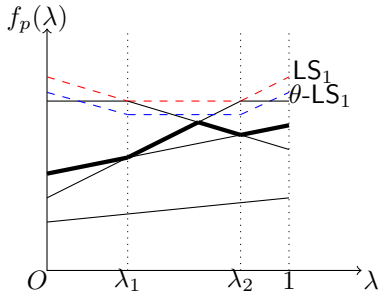


Figure 1 The arrangement $\mathbb{A}(D)$ (black lines), its 1-level set LS_1 (red dashed line), θ-scaled 1-level set (blue lines), and 3-level set (thick black line)

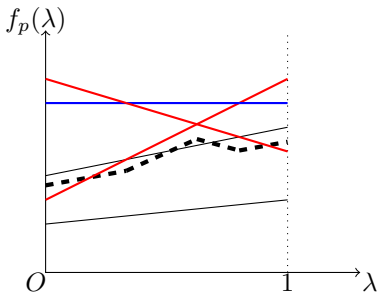


Figure 2 Illustrating Example 2: θ-LS_k is the dashed thick curve

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

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

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

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

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

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

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

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

3.1 The Warm-up Algorithms

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

3.1.1 Reducing Dec-RMS to Interval Coverage

Note that since LS_1 (which is actually the upper envelop of $\mathbb{A}(D)$) is convex, the scaled level set θ-LS_1 is also convex. Also note that $m \le n$ in this case. We can compute LS_1 (and thus θ-LS_1) in $O(n\log n)$ time [16]. We use $\lambda_0 = 0, \lambda_1, \ldots, \lambda_{m-1}, \lambda_m = 1$ to denote all breaking

points of θ-LS_1 (see Fig. 1). Our task is to cover θ-LS_1 using at most r lines. For a line f_p, we let $I(p) = \{\lambda \mid f_p(\lambda) \geq \theta\text{-}\mathsf{LS}_1(\lambda)\}$. The convexity of θ-LS_1 implies that $I(p)$ is a closed interval (which may be empty). Moreover, the interval can be computed in $O(\log n)$ time by binary search.

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

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

We call the above algorithm D-IntCov-1.

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

3.1.2 An Approximating Algorithm for RMS

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

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

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

3.2 The Decision Algorithm for Dec-k-RMS

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

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

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

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

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

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

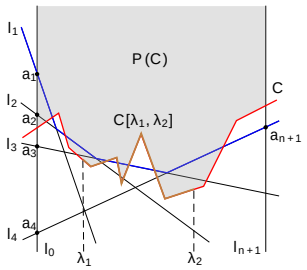


Figure 3 The blue curve denotes $U(\{l_1, l_2, l_3, l_4\})$. The brown curve denotes $C[\lambda_1, \lambda_2]$. The gray area denotes $P(C)$.

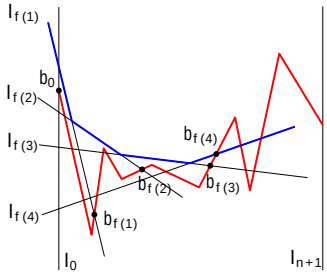


Figure 4 Illustrating the set $R_i = \{l_0, l_{f(1)}, l_{f(2)}, l_{f(3)}, l_{f(4)}\}$ with $g_i = 4$. The red curve is C and the blue curve is $U(R_i)$. The point p is at $b_{f(4)}$.

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

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

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

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

To simplify the notation, we use C to refer to θ-LS$_k$. For any two values λ_1 and λ_2 with $\lambda_1 \leq \lambda_2$, 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 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 the vertical line $\lambda = \lambda'$ and C' is not lower than that of the vertical line $\lambda = \lambda'$ and C. For two points $p_1 \neq p_2$, we use $\overline{p_1 p_2}$ to denote the line segment with endpoints p_1 and p_2. See Figure 3 for an illustration of the definitions given above.

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

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

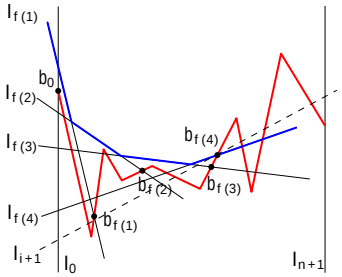


Figure 5 Illustrating the *special case*. Notice that $l_{i+1} \cap \overline{a_{f(4)}b_{f(4)}} = b_{f(4)}$.

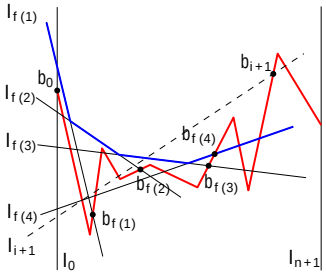


Figure 6 Illustrating the case where l_{i+1} (the dashed line) intersects $\overline{a_{f(g_i)}b_{f(g_i)}}$ with $g_i = 4$. In this example, $R_{i+1} = \{l_0, l_{f(1)}, l_{f(2)}, l_{i+1}\}$ because l_{i+1} intersects $\overline{a_{f(1)}b_{f(1)}}$ but is not above $C[b_{f(1)}, b_{f(2)}]$.

slightly such that it has a positive slope.

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

3.2.1 The Algorithm

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.3 Optimization Algorithms

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

3.3.1 An Exact Algorithm for k-RMS

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

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

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

The following lemma is a consequence of Lemma 9.

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

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

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

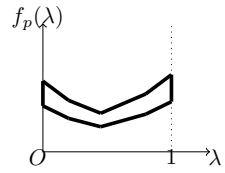
Clearly, the set $\mathsf{Cand}(D)$ consists of at most $|D| \cdot |X(D)| = O(n^3)$ values. To solve the problem k-RMS, we can call the decision algorithm D-Greedy-k to find the largest feasible $\theta \in \mathsf{Cand}(D)$. Computing the set $\mathsf{Cand}(D)$ explicitly would take $\Omega(n^3)$ time.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

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

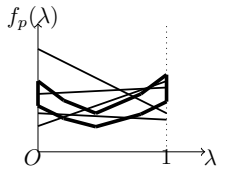
The algorithm consists of multiple stages. In each stage, we use a line sweeping algorithm on λ, keeping track of the lines l of D ordered by $l(\lambda)$. For the i-th time the sweeping line hits a $\lambda_i \in X(D)$, we compute a value θ_i of θ (according to Lemma 10) determined by the line ranked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

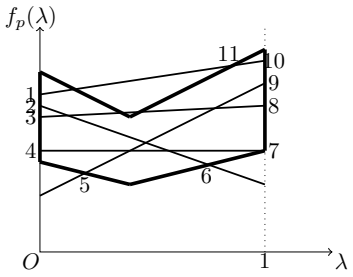
$$\sum\{w_i \mid \theta_i < \theta_m\} \le \frac{1}{2}\sum_i w_i < \sum\{w_i \mid \theta_i \le \theta_m\}.$$

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



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





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

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

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

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

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

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

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

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

We proceed as above until there are at most n candidate values in the search range (i.e., $N \leq n$). Finally, we run the following *post-processing step*. Let U be the union of the set of

these at most n candidate values in the search range and $\mathsf{Cand}_1(D)$. By the above algorithm, the optimal value θ is in U. Since $|\mathsf{Cand}_1(D)| = O(n)$, $|U| = O(n)$. We sort and perform a binary search on the values of U using the decision procedure D-Greedy-k to eventually compute the optimal θ value. This finishes our algorithm.

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

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

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

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

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

4 NP-Hardness

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

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

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

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

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

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

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