do the reduction to prove Theorem 17.

We first define an intermediate problem, called the *inner vertex cover problem* (IVC), on a class of so-called *normalized PLSG graphs*, and show that it is NP-hard in Section 4.1. Then, we provide the reduction to RMS from IVC in Section 4.2.

4.1 IVC on Normalized PLSG

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 $\alpha > 0$ 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. We first show it is NP-hard as well, by a reduction from vertex cover on convex PSLG. This is done by constructing a normalized PSLG G_4 from every convex PSLG G_0, and the complete proof can be found in Appendix B.

▶ **Lemma 19.** *IVC is* NP*-hard on a normalized PLSG.*

4.2 Reduction to RMS

We construct a 3D point set D for RMS from any given IVC instance on a normalized PSLG $G_4 = (V_4, E_4)$. Consider the eighth sphere $\mathcal{S} = \{(x, y, z) \in \mathbb{R}^3_+ \mid x^2 + y^2 + z^2 = \rho^2\}$, 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 20, whose proof is in Appendix B.

▶ **Lemma 20.** *For any two points $A, B \in \mathcal{D}$, we have*

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

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 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 regret ratio ϵ if and only if all its neighbors are in R. A k-regret set R is called a *(k, ϵ)-set* if the regret ratio is at most ϵ. We claim that by properly choosing ϵ, a subset $R \subseteq D$ is an $(1, \epsilon)$-set if and only if $\eta^{-1}(R)$ is an inner vertex cover of G_4, and this is implied by the following two lemmas.

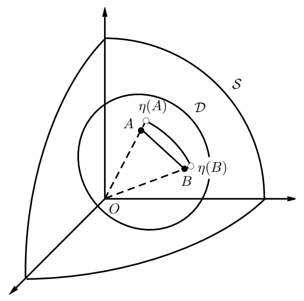


Figure 6 Sphere projection for G_4 from the disk $\mathcal{D}$

▶ **Lemma 21.** *There exists a constant ϵ such that for any subset $R \subseteq D$, if $\eta^{-1}(R)$ is an inner vertex cover of G_4, then R is an $(1, \epsilon)$-set of D.*

According to the proof of Lemma 21 in the appendix, we can set $\epsilon = 1 - \sqrt{1 - l^2(1 + \alpha)^2/(\rho^2 - 1)}$. Note that we can regard l as a small constant since $\mathcal{D}$ is a unit disk, and ρ can be arbitrarily large.

▶ **Lemma 22.** *There exists a constant α such that for any subset $R \subseteq D$, if $\eta^{-1}(R)$ is not an inner vertex cover of G_4, then R is not an $(1, \epsilon)$-set of D.*

Combining Lemmas 21 and 22 leads to Lemma 23.

▶ **Lemma 23.** *G_4 has an inner vertex cover of size k' if and only if D has an $(1, \epsilon)$-set of size r, where D, ϵ, r can be obtained from G_4 and k' in polynomial time.*

Theorem 17 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 24.** [2, 7, 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 25.** *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 $|R| \le |R'|$, it is immediate to translate Theorem 24 to an approximation algorithm for RMS in high dimensions.

▶ **Corollary 26.** *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 $\theta^* \in [0, 1]$. The proof of Theorem 27 is in Appendix C.

▶ **Theorem 27.** *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 [17]. 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 [9]. [27] proposed an efficient algorithm for 1-RMS.

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 [6] 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., [8, 14]), matrix approximation [11, 14] and stochastic points [19].

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 Pankaj K Agarwal and Micha Sharir. Arrangements and their applications. *Handbook of computational geometry*, pages 49–119, 2000.

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

5 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.

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

7 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.

8 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.

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

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 Herbert Edelsbrunner, Joseph O'Rourke, and Raimund Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15(2):341–363, 1986.

14 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.

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

16 Michael R Garey and David S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

17 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.

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

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

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. URL: `http://www.sciencedirect.com/science/article/pii/002001909190177J`, doi:`http://dx.doi.org/10.1016/0020-0190(91)90177-J`.

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.

APPENDIX

A Missing Proofs of the Algorithm D-Greedy-k for Dec-k-RMS

In this section, we present the missing details of our algorithm D-Greedy-k described in Section 3.2. In particular, we give the proofs of Lemmas 6, 7 and 8, and we prove the correctness and the time complexity of our algorithm. The pseudocode is also provided.

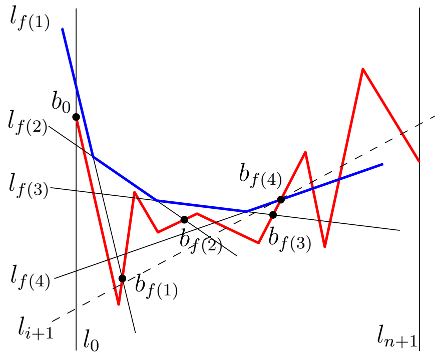


Figure 7 Illustrating the special case: the dashed line is l_{i+1}. In this case, b_{i+1} is $b_{f(4)}$.

A.1 Proof of Lemma 6

Assume the slope of l_i is larger than or equal to that of l_j. Then, since a_j is lower than or equal to a_i, the line segment of l_i between the two vertical lines l_0 and l_{n+1} must be higher than or equal to the line segment of l_j between l_0 and l_{n+1}. This implies that l_i "dominates" l_j. In other words, if there is an optimal solution set R that contains l_j, we can obtain another optimal solution set by replacing l_j by l_i in R. The lemma thus follows.

A.2 Proof of Lemma 7

Suppowe we want 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 $C[b_{f(g_i-1)}, b_{i+1}]$.

The merge procedure works by finding the upper tangent of the upper hulls of $C[b_{f(g_i-1)}, b_{f(g_i)}]$ and $C[b_{f(g_i)}, b_{i+1}]$. Since the two upper hulls are defined on the λ-values in two interior-disjoint intervals $[\lambda(b_{f(g_i-1))}, \lambda(b_{f(g_i)})]$ and $[\lambda(b_{f(g_i)}), \lambda(b_{i+1})]$, finding the upper tangent can be done easily by the standard approach by traversing on $L(b_{f(g_i-1)}, b_{f(g_i)})$ from right to left and traversing $L(b_{f(g_i)}, b_{i+1})$ from left to right. During finding the tangent, the edges of $L(b_{f(g_i-1)}, b_{f(g_i)})$ (resp., $L(b_{f(g_i)}, b_{i+1})$) to the right (resp., left) of the tangent point are removed. After the tangent is found, $L(b_{f(g_i-1)}, b_{i+1})$ is simply the concatenation of the remaining list $L(b_{f(g_i-1)}, b_{f(g_i)})$, the tangent, and the remaining list $L(b_{f(g_i)}, b_{i+1})$. Hence, the running time of the merge procedure is $O(k'+1)$, where k' is the total number of edges removed from the original lists $L(b_{f(g_i-1)}, b_{f(g_i)})$ and $L(b_{f(g_i)}, b_{i+1})$.

Once an edge is removed from either of the above two lists, it will never appear in any upper hull maintained in future algorithm. Therefore, the total sum of the values k' in all upper hull merge procedures in the entire algorithm is at most k^*, where k^* is the total number of different edges in all upper hulls that have ever been maintained by our algorithm.

We claim that $k^* = O(n+m)$. Indeed, consider any edge e in any upper hull maintained by our algorithm. e becomes an edge of an upper hull either when we move p rightwards on C (i.e., e is an edge of the upper hull of $C[b_{f(i)}, b_{i+1}]$ for some i) or due to that e is computed as the upper tangent of two upper hulls in an upper hull merge procedure. It is not difficult to see that the total number of all such edges e in the above first case is $O(m)$. For the second case, note that each merge procedure generates at most one upper tangent and the toal number of merge procedures in the entire algorithm is at most n. Thus, the total number of all such edges e in the above second case is $O(n)$. Hence, $k^* = O(m+n)$.

The lemma thus follows.

A.3 Proof of Lemma 8

As discussed in the algorithm description, suppose we want to determine whether 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 $L(b_{f(g_i-1)}, b_{f(g_i)})$.

Let U denote the upper hull of $C[b_{f(g_i-1)}, b_{f(g_i)}]$. Let $v(l_{i+1})$ denote a vertex of U such that the line through $v(l_{i+1})$ and parallel to l_{i+1} is above U. Observe that l_{i+1} is above U if and only if l_{i+1} is above $v(l_{i+1})$. Therefore, once we find $v(l_{i+1})$, we can determine whether l_{i+1} is above U in constant time. To find $v(l_{i+1})$, we can simply traverse U from right to left, i.e., scanning the list $L(b_{f(g_i-1)}, b_{f(g_i)})$ from right to left until we obtain $v(l_{i+1})$. Hence, the running time is linear in the number of edges of U to the right of $v(l_{i+1})$. In addition, we maintain the point $v(l_{i+1})$ for future use, as follows.

Suppose when our algorithm processes the line l_{i+2}, we also need to determine whether l_{i+2} is above U. Similarly, the key is to find the corresponding point $v(l_{i+2})$. Recall that the slopes of the lines of D in their index order are sorted increasingly. Hence, the slope of l_{i+2} is larger than that of l_{i+1}. This implies that the point $v(l_{i+2})$ on U must be either the same as $v_{l_{i+1}}$ or to the left of it. Thus, to find $v(l_{i+2})$, we only need to scan the list $L(b_{f(g_i-1)}, b_{f(g_i)})$ leftwards starting from the vertex $v(l_{i+1})$. After $v(l_{i+2})$ is found, similarly, we maintain $v(l_{i+2})$ for future use, but $v(l_{i+1})$ does not need to be maintained any more. In this way, each edge of the list $L(b_{f(g_i-1)}, b_{f(g_i)})$ will be traversed only once during the upper hull walking procedure in the entire algorithm. We have shown in the proof of Lemma 7 that the total number of edges in all upper hulls maintained by our algorithm is $O(m+n)$. Hence, the total time of the upper hull walking procedure in the entire algorithm is $O(m+n)$.

According to the above discussion, in general, suppose line l_i has just been processed by our algorithm, other than the eight properties of R_i maintained by our algorithm, we also maintain the vertex $v(l_i)$ on the upper hull of $C[b_{f(i-1)}, b_{f(i)}]$, which is used for the upper hull walking procedure as discussed above.

A.4 The Running Time and the Correctness of Our Algorithm

▶ **Lemma 28.** *After the $O(n \log n + m \log^{1+\delta} k)$-time preprocessing that computes k-level set LS_k and sorts the lines of D, given any $\theta > 0$, the running time of the algorithm is $O(n+m)$.*

Proof. Consider a general step of the algorithm for processing line l_{i+1}. If l_{i+1} does not intersect $\overline{a_{f(g_i)} b_{f(g_i)}}$, then l_{i+1} is simply ignored. Thus, the step takes $O(1)$ time in this case.

If l_{i+1} intersects $\overline{a_{f(g_i)} b_{f(g_i)}}$, as discussed before, we can determine whether the special case happens in constant time. If yes, then l_{n+1} is simply ignored, which takes $O(1)$ time.

Otherwise, we find b_{i+1} by moving p on C, whose running time is linear in the number of edges of C between $b_{f(g_i)}$ and b_{i+1}. Observe that the point p always moves rightwards on C. Thus, the total time of this procedure in the entire algorithm is linear in the size of C, which is $O(m)$. We also need to construct the upper hull list $L(b_{f(g_i)}, b_{i+1})$, which also takes linear time in the number of edges of C between $b_{f(g_i)}$ and b_{i+1}. Hence, the total time of the above procedure for building the upper hull lists in the entire algorithm is $O(m)$.

Next, the algorithm determines the set R_{i+1}, by consider the lines of R_i in the reverse order of their indices. There are two major procedures: the upper hull walking procedure and the upper hull merge procedure. We have already shown that both procedures take $O(m+n)$ time in total in the entire algorithm. Other than the above two major procedures, determining the set R_{i+1} takes $O(|R_i| - |R'_i| + 1)$ time. Note that $|R_i| - |R'_i|$ is the number of lines removed from R_i. An easy observation is that any line of D_{n+1} can be removed from R_i for some i at most once in the entire algorithm. Thus, the total time for determining the set R_{i+1} in the entire algorithm is $O(n + m)$.

The lemma thus follows. ◀

With the next lemma, which proves the correctness of the algorithm, Theorem 5 follows.

▶ **Lemma 29.** *The algorithm correctly solves the* Dec-k-RMS *problem, i.e., the set $R_{n+1} \setminus \{l_0, l_{n+1}\}$ is an optimal solution.*

Proof. First, we show that our algorithm correctly maintains the eight properties of the set R_i for all $i = 0, 1, \ldots, n+1$.

Initially when $i = 0$, we have $R_0 = \{l_0\}$ and $p = b_0$. All properties trivially hold for R_0. In general, suppose R_i maintains the eight properties for some i. We show below that after l_{i+1} is processed, R_{i+1} also has these properties.

According to our algorithm, if l_{i+1} is ignored, then $R_{i+1} = R_i$ and p does not change. Since nothing is changed, R_{i+1} still has the eight properties. Otherwise, $R_{i+1} = R'_i \cup \{l_{i+1}\}$ and $p = b_{i+1}$, where R'_i consists of the first $i' + 1$ lines in R_i for some $0 \leq i' \leq g_i$, i.e., $R'_i = \{l_{f(0)}, l_{f(1)}, \ldots, l_{f(i')}\}$.

According to our algorithm, l_{i+1} intersects $\overline{a_{f(i')}b_{f(i')}}$ and is above $C[b_{f(i')}, b_{i+1}]$. Further, if $i' \geq 1$, either l_{i+1} does not intersect $\overline{a_{f(i'-1)}b_{f(i'-1)}}$, or they intersect but l_{i+1} is not above $C[b_{f(i'-1)}, b_{f(i')}]$. Hence, properties (5) and (6) hold.

As b_{i+1} is computed, property (4) holds. According to our algorithm, the upper hull of $C[b_{f(i')}, b_{i+1}]$ has been computed and maintained in the list $L[b_{f(i')}, b_{i+1}]$. Hence, property (7) holds. We have discussed before that $\lambda(b_{i+1}) > \lambda(b_{f(g_i)})$. Hence, property (8) holds.

In addition, properties (1), (2), and (3) trivially hold. Therefore, all eight properties hold for R_{i+1}.

In the following, we show that the algorithm correctly computes an optimal solution set (i.e., $R_{n+1} \setminus \{l_0, l_{n+1}\}$) for the problem Dec-$k$-RMS. To this end, we will prove that for each $0 \leq i \leq n+1$, the set $R_i = \{l_{f(0)}, l_{f(1)}, \ldots, l_{f(g_i)}\}$ has the following additional *key properties*.

1. $\lambda(b_{f(g_i)})$ is the largest value λ in $[0, 1]$ such that $C[0, \lambda]$ is covered by the lines of D_i.
2. For any t with $1 \leq t \leq g_i$ and $f(t) \neq n+1$, $\lambda(b_{f(t)})$ is the largest value λ in $[0, 1]$ such that $C[0, \lambda]$ is covered by at most t lines of D_i.
3. For any t with $1 \leq t \leq g_i$ and $f(t) \neq n+1$, $\{l_{f(1)}, l_{f(2)}, \ldots, l_{f(t)}\}$ is the smallest subset of D_i that can cover $C[0, \lambda(b_{f(t)})]$.

Before proving the key properties, we show that if the key properties hold, $R_{n+1} \setminus \{l_0, l_{n+1}\}$ is an optimal solution set of Dec-k-RMS, i.e., it is a smallest subset of D that can cover $C[0, 1]$.

Indeed, since the upper envelop of all lines of D is above C, the lines of D can cover $C[0, 1]$. After l_n is processed, by the key property (1), the lines of R_n must cover $C[0, 1]$, i.e., $\lambda(b_{g_n}) = 1$. Thus, l_{n+1} intersects $\overline{a_{f(g_n)}b_{f(g_n)}}$. According to our algorithm, after l_{n+1}

is processed, l_{n+1} must be in R_{n+1} (in fact $f(g_{n+1}) = n+1$). This shows that R_{n+1} covers $C[0,1]$ and l_{n+1} is in R_{n+1}.

Consider the set R_{n+1} and $t = g_{n+1} - 1$. According to our algorithm, $\overline{a_{f(g_{n+1})}b_{f(g_{n+1})}}$ must intersect $\overline{a_{f(t)}b_{f(t)}}$. Note that $f(g_{n+1}) = n+1$. By our way of defining a_{n+1}, b_{n+1}, and $b_{f(t)}$, the point $b_{f(t)}$ must be on l_{n+1}, i.e., $\lambda(b_{f(t)}) = 1$. By the key property (3), $\{l_{f(1)}, l_{f(2)}, \ldots, l_{f(t)}\}$, which is $R_{n+1} \setminus \{l_0, l_{n+1}\}$, is the smallest subset of D_{n+1} that can cover $C[0, \lambda(b_{f(t)})] = C[0,1]$. This shows that $R_{n+1} \setminus \{l_0, l_{n+1}\}$ is an optimal solution set for the problem Dec-k-RMS.

It remains to prove that the key properties hold on R_i for each $i = 0, 1, \ldots, n+1$. We prove it by induction, as follows.

When $i = 0$, the key properties trivially hold on $R_0 = \{l_{f(0)}\}$ since $\lambda(b_{f(0)}) = 0$.

We assume that the key properties hold for R_i for some i with $0 \le i \le n$. Next, we show that the key properties also hold for $i+1$. Our goal is to prove the following: (1) $\lambda(b_{f(g_{i+1})})$ is the largest value λ in $[0,1]$ such that $C[0,\lambda]$ is covered by the lines of D_{i+1}; (2) for any t with $1 \le t \le g_{i+1}$ and $f(t) \ne n+1$, $\lambda(b_{f(t)})$ is the largest value λ in $[0,1]$ such that $C[0,\lambda]$ is covered by at most t lines of D_{i+1}; (3) for any t with $1 \le t \le g_{i+1}$ and $f(t) \ne n+1$, $\{l_{f(1)}, l_{f(2)}, \ldots, l_{f(t)}\}$ is the smallest subset of lines of D_{i+1} that can cover $C[0, \lambda(b_{f(t)})]$.

According to our algorithm, there are three cases depending on whether and how l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$. We analyze the three cases below.

The first case If l_{i+1} does not intersect $\overline{a_{f(g_i)}b_{f(g_i)}}$, according to our algorithm, $R_{i+1} = R_i$ and $g_{i+1} = g_i$.

For the key property (1), assume to the contrary that it is not true. Then, there is a value $\lambda' > \lambda(b_{f(g_{i+1})}) = \lambda(b_{f(g_i)})$ such that $C[0,\lambda']$ is covered by D_{i+1}. To simplify notation, let $j = f(g_i)$.

Let p be the point of C to the right of $b_{f(g_i)}$ and infinitely close to $b_{f(g_i)}$. Hence, $\lambda(b_j) < \lambda(p) \le \lambda'$. Since a_{i+1} is below a_j and l_{i+1} does not intersect $\overline{a_j b_j}$, l_{i+1} must be strictly below p. Because $C[0,\lambda']$ is covered by D_{i+1}, D_{i+1} must have a line l' that is above p. Since l_{i+1} is not above p, l' is not l_{i+1} and thus is in D_i. Clearly, l' is above l_{i+1} at $\lambda(p)$.

Since $\lambda(p) < \lambda'$, the lines of D_{i+1} cover $C[0,\lambda(p)]$. We claim that the lines of D_i cover $C[0,\lambda(p)]$. Indeed, since l' is in D_i, the intersection of l' and l_0 is above a_{i+1}. Further, since l' is above l_{i+1} at $\lambda(p)$, we obtain that l' is above l_{i+1} on the interval $[0,\lambda(p)]$. This means that for any point $q \in C[0,\lambda(p)]$ covered by l_{i+1}, q is also covered by l'. Therefore, since the lines of D_{i+1} cover $C[0,\lambda(p)]$, the lines of $D_i = D_{i+1} \setminus \{l_{i+1}\}$ also cover $C[0,\lambda(p)]$.

However, since the key property (1) holds for i, $\lambda(b_j)$ is the largest value λ in $[0,1]$ such that $C[0,\lambda]$ is covered by the lines of D_i. Because $\lambda(b_j) < \lambda(p)$, we obtain contradiction.

This proves that the key property (1) holds for $i+1$.

For the key property (2), assume to the contrary that it is not true for some t with $1 \le t \le g_{i+1}$ and $f(t) \ne n+1$. Then, there must be a subset R' of at most t lines of D_{i+1} such that they cover $C[0,\lambda']$ for some λ' with $\lambda' > \lambda(b_{f(t)})$. Since the key property (2) holds for D_i by our assumption, R' must contain l_{i+1}.

Let $j = g_i$. Since we have proved above that the key property (1) holds for $i+1$, we have $\lambda' \le \lambda(b_j)$. Since l_{i+1} does not intersect $\overline{a_{f(g_i)}b_{f(g_i)}}$ and a_j is above a_{i+1}, l_j is above l_{i+1} on the interval $[0,\lambda(b_j)]$. Hence, for any point $q \in C[0,\lambda(b_j)]$ covered by l_{i+1}, q is also covered by l_j. Therefore, the lines in $S = \{l_j\} \cup R' \setminus \{l_{i+1}\}$ also cover $C[0,\lambda']$. Since $|R'| \le t$, $|S| \le t$. Since $S \subset D_i$ and $\lambda' > \lambda(b_{f(t)})$, we obtain contradiction with that the key property (2) holds for i.

By similar analysis, we can prove the key property (3) as well. The details are omitted.

The second case If l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$ and the special case happens, according to our algorithm, as in the above case, $R_{i+1} = R_i$ and $g_{i+1} = g_i$.

Let p be the point of C to the right of $b_{f(g_i)}$ and infinitely close to $b_{f(g_i)}$. Observe that in this case l_{i+1} is still strictly below p. Therefore, we can use the same analysis as in the above first case to prove the three key properties. We omit the details.

The third case If l_{i+1} intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$ and the special case does not happen, then according to our algorithm, $R_{i+1} = R'_i \cup \{l_{i+1}\}$, where R'_i consists of the first $i'+1$ lines in R_i for some $0 \leq i' \leq g_i$, i.e., $R'_i = \{l_{f(0)}, l_{f(1)}, \ldots, l_{f(i')}\}$. Note that $f(g_{i+1}) = i+1$ and $\lambda(b_{f(g_{i+1})}) \geq \lambda(b_{f(g_i)})$.

According to our algorithm, l_{i+1} intersects $\overline{a_{f(i')}b_{f(i')}}$ and is above $C[b_{f(i')}, b_{i+1}]$. Further, if $i' \geq 1$, either l_{i+1} does not intersect $\overline{a_{f(i'-1)}b_{f(i'-1)}}$, or they intersect but l_{i+1} is not above $C[b_{f(i'-1)}, b_{f(i')}]$.

For the key property (1), assume to the contrary that it is not true. Then, there is a value $\lambda' > \lambda(b_{f(g_{i+1})}) = \lambda(b_{i+1})$ such that $C[0,\lambda']$ is covered by D_{i+1}.

Let p be the point of C to the right of b_{i+1} and infinitely close to b_{i+1}. Hence, $\lambda(b_{i+1}) < \lambda(p) \leq \lambda'$. By the definition of b_{i+1}, l_{i+1} is not above p.

Since the lines of D_{i+1} cover $C[0,\lambda']$ and $\lambda(p) \leq \lambda'$, the lines of D_{i+1} also cover $C[0,\lambda(p)]$. Clearly, D_{i+1} must have a line l' above p. Since l_{i+1} is not above p, l' is in D_i.

Since l' is above p and l_{i+1} is not above p, l' is above l_{i+1} at $\lambda(p)$. Further, since the intersection of l' and l_0 is above a_{i+1}, l' is above l_{i+1} on the interval $[0,\lambda(p)]$. This means that any point $q \in C[0,\lambda(p)]$ covered by l_{i+1}, q is also covered by l'. Therefore, since the lines of D_{i+1} cover $C[0,\lambda(p)]$, the lines of D_i also cover $C[0,\lambda(p)]$. As $\lambda(p) > \lambda(b_{i+1}) \geq \lambda(b_{f(g_i)})$. This contradicts with that the key property (1) holds on i.

This proves that the key property (1) holds on $i+1$.

Next we prove the key property (2). Depending on whether $t = g_{i+1}$ or not, there are two subcases.

1. If $t = g_{i+1}$, then since we have proved that the key property (1) holds for D_{i+1}, and further, since the lines of $R_{i+1} \setminus \{l_0\}$, which has g_{i+1} lines, cover $C[0,\lambda(b_{f(g_{i+1})})]$, the key property (2) trivially follows.
2. If $t \leq g_{i+1}-1$, suppose to the contrary that the key property (2) does not hold for t. Then, $t \geq 1$ and there exists a value λ' larger than $\lambda(b_{f(t)})$ such that $C[0,\lambda']$ is covered by a set R' of at most t lines of D_{i+1}. By our assumption that the key property (2) holds for i, $\lambda(b_{f(t)})$ is the largest value λ in $[0,1]$ such that $C[0,\lambda]$ is covered by at most t lines of D_i. Therefore, l_{i+1} must be in R'.
   Since $t \leq g_{i+1}-1$, $l_{f(t)}$ is in R'_i and $1 \leq t \leq i'$. Recall that either l_{i+1} does not intersect $\overline{a_{f(i'-1)}b_{f(i'-1)}}$, or they intersect but l_{i+1} is not above $C[b_{f(i'-1)}, b_{f(i')}]$. We claim that this is also true for any t with $1 \leq t \leq i'$, i.e., either l_{i+1} does not intersect $\overline{a_{f(t-1)}b_{f(t-1)}}$, or they intersect but l_{i+1} is not above $C[b_{f(t-1)}, b_{f(t)}]$.
   Indeed, if $t = i'$, the claim trivially holds. Otherwise, we first show that the claim holds for $i'-1$. Assume to the contrary that this is not true. Then, l_{i+1} intersects $\overline{a_{f(i'-2)}b_{f(i'-2)}}$ and l_{i+1} is above $C[b_{f(i'-2)}, b_{f(i'-1)}]$. Recall that either l_{i+1} does not intersect $\overline{a_{f(i'-1)}b_{f(i'-1)}}$, or they intersect but l_{i+1} is not above $C[b_{f(i'-1)}, b_{f(i')}]$. We analyze the two cases below.

   a. If l_{i+1} does not intersect $\overline{a_{f(i'-1)}b_{f(i'-1)}}$, then since a_{i+1} is below $a_{i'-1}$, l_{i+1} is below $b_{f(i'-1)}$. On the other hand, since $l_{i'}$ intersects $\overline{a_{f(i'-1)}b_{f(i'-1)}}$, $l_{i'}$ is above $b_{f(i'-1)}$.

Hence $l_{i'}$ is above l_{i+1} at $\lambda(b_{f(i'-1)})$. Further, as $a_{i'}$ is above a_{i+1}, $l_{i'}$ is above l_{i+1} on the interval $[0, \lambda(b_{f(i'-1)})]$.
Consequently, since l_{i+1} intersects $\overline{a_{f(i'-2)}b_{f(i'-2)}}$ and l_{i+1} is above $C[b_{f(i'-2)}, b_{f(i'-1)}]$, $l_{i'}$ also intersects $\overline{a_{f(i'-2)}b_{f(i'-2)}}$ and is above $C[b_{f(i'-2)}, b_{f(i'-1)}]$. But this contradicts with the sixth property of R_{i+1}, which says that either $l_{i'}$ does not intersect $\overline{a_{f(i'-2)}b_{f(i'-2)}}$, or they intersect but $l_{i'}$ is not above $C[b_{f(i'-2)}, b_{f(i'-1)}]$.

b. If l_{i+1} intersects $\overline{a_{f(i'-1)}b_{f(i'-1)}}$ but is not above $C[b_{f(i'-1)}, b_{f(i')}]$, the analysis is similar.
Let p be a point on $C[b_{f(i'-1)}, b_{f(i')}]$ that is above l_{i+1}. Since $l_{i'}$ is above $C[b_{f(i'-1)}, b_{f(i')}]$, $l_{i'}$ is above p. Thus, $l_{i'}$ is above l_{i+1} at $\lambda(p)$. Since $a_{i'}$ is above a_{i+1}, we obtain that $l_{i'}$ is above l_{i+1} on the interval $[0, \lambda(p)]$. As p is on $C[b_{f(i'-1)}, b_{f(i')}]$, $\lambda(p) \geq \lambda(b_{f(i'-1)})$. Therefore, we also obtain that $l_{i'}$ is above l_{i+1} on the interval $[0, \lambda(b_{f(i'-1)})]$.
The rest of the analysis is the same as the first case, and we also obtain contradiction.

This proves that the claim holds for $i' - 1$. By using the similar analysis inductively, we can prove that the claim holds for any t with $1 \leq t \leq i'$.
By the above claim, either l_{i+1} does not intersect $\overline{a_{f(t-1)}b_{f(t-1)}}$, or they intersect but l_{i+1} is not above $C[b_{f(t-1)}, b_{f(t)}]$. We prove the key property (2) for the two cases below.

a. If l_{i+1} does not intersect $\overline{a_{f(t-1)}b_{f(t-1)}}$, let $j = f(t-1)$. Let p be the point on C to the right of b_j and infinitely close to b_j. Hence, $\lambda(b_j) < \lambda(p) \leq \lambda'$. Since a_j is above a_{i+1} and l_{i+1} does not intersect $\overline{a_{f(t-1)}b_{f(t-1)}}$, l_{i+1} is not above p.
Since $C[0, \lambda']$ is covered by R', there must be a line l' in R' that is above p. Clearly, l' is not l_{i+1}. Thus, l' is in D_i. Since l' is above l_{i+1} at $\lambda(p)$ and a_{i+1} is below the intersection of l' and l_0, l' is above l_{i+1} on the interval $[0, \lambda(p)]$. This implies that for any point $q \in C[0, \lambda(p)]$ that is covered by l_{i+1}, q is also covered by l'.
Therefore, the lines of $S = R' \setminus \{l_{i+1}\}$ must cover $C[0, \lambda(p)]$. By our assumption that the key property (2) holds for i, $\lambda(b_{f(t-1)}) = \lambda(b_j)$ is the largest value λ in $[0, 1]$ such that $C[0, \lambda]$ is covered by at most $t - 1$ lines of D_i. Since $|S| \leq t - 1$, $S \subseteq D_i$, and $\lambda(p) > \lambda(b_j)$, the above obtains a set S of at most $t-1$ lines of D_i that cover $C[0, \lambda(p)]$ with $\lambda(p) > \lambda(b_j)$, which incurs contradiction.

b. If l_{i+1} intersects $\overline{a_{f(t-1)}b_{f(t-1)}}$ but l_{i+1} is not above $C[b_{f(t-1)}, b_{f(t)}]$, then let p be a point of $C[b_{f(t-1)}, b_{f(t)}]$ that is strictly above l_{i+1}. Let $j = f(t-1)$.
Since l_{i+1} intersects $\overline{a_j b_j}$ and a_j is above a_{i+1}, l_{i+1} is above b_j. By the definition of p, it holds that $\lambda(p) > \lambda(b_j)$.
Due to $\lambda' > \lambda(b_{f(t)})$ and $\lambda(b_{f(t)}) \geq \lambda(p)$, $\lambda' > \lambda(p)$. Since $C[0, \lambda']$ is covered by the lines of R', $C[0, \lambda(p)]$ is also covered by the lines of R'. Hence, there must be a line l' of R' that covers p. Clearly, l' is not l_{i+1}. Thus, l' is in D_i.
The rest of the analysis follows exactly the same as the above case. We can obtain contradiction again.

The above proves that the key property (2) holds for $i + 1$.

Finally, we prove that the key property (3) holds for $i + 1$.

Consider any t with $1 \leq t \leq g_{i+1}$ and $f(t) \neq n + 1$. Our goal is to show that $\{l_{f(1)}, l_{f(2)}, \ldots, l_{f(t)}\}$ is the smallest subset of D_{i+1} that can cover $C[0, \lambda(b_{f(t)})]$. According to our algorithm, the above set of lines cover $C[0, \lambda(b_{f(t)})]$.

If $t = 1$, then since $\lambda(b_{f(t)}) > 0$, we need at least one line to cover $C[0, \lambda(b_{f(t)})]$. Therefore, (3) trivially holds. In the following, we assume $t \geq 2$.

Suppose to the contrary that (3) is not true. Then, there is a subset $R' \subseteq D_{i+1}$ of at most $t-1$ lines that cover $C[0, \lambda(b_{f(t)})]$. By our assumption that the key property (2) holds on D_i, $\lambda(b_{f(t-1)})$ is the largest value λ in $[0,1]$ such that $C[0,\lambda]$ is covered by at most $t-1$ lines of D_i. According to the eighth property of R_{i+1}, $\lambda(b_{f(t-1)}) < \lambda(b_{f(t)})$. This implies that l_{i+1} must be in R'.

Regardless of whether t is g_{i+1} or not, in light of the claim we have proved above, either l_{i+1} does not intersect $\overline{a_{f(t-2)}b_{f(t-2)}}$, or they intersect but l_{i+1} is not above $C[b_{f(t-2)}, b_{f(t-1)}]$. We analyze the two cases below. The analysis is somewhat similar to that for proving the key property (2), so we briefly discuss it.

1. If l_{i+1} does not intersect $\overline{a_{f(t-2)}b_{f(t-2)}}$, then let $j = f(t-2)$. Let p be the point on C to the right of b_j and infinitely close to b_j. Hence, $\lambda(b_j) < \lambda(p) \leq \lambda(b_{f(t)})$. Since l_{i+1} does not intersect $\overline{a_{f(t-2)}b_{f(t-2)}}$ and a_{i+1} is below $a_{f(t-2)}$, l_{i+1} is not above p.
   Since $C[0, \lambda(b_{f(t)})]$ is covered by the lines of R', there must be a line l' in R' that is above p. Since l_{i+1} is not above p, l' is in D_i. Since l' is above p and l_{i+1} is not, l' is above l_{i+1} at $\lambda(p)$. Further, since the intersection of l' and l_0 is above a_{i+1}, l' is above l_{i+1} on the interval $[0, \lambda(p)]$.
   By the same analysis as that for the key property (2), we can obtain that the lines of $S = R' \setminus \{l_{i+1}\}$ cover $C[0, \lambda(p)]$. Since $|R'| \leq t-1$, $|S| \leq t-2$. Note that $S \subseteq D_i$ and $\lambda(b_j) < \lambda(p)$. However, because the key property (2) holds for i, $\lambda(b_j)$ is the largest value $\lambda \in [0,1]$ such that $C[0,\lambda]$ is covered by at most $t-2$ lines of D_i. Thus, we obtain contradiction.
2. If l_{i+1} intersects $\overline{a_{f(t-2)}b_{f(t-2)}}$ but l_{i+1} is not above $C[b_{f(t-2)}, b_{f(t-1)}]$, then let p be a point of $C[b_{f(t-2)}, b_{f(t-1)}]$ that is strictly above l_{i+1}. Let $j = f(t-2)$.
   Since l_{i+1} intersects $\overline{a_j b_j}$ and a_j is above a_{i+1}, l_{i+1} is above b_j. By the definition of p, it holds that $\lambda(p) > \lambda(b_j)$.
   Note that $\lambda(p) \leq \lambda(b_{f(t-1)}) \leq \lambda(b_{f(t)})$. Since $C[0, \lambda(b_{f(t)})]$ is covered by the lines of R', $C[0, \lambda(p)]$ is also covered by the lines of R'. Hence, there must be a line l' of R' that is above p. Since l_{i+1} is not above p, l' is not l_{i+1} and thus is in D_i. Further, l' is above l_{i+1} at $\lambda(p)$.
   Again, by the similar analysis as before, we can obtain that the lines of $S = R' \setminus \{l_{i+1}\}$ cover $C[0, \lambda(p)]$. We again obtain contradiction as the above first case.

This proves that the key property (3) holds for $i+1$.

The lemma thus follows. ◀

B Proofs of Lemmas for the NP-hardness in Section 4

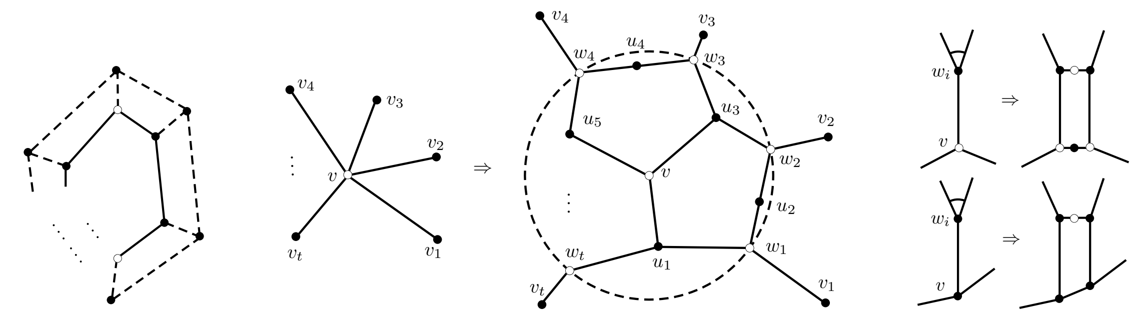
B.1 Proof of Lemma 19

In this section, we show IVC is NP-hard on a normalized PLSG. We first outline the major steps of our reduction, and then explain how we implement these steps in details.

1. VC on convex PLSG G_0 can be reduced to IVC on convex PSLG G_1, by adding an enlarged ring of outer vertices and link each new outer vertex with the original one, as illustarted in Figure 8a
2. IVC on G_1 can be reduced to IVC on convex, low-degree PSLG G_2. For each inner vertex v, we adapt a vertex gadget from [16] shown in Figure 8b. In respect to the neighbor v_i we place w_i on a small circle C_v centered at v, and u_i is the midpoint of $w_{i-1}w_i$. Select

Algorithm 1: The Algorithm D-Greedy-k

```
Input  : $\theta$, the sorted list $D = \{l_1, l_2, \ldots, l_n\}$ after the pruning procedure, and the
         $k$-level set $\mathsf{LS}_k$ of $D$.
Output: A smallest subset $R \subseteq D$ of lines that cover $\theta$-$\mathsf{LS}_k$.
begin
    Compute $C = \theta$-$\mathsf{LS}_k$ based on $\theta$ and $\mathsf{LS}_k$.
    $a_0 \leftarrow$ the highest point of $l_0$ in the infinity.
    $b_0 \leftarrow$ the intersection of $l_0$ and $C$.
    for $i = 0$ to $n$ do
        if $i < n$ then
            $a_{i+1} \leftarrow$ the intersection of $l_0$ and $l_{i+1}$.
        else
            $a_{i+1} \leftarrow$ the intersection of $C$ and $l_{n+1}$.
        if $l_{i+1}$ does not intersect $\overline{a_{f(g_i)} b_{f(g_i)}}$ then
            $R_{i+1} \leftarrow R_i$.
        else
            if $l_{i+1} \cap \overline{a_{f(g_i)} b_{f(g_i)}} = b_{f(g_i)}$ and $b_{i+1} = b_{f(g_i)}$ then
                $R_{i+1} \leftarrow R_i$.
            else
                Compute $b_{i+1}$ and the list $L(b_{f(g_i)}, b_{i+1})$ by moving $p$ rightwards on $C$.
                $t = f(g_i)$.
                while $t > 0$ and $l_{i+1}$ intersects $\overline{a_{f(t-1)} b_{f(t-1)}}$ do
                    if $l_{i+1}$ is above the upper hull of $C[b_{f(t-1)}, b_{f(t)}]$ then
                        Remove $l_{f(t)}$ from $R_i$.
                        Merge $L[b_{f(t-1)}, b_{f(t)}]$ and $L[b_{f(t)}, b_{i+1}]$ to obtain the list
                         $L[b_{f(t-1)}, b_{i+1}]$.
                        $t \leftarrow t - 1$.
                    else
                        break.
                $R_{i+1} \leftarrow R_i \cup \{l_{i+1}\}$.
    return $R \leftarrow R_{n+1} \setminus \{l_0, l_{n+1}\}$.
```



(a) Construction of G_1 **(b)** The vertex gadget for inner vertices **(c)** The edge gadget for small angles (upper: v is inner; lower: v is outer)

Figure 8 Illustrations of PSLG reductions

three distinct vertices among the u_i's and connect them with v, while lifting them closer to meet the angle bound. Proposition 31 in the appendix ensures that there exists such a selection that the three interior angles at v are also in the range of $[\pi/4, \pi]$.

3. IVC on G_2 can be reduced to IVC on convex, low-degree, bounded-angle PSLG G_3. In the construction of G_2 one could see that every interior angle is not larger than π, and those angles equal to π at a degree 3 vertex v can be reduced a little by dragging v off. On the other hand, angles smaller than $\pi/4$ only appear at $\angle u_i w_i u_{i+1}$. To eliminate these angles we replace the opposite edges $w_i v$ by narrow rectangles (or narrow trapezoid if v is an outer vertex), as shown in Figure 8c.
4. IVC on G_3 can be reduced to IVC on a normalized PSLG G_4 for any fixed parameter $\alpha > 0$ by slicing the edges into odd number of small segments. Suppose L_{min} is the smallest length of edges in G_3, and we set the standard length $l = \dfrac{\alpha}{1+\alpha} L_{min}$. Then for each edge of length L with one inner endpoint, slice it into $\lfloor L/l \rfloor$ or $\lfloor L/l \rfloor + 1$ isometric segments by parity. It is clear that the length of these segments are in the range $[l(1-\alpha), l(1+\alpha)]$.

Notice that in the normalized G_4, the parameter α is independent with the angles, and therefore we can replace the range $[\pi/4, \pi)$ by $[\pi/4, \pi - \gamma)$, where $\gamma > 0$ is a constant which will be used in bounds later on.

Here we show the transformations on PSLG described in Section 4 indeed derive reductions. In the following proofs, $G_i = (V_i, E_i)$.

▶ Proposition 30. *$IVC(G_1) = VC(G_0) + |V_1| - |V_0|$.*

Proof. Notice that $|V_1| - |V_0|$ is the number of outer vertices. For an inner vertex cover of G_1, the part within G_0 is still a vertex cover of G_0, therefore $IVC(G_1) \geq VC(G_0) + |V_1| - |V_0|$. On the other hand, on a vertex cover of G_0, adding all the new outer vertices forms an inner vertex cover of G_1. Thus $IVC(G_1) = VC(G_0) + |V_1| - |V_0|$. ◀

▶ Proposition 31. *Within the construction of G_2, for any inner vertex v there are three distinct u_i satisfying the angle bound on v.*

Proof. Illustrated in Figure 9, where we consider the argument of vertices on C_v, and let $\mathrm{Arg}(u_1) = 0$. Consider the first $w_i = w_{i_1}$ which $\mathrm{Arg}(w_i) > \pi$. Without lost of generality we

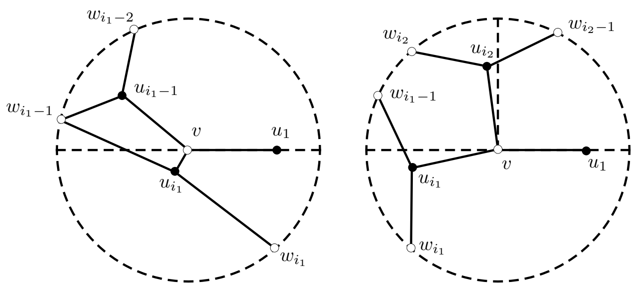


Figure 9 Selection rules of u_i's

can assume that $\mathrm{Arg}(w_{i_1}) - \pi \geq \pi - \mathrm{Arg}(w_{i_1-1})$, and that leads to the bound of $\angle u_1 v u_{i_1}$:

$$\pi \geq \angle u_1 v u_{i_1} \geq \frac{1}{2}(\pi - \mathrm{Arg}(w_{i_1-1})) \geq \frac{\pi}{2}.$$

Now consider two possibilities of $\mathrm{Arg}(u_{i_1})$. If $\mathrm{Arg}(u_{i_1}) \geq \frac{5\pi}{4}$, we just select u_1, u_{i_1} and u_{i_1-1}. In this case we have

$$\pi \geq \mathrm{Arg}(w_{i_1-1}) \geq \angle u_1 v u_{i_1-1} \geq \frac{1}{2}\mathrm{Arg}(w_{i_1-1}) \geq \frac{\pi}{4}$$
$$\pi \geq \angle u_{i_1-1} v u_{i_1} \geq \frac{5\pi}{4} - \pi = \frac{\pi}{4}.$$

Otherwise, consider the first $w_i = w_{i_2}$ which $\mathrm{Arg}(w_i) > \frac{\pi}{2}$, and therefore $i_2 < i_1$. Now we claim that selecting u_1, u_{i_1}, u_{i_2} meets the requirements, since we have

$$\pi \geq \mathrm{Arg}(w_{i_1-1}) \geq \mathrm{Arg}(w_{i_2}) \geq \angle u_1 v u_{i_2} \geq \frac{1}{2}\mathrm{Arg}(w_{i_2}) \geq \frac{\pi}{4}$$
$$\pi \geq \frac{5\pi}{4} - \frac{1}{2}\mathrm{Arg}(w_{i_2}) \geq \angle u_{i_2} v u_{i_1} \geq \frac{1}{2}(\mathrm{Arg}(w_{i_1}) - \mathrm{Arg}(w_{i_2-1})) \geq \frac{\pi}{4}.$$

◀

▶ Proposition 32. $IVC(G_2) = IVC(G_1) + \frac{1}{2}(|V_2| - |V_1|)$.

Proof. Given an inner vertex cover $C_1 \subseteq V_1$ of G_1, we select vertices of the subset $C_2 \subseteq V_2$ by the following rules on every original $v \in V_1$:

- If v is an outer vertex or $\deg(v) = 2$, maintain the status of v.
- Otherwise:
  - If $v \in C_1$, choose v and all related w_i;
  - If $v \notin C_1$, choose all related u_i.

Then one can verify that C_2 is an inner vertex cover of G_2, and

$$IVC(G_2) \leq |C_2| = |C_1| + \frac{1}{2}|\{w_i, u_i\}|.$$

On the other hand, consider any inner vertex cover C_2 of G_2. For each inner vertex $v \in V_1$, denote the set $R(v)$ containing v and all related u_i, w_i (if there are any). It immediately turns out

$$|C_2 \cap R(v)| \geq \begin{cases} (|R(v)| - 1)/2 & \text{if } v \notin C_2 \\ (|R(v)| + 1)/2 & \text{if } v \in C_2. \end{cases}$$

Also, for $(v, v') \in E_1$ where neither v, v is in C_2, the corresponding gadgets are connected with a edge $(w, w') \in E_2$. Without loss of generality, assume that $w \in C_2$. Since the three u_i vertices connected to v must be in C_2, it also holds $|C_2 \cap R(v)| \geq (|R(v)|+1)/2$. Hence all inner vertices $v \in V_1$ satisfyings $|C_2 \cap R(v)| \geq (|R(v)|+1)/2$, along with all outer vertices, form an inner vertex cover of G_1. Therefore

$$IVC(G_1) \leq |C_2| - \sum_v (|R(v)| - 1)/2 = |C_2| - \frac{1}{2}|\{w_i, u_i\}|,$$

and by $|\{w_i, u_i\}| = |V_2| - |V_1|$ the proposition get proved. ◀

▶ Proposition 33. $IVC(G_3) = IVC(G_2) + 2g$, where g is the number of edge gadgets added.

Proof. By exhausting all possibilities on a gadget, it is an easy observation that each edge gadget always add two more vertices in the minimum inner vertex cover, so the result holds. ◀

▶ Proposition 34. $IVC(G_4) = IVC(G_3) + \frac{1}{2}(|V_4| - |V_3|)$.

Proof. Consider an inner vertex cover C of G_4. Among the p_e vertices on an inner edge $e \in G_3$, at least $\frac{1}{2}p_e$ of them should be chosen. Also if neither the two endpoints of e are in C, at least $\frac{1}{2}p_e + 1$ of those padded vertices should be chosen. Thus

$$|C| \geq |V_3 \cap C| + \#\{vw \in E_3 \mid v, w \notin C\} + \frac{1}{2}\sum p_e \geq IVC(G_3) + \frac{1}{2}(|V_4| - |V_3|).$$

On the other hand, C can be constructed from a minimum inner vertex cover of G_3 by alternately select padded vertices, and in this case it is exactly $IVC(G_4) = IVC(G_3) + \frac{1}{2}(|V_4| - |V_3|)$. ◀

This finish the proof of Lemma 19.

B.2 Proof of Lemma 20

Proof. Notice that $\langle \eta(A), \eta(B) \rangle = \rho^2 \cos\angle AOB$, and by the law of cosine we have

$$\begin{aligned}\cos\angle AOB &= \frac{OA^2 + OB^2 - AB^2}{2OA \cdot OB} = \frac{OB + \dfrac{OA^2 - AB^2}{OB}}{2OA} \\ &\geq \frac{\sqrt{OA^2 - AB^2}}{OA} = \sqrt{1 - \frac{AB^2}{OA^2}} \geq \sqrt{1 - \frac{AB^2}{\rho^2 - 1}}.\end{aligned}$$

On the other hand, notice that $2\rho \sin \frac{1}{2}\angle AOB \geq AB$, we have

$$\cos\angle AOB = 1 - 2\sin^2 \frac{1}{2}\angle AOB \leq 1 - \frac{AB^2}{2\rho^2}$$

. ◀

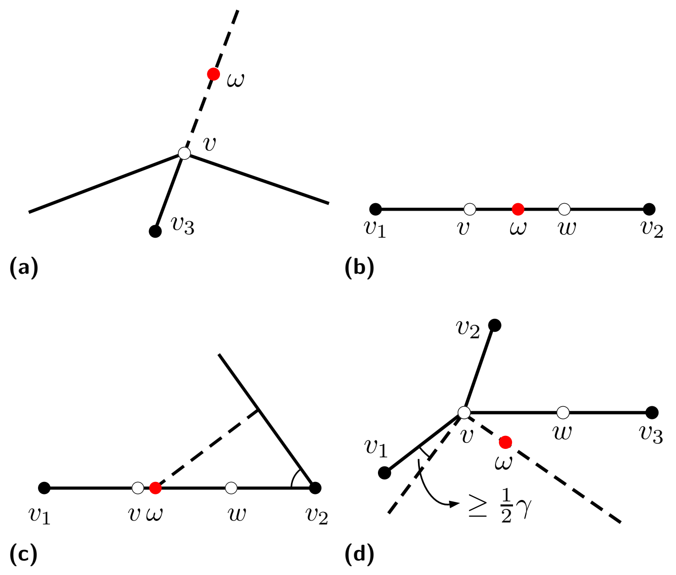


Figure 10 Four possibilities for the selection of ω

B.3 Proof of Lemma 21

Proof. By the definition of k-regret set, we have to prove that

$$\max_{x \in R} \langle x, \omega \rangle \geq \langle (1-\epsilon)x_0, \omega \rangle, \ \forall x_0 \in D, \forall \omega \in \mathbb{R}^3_+. \tag{2}$$

If $x_0 \in R$, this trivially holds; so later on we assume that $x_0 \notin R$. Since $\eta^{-1}(R)$ is an inner vertex cover, $v = \eta^{-1}(x_0)$ is an inner vertex and the adjacent vertices of v are all in $\eta^{-1}(R)$. Consider the degree of v:

- If $\deg(v) = 2$, then x_0 is on the orthodrome between points x_1 and x_2. When $(1-\epsilon)x_0$ is in the triangle $\triangle O x_1 x_2$, the inequality (2) holds for all ω.
- If $\deg(v) = 3$, then x_0 is on a spherical triangle with vertices x_1, x_2 and x_3 by convexity. When $(1-\epsilon)x_0$ is in the tetrahedron $Ox_1x_2x_3$, the inequality (2) holds for all ω.

Notice that when $(1-\epsilon)\rho^2 < \min_i \langle x_0, x_i \rangle$, where $i = 1, 2$ or $i = 1, 2, 3$, the requirements in both cases can be ensured, and according to Lemma 20 we claim that R is an $(1, \epsilon)$-set of D when ϵ satisfies

$$1 - \epsilon \leq \sqrt{1 - \frac{l^2(1+\alpha)^2}{\rho^2 - 1}}.$$

◀

B.4 Proof of Lemma 22

Proof. When $\eta^{-1}(R)$ is not an inner vertex cover, there are only two cases: $\eta^{-1}(R)$ does not choose an outer vertex, or an edge is not covered by $\eta^{-1}(R)$. To prove R is not an $(1, \epsilon)$-set, we shall find $\omega \in \mathcal{S}$ and $x_0 \in D \setminus R$ such that

$$\max_{x \in R} \langle x, \omega \rangle < (1-\epsilon) \langle x_0, \omega \rangle$$

. And with Lemma 20, it can be done by find $\omega \in \mathcal{D}$ and $v \in V_4 \setminus \eta^{-1}(R)$ such that

$$1 - \frac{1}{2\rho^2} \min_{x \in \eta^{-1}(R)} \|x\omega\|^2 < (1-\epsilon)\sqrt{1 - \frac{\|v\omega\|^2}{\rho^2 - 1}}$$

Now suppose $\min \|x\omega\| = k_1 l$, $\|v\omega\| = k_2 l$ and plug in together with the value of ϵ. Notice that ρ can be sufficiently large, so we only need the strict inequality $k_2^2 + (1+\alpha)^2 < k_1^2$ to hold.

1. If in G_4, an outer vertex $v \notin \eta^{-1}(R)$, shown in Figure 10a. Suppose the adjacent vertices of v is v_1, v_2, v_3, where v_1 and v_2 are also outer vertices. Then let ω be the point opposite v_3 at a distance l from v. Then $k_1 \geq 1 + (1-\alpha)$ and $k_2 = 1$.
2. If an edge vw is not covered, that is, $v, w \notin \eta^{-1}(E)$, and $\deg(v) = \deg(w) = 2$, suppose the other two vertices adjacent to v and w are v_1 and v_2.
   - If $\deg(v_1) = \deg(v_2) = 2$ as in Figure 10b, choose ω to be the point on segment vw with $\|v\omega\| = \frac{1}{2}l$. Since the angles attached to inner vertices in G_4 are at least $\frac{\pi}{4}$, it holds that $k_1 \geq \min\{\|v_1\omega\|, \|v_2\omega\|\}/l \geq \frac{3}{2} - 2\alpha$, and $k_2 = \frac{1}{2}$.
   - If $\deg(v_1) = 2$ and $\deg(v_2) = 3$ as in Figure 10c, we choose ω to be point on segment vw with $\|v\omega\| = \frac{1}{5}l$. For the same reason, we have $k_1 \geq \frac{6}{5} - \alpha$, and $k_2 = \frac{1}{5}$.
3. If an edge vw is not covered, and $\deg(v) = 3$, $\deg(w) = 2$, shown in Figure 10d. Suppose the adjacent vertices are v_1, v_2 and v_3. Then ω is set on the reverse bisector of $\angle v_1 v v_2$, at a distance $\frac{1}{2}l$ from v. Since $\angle v_1 v \omega = \angle v_2 v \omega$ are at least $\frac{\pi + \gamma}{2}$, we know that $k_1 \geq \frac{\|v_1\omega\|}{l} \geq \sqrt{\frac{1}{4} + (1-\alpha)^2 + (1-\alpha)\sin\frac{1}{2}\gamma}$, and $k_2 = \frac{1}{2}$.

As a conclusion, we list the following table for the requirements on α in the above four possible cases, from which it is clear to see the validity of the proposition:

$k_1 \geq$	$k_2 =$	α
$2 - \alpha$	1	$\alpha < \frac{1}{3}$
$\frac{3}{2} - 2\alpha$	$\frac{1}{2}$	$\alpha < \frac{4 - \sqrt{13}}{3}$
$\frac{6}{5} - \alpha$	$\frac{1}{5}$	$\alpha < \frac{10}{37}$
$\sqrt{\frac{1}{4} + (1-\alpha)^2 + (1-\alpha)\sin\frac{1}{2}\gamma}$	$\frac{1}{2}$	$\frac{4\alpha}{1-\alpha} < \sin\frac{1}{2}\gamma$

◀

C Other Missing Proofs

C.1 The Proof of Lemma 10

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.

C.2 The Proof of Lemma 11

Notice that by Lemma 10, θ is optimized so that R covers θ-LS_k within $X(D)$. This implies that θ-LS_k and the upper envelop of R coincide at some $\lambda \in X(D)$, so the lemma holds.

C.3 The Proof of Lemma 13

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.

C.4 The Proof of Lemma 14

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.

C.5 The Proof of Lemma 25

Observe that in our setting,

$$\max_{x\in D^{\pm}} \langle x, \omega\rangle - \min_{y\in D^{\pm}} \langle y, \omega\rangle = 2 \max_{x\in D^{\pm}} \langle x, \omega\rangle = 2 \max_{x\in D} \langle x, \mathsf{abs}(\omega)\rangle.$$

The same is true if we replace D and $D^{\pm}$ in the above equation by R and $R^{\pm}$, respectively. Since R' is a subset of $R^{\pm}$, we have

$$\max_{x\in R'} \langle x, \omega\rangle - \min_{y\in R'} \langle y, \omega\rangle \le \max_{x\in R^{\pm}} \langle x, \omega\rangle - \min_{y\in R^{\pm}} \langle y, \omega\rangle = 2 \max_{x\in R} \langle x, \mathsf{abs}(\omega)\rangle.$$

By the definition of ϵ-kernel, for any vector ω, it holds that

$$1-\epsilon \le \frac{\max_{x\in R'} \langle x, \omega\rangle - \min_{y\in R'} \langle y, \omega\rangle}{\max_{x\in D^{\pm}} \langle x, \omega\rangle - \min_{y\in D^{\pm}} \langle y, \omega\rangle} \le \frac{\max_{x\in R} \langle x, \mathsf{abs}(\omega)\rangle}{\max_{x\in D} \langle x, \mathsf{abs}(\omega)\rangle}.$$

The lemma thus follows.

C.6 The Proof of Theorem 27

We present our algorithm for Theorem 27 in Section 5.2.

Suppose there exists a set of r' points in D whose maximum regret ratio is $1-\theta$ for some $\theta \in [0, 1]$. We first present an algorithm that can find a set R of at most $r' \cdot (d \cdot \ln 2n + 1)$ points from D with maximum regret ratio at most $1 - \theta$.

In the d-dimensional space $\mathbb{R}^d$, we fix an arbitrary $(d-1)$-dimensional simplex $\mathcal{A}$ with vertices $A_1, A_2, \ldots, A_d$ whose last coordinates are all 0. For a nonnegative weight vector ω

with $\|\omega\|_1 = 1$, we can regarded it as the *barycentric coordinates* of the point $\sum \omega[i]A_i$ in $\mathcal{A}$. For any point $x \in D$, we define a $(d-1)$-dimensional simplex S_x as the simplex with following vertices:

$$(A_1[1], \dots, A_1[d-1], x[1]), (A_2[1], \dots, A_2[d-1], x[2]), \dots, (A_d[1], \dots, A_d[d-1], x[d]).$$

See Figure 11 for $d = 3$.

For any $\omega \in \mathcal{A}$ (ω in barycentric coordinate), define $\mathcal{R}_\omega$ as the vertical ray originating from ω with direction $(0, 0, \dots, 0, 1)$. An observation is that the ray intersects S_x at a point whose height (i.e., the last coordinate) is exactly equal to $\langle x, \omega \rangle$. Thus we can define *the k-level set* LS_k accordingly: LS_k consists of the k-th highest intersection points of $\mathcal{R}_\omega$ with $\{S_x \mid x \in D\}$, for all $\omega \in \mathcal{A}$ (see Figure 11 for $d = 3$). For convenience, depending on the context, we also consider S_x and LS_k as functions of ω (i.e., as the height of the intersection point with $\mathcal{R}_\omega$).

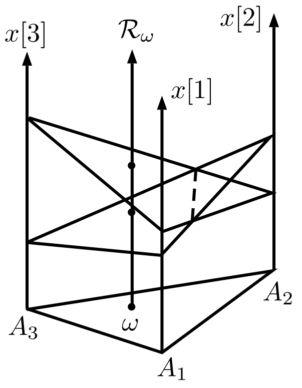


Figure 11 Illustrating two triangles defined by two points x in D in $\mathbb{R}^3$.

For each $x \in D$, let H_x denote the hyperplane containing S_x. For each $x \in D$, let x_θ denote the point whose i-th coordinate is $\theta \cdot x[i]$ for each $i \in [1, d]$, and we define S_{x_θ} and H_{x_θ} accordingly. For a set $\mathcal{H}$ of hyperplanes, we use $\mathbb{A}(\mathcal{H})$ to denote the *arrangement* of $\mathcal{H}$ (See [3] for a survey on arrangements). Using the algorithm in [13], we can compute the hyperplane arrangement $\mathbb{A} = \mathbb{A}(\{H_x, H_{x_\theta}\}_{x \in D})$ in $O((2n)^d)$ time.

We reduce our problem to the set cover problem as follows. There are at most $(2n)^d$ faces with dimension $d-1$ in $\mathbb{A}$. Let the $(d-1)$-faces of $\mathbb{A}$ contained in θ-LS_k be the elements to be covered in the set cover instance. Each set in the set cover instance corresponds to S_x for each $x \in D$, and the elements it covers are those $(d-1)$-faces below S_x. Using the classical greedy algorithm [21], we can solve the set cover problem in $n^{O(d)}$ time with approximation ratio $\ln(2n)^d + 1 = d \ln(2n) + 1$. Let R be the set of points of D corresponding to the solution of the set cover. Recall that there exist a set of r' points in D whose maximum regret ratio is at most $1 - \theta$. The approximation ratio of the set cover solution guarantees that $|R| \leq r' \cdot (d \ln(2n) + 1)$. This finishes our algorithm.

In the following, we use the above algorithm as a subroutine to give an algorithm for Theorem 27

Similar to Lemma 11, the optimal regret ratio $1 - \theta^*$ must correspond to a vertex in the arrangement $\mathbb{A}' = \mathbb{A}(\{S_x\}_{x \in D})$. Formally, the value θ^* must be in the set

$$\mathsf{Cand}(D) := \left\{ \frac{S_x(\omega)}{\mathsf{LS}_k(\omega)} \mid x \in D, \{S_x \cap \mathcal{R}_\omega\} \in V(\mathbb{A}') \text{ or } \omega \in \{A_1, \dots, A_d\} \right\},$$

where $V(\mathbb{A}')$ denotes the set of vertices of $\mathbb{A}'$. Notice that $|\mathsf{Cand}(D)| \leq n \cdot (|\mathbb{A}'| + d) = n^{O(d)}$ and $\mathsf{Cand}(D)$ can be computed in $n^{O(d)}$ time.

For each $\theta \in \mathsf{Cand}(D)$, we apply the above algorithm on θ, and we say that θ is a *feasible value* if the size of the solution R obtained by the algorithm is at most $r \cdot (d \ln(2n) + 1)$. Let θ' be the maximum feasible value. We return the solution R' obtained by our algorithm on θ' as the solution to our original problem k-RMS. On the one hand, since θ' is feasible, we know that $|R'| \leq r \cdot (d \ln(2n) + 1)$. On the other hand, observe that θ^* must be a feasible value. Since θ' is the largest feasible value, we have $\theta' \geq \theta^*$. Therefore, $1 - \theta'$, which is the maximum regret ratio of R', is at most $1 - \theta^*$. Theorem 27 is thus proved.

D Experimental Evaluation

We compare our algorithms in Section 3 and the previous algorithm in [9] for the problems in $\mathbb{R}^2$. All experiments were conducted on a machine with Intel Core i5 CPU @ 2.90GHz and 16GB memory running Mac OS X operation system. The codes were written in C++.

D.1 Algorithms and Datasets

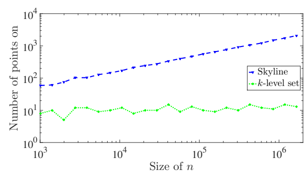
We implemented the following three algorithms for solving the optimization problems RMS and k-RMS: E-Pre-k [9], A-IntCov-1, and A-Greedy-k. For the two approximation algorithms A-IntCov-1 and A-Greedy-k, we set the additive error ϵ to be 10^{-11}, which might be sufficient for most real world applications.

Note that if some point $x \in D$ is Pareto dominated by some other point $x' \in D$ (i.e., $x[1] \leq x'[1]$ and $x[2] \leq x'[2]$ and $x \neq x'$), then x will never appear in the optimal set of R [2]. Those x's which are not Pareto dominated form the *skyline* [4] of D. The algorithms have been optimized to consider only the points on the skyline as candidates for R. Since all algorithms need the k-level set and the skyline information, we do not include the running time for computing the k-level set and skyline information in the running time of each individual algorithm. For the same reason, the input points are already sorted by their slopes.

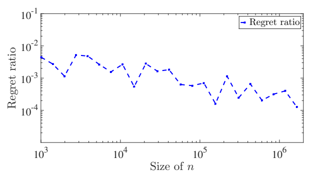
Datasets: We use three datasets. The first two are synthetic, and the last one is a real dataset and can be downloaded from http://www.cru.uea.ac.uk/cru/data/hrg/tmc/.

1. Dataset 1 was generated according to the following procedure: Each point is in the form of $(x, y - \alpha x)$ where x and y are two independent uniform random value in $[0, 1]$. Here, we set $\alpha = 1.75$. Then, a normalization is performed to ensure the points lying in $[0, 1] \times [0, 1]$.
2. Dataset 2 was generated following the procedure described in [4]. It is a fairly standard data generator used in skyline-related settings.
3. Dataset 3 is a real dataset. We use a linear combination of temperature in December, January, and February as the first dimension, and that in June, July and August as the second dimension. In our experiments, we vary n $(= |D|)$. The size of Dataset 3 is equal to n where we chose the first n points in the dataset.

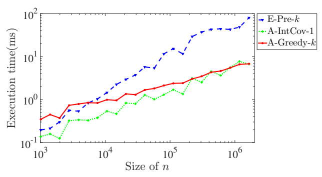
[2] It might appear in the optimal R, but $R \setminus \{x\}$ has the same regret ratio in that case.



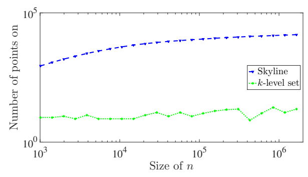
(a) Skyline and k-level: Dataset 1



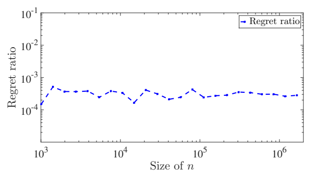
(b) Regret ratio: Dataset 1



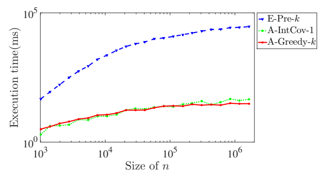
(c) Execution time: Dataset 1



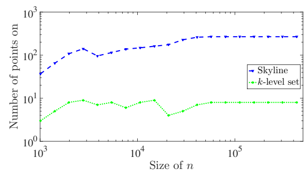
(d) Skyline and k-level: Dataset 2



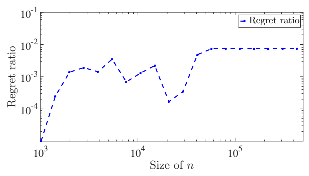
(e) Regret ratio: Dataset 2



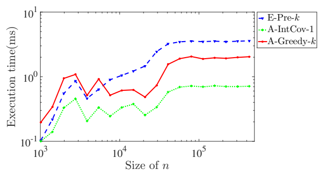
(f) Execution time: Dataset 2



(g) Skyline and k-level: Dataset 3



(h) Regret ratio: Dataset 3



(i) Execution time: Dataset 3

Figure 12 Results of experiments

D.2 Results of Experiments

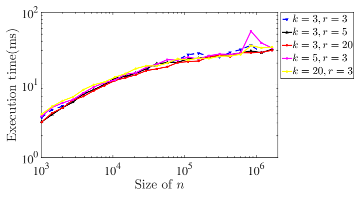
Figure 12 shows the results for $k = 1$ (so we can compare all six algorithms). Each row shows the results on a dataset. The first column indicates the size of the skylines and the k-levels. The second column presents the optimal regret ratios. The third column shows the execution time of each algorithm on each dataset.

We ran our algorithms on different n's, varying from 10^3 to 2×10^6. For each chosen n, three kinds of datasets were generated accordingly and each algorithm was run 10 to 10^3 times depending on the size of the input, while execution times were taken the average. Here, we set $r = 3$. The reason is that this choice yields a regret ratio that can be significantly lower, as shown below. Note that all the plots in Figure 12 are in the log-log scale. We sometimes set a lower bound on plotted data to make them lie within the plots.

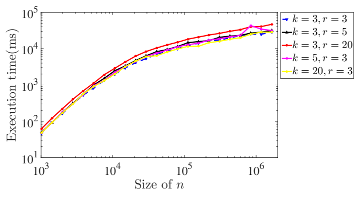
In each of the synthetic datasets, the skyline is as large as from 10^3 to 10^4 when n takes its maximum value, and this helps distinguish the efficiencies of the algorithms. Besides, the regret ratios calculated from each dataset are in a desired range: it is typically as small as from 10^{-4} to 10^{-2}, and this justifies that the k-regret minimizing sets are indeed very good "representations" of the original dataset for all preferences.

In all three datasets, both of our new algorithms beat E-Pre-k. The running time differs from 1.5 to 4 orders of magnitude when n is large.

To further investigate the cases where r and k vary, we try different r and k values and compare E-Pre-k and A-Greedy-k. We use Dataset 2, which is the benchmark in this setting. The results are shown in Figure 13. We can see from the figure that regardless of what r and k are, the running time for A-Greedy-k is at most approximately 100ms (even when n takes its maximum value), while the running time of E-Pre-k exceeds 10^4ms in the corresponding settings. Our algorithm is at least 100 times faster in almost all instances.



(a) When r and k change: A-Greedy-k



(b) When r and k change: E-Pre-k

Figure 13 Further comparison

As a result, we recommend the algorithm A-Greedy-k to be used in practice for the k-RMS problem. The algorithm D-Greedy-k is recommended as the decision algorithm. Hence, the algorithm D-Greedy-k is both theoretically and practically interesting.