Computing k-Regret Minimizing Sets

Sean Chester, Alex Thomo, S. Venkatesh, and Sue Whitesides
Computer Science Department
University of Victoria
PO Box 1700 STN CSC
Victoria, Canada, V8W 2Y2
{schester,sue}@uvic.ca, {thomo,venkat}@cs.uvic.ca

ABSTRACT

Regret minimizing sets are a recent approach to representing a dataset D by a small subset R of size r of representative data points. The set R is chosen such that executing any top-1 query on R rather than D is minimally perceptible to any user. However, such a subset R may not exist, even for modest sizes, r. In this paper, we introduce the relaxation to k-regret minimizing sets, whereby a top-1 query on R returns a result imperceptibly close to the top-k on D.

We show that, in general, with or without the relaxation, this problem is NP-hard. For the specific case of two dimensions, we give an efficient dynamic programming, plane sweep algorithm based on geometric duality to find an optimal solution. For arbitrary dimension, we give an empirically effective, greedy, randomized algorithm based on linear programming. With these algorithms, we can find subsets R of much smaller size that better summarize D, using small values of k larger than 1.

1. INTRODUCTION

For a user navigating a large dataset, the availability of a succinct representative subset of the data points is crucial. For example, consider Table 1, D_{nba}, a toy, but real, dataset consisting of the top eight scoring NBA players from the 2009 basketball season. A user viewing this data would typically be curious which of these eight players were "top of the class" that season. That is, he is curious which few points best represent the entire dataset, without his having to peruse it in entirety.

A well-established approach to representing a dataset is with the *skyline* operator [2] which returns all pareto-optimal points.[1] The intention of the skyline operator is to reduce the dataset down to only those points that are guaranteed to best suit the preferences or interests of *somebody*. If the toy dataset in Table 1 consisted only of the attributes *points* and *rebounds*, then the skyline would consist only of the players Kevin Durant, Amare Stoudemire, and Zach Randolph. So, these three players would represent well what are the most impressive combinations of point-scoring and rebounding statistics. However, the skyline is a powerful summary operator only on low dimensional datasets. Even for this toy example, *everybody* is in the skyline if we consider all four attributes. In general, there is no guarantee that the skyline is an especially succinct representation of a dataset.

id	player name	points	rebs	steals	fouls
1	Kevin Durant	**2472**	623	112	171
2	LeBron James	2258	554	125	119
3	Dwyane Wade	2045	373	**142**	181
4	Dirk Nowitzki	2027	520	70	208
5	Kobe Bryant	1970	391	113	187
6	Carmelo Anthony	1943	454	88	225
7	Amare Stoudemire	1896	732	52	**281**
8	Zach Randolph	1681	**950**	80	226

Table 1: D_{nba}. Statistics for the top NBA point scorers from the 2009 regular season, courtesy `databasebasketball.com`. The top score in each statistic is bolded.

1.1 Regret minimizing sets

A promising new alternative is the *regret minimizing set*, introduced by Nanongkai et al. [18], which hybridizes the skyline operator with top-k queries. A top-k query takes as input a weight vector $\mathbf{w}$ and scores each point by inner product with $\mathbf{w}$, reporting the k points with highest scores. For example, on weights $\langle .5, .5, 0, 0\rangle$, Randolph earns the highest normalized score ($1681/2472 * .5 + 950/950 * .5 = 0.840$), compared to Kevin Durant next (0.828) and then Amare Stoudemire (0.769). So the top-2 query returns Randolph and Durant.

To evaluate whether a subset effectively represents the entire dataset well, Nanongkai et al. introduce *regret ratio* as the ratio of how far from the best score in the dataset is the best score in that subset. For $S = \{\text{Stoudemire}, \text{Durant}\}$, the regret ratio on a top-1 query $\langle .5, .5, 0, 0\rangle$ is:

$$(0.840 - 0.828)/0.840 = 0.0143,$$

since the score for Randolph is the best in the dataset at 0.840, and the score for Durant is the best in the subset at 0.828. Hence, a user would be 98.57% happy if executing that top-1 query on S rather than all of D_{nba}.

Motivated to derive a succinct representation of a dataset, one with fixed cardinality, Nanongkai et al. introduce *regret minimizing sets* [18],[2] posing the question, "Does there exist one set of r points

[1] Pareto-optimal points are those for which no other point is higher ranked with respect to every attribute.

[2] In [18], Nanongkai et al. call this a k-*regret minimizing set*, but we instead refer to their concept as a 1-regret minimizing set of size r. We explain this choice at the end of Section 2, when the rationale will be clearer.

Proceedings of the VLDB Endowment, Vol. 7, No. 5

id	normalized points x	y	scores for given weight vectors $\mathbf{w}_0 = \langle 1.00, 0.00\rangle$	$\mathbf{w}_1 = \langle 0.72, 0.28\rangle$	$\mathbf{w}_2 = \langle 0.52, 0.48\rangle$	$\mathbf{w}_3 = \langle 0.32, 0.68\rangle$	$\mathbf{w}_4 = \langle 0.00, 1.00\rangle$
Durant	1.00	0.66	1.00	0.90	0.84	0.77	0.66
Stoudemire	0.77	0.77	0.77	0.77	0.77	0.77	0.77
Randolph	0.68	1.00	0.68	0.77	0.83	0.90	1.00
James	0.91	0.58	0.91	0.82	0.75	0.69	0.58

(a) The scores for the normalized Durant, Stoudemire, Randolph, and James points on five different weight vectors, each given in a separate column.

Set	1-regret ratio $\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$	$\mathbf{w}_4$	max	2-regret ratio $\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$	$\mathbf{w}_4$	max
{Durant}	.00	$\frac{\max(0,0.90-0.90)}{0.90} = .00$	.00	.14	.34	.34	$\frac{\max(0,0.91-1.00)}{0.91} = .00$	.00	.00	.00	.14	.14
{Stoudemire}	.23	$\frac{\max(0,0.90-0.77)}{0.90} = .14$	.08	.14	.23	.23	$\frac{\max(0,0.91-0.77)}{0.91} = .15$	.06	.07	.00	.00	.15
{Randolph}	.32	$\frac{\max(0,0.90-0.77)}{0.90} = .14$	.00	.00	.00	.32	$\frac{\max(0,0.91-0.68)}{0.91} = .25$	.06	.00	.00	.00	.25
{Durant,Stou.}	.00	$\frac{\max(0,0.90-0.90)}{0.90} = .00$	.00	.14	.23	.23						
{Durant,Rand.}	.00	$\frac{\max(0,0.90-0.90)}{0.90} = .00$	.00	.00	.00	.00						
{Stou.,Rand.}	.23	$\frac{\max(0,0.90-0.77)}{0.90} = .14$	.00	.00	.00	.23						

(b) The 1-regret and 2-regret ratios computed on each of the five vectors in (a) for the size 1 and size 2 subsets with Durant, Stoudemire and/or Randolph.

Table 2: A small example of the kRMS problem. In (a), we show scores for five vectors on a few points from Table 1 that have been normalized and, to simplify the example, projected on the first two attributes. In (b), we calculate the 1-regret ratio and 2-regret ratio for some subsets of data points. The calculation from Definition 2.2 is shown for $\mathbf{w}_1$ for 1RMS and $\mathbf{w}_0$ for 2RMS. (2RMS sets of size 2 are not shown because the example is small.) The size 1 2RMS solution, {Durant}, and 1RMS solution, {Stoudemire}, differ.

that makes every user at least $x\%$ happy (i.e., returns within $x\%$ of correct on any top-1 query)?"

As an example, taking $r = 1$ and projecting on just the attributes *points* and *rebounds*, Amare Stoudemire is the player closest to the top-ranked choice on a worst case user weight vector (as will be clear in the example of Table 2 that we go through in Section 2). Still, however, he scores 23% below Durant on the query $\langle 1.00, 0.00\rangle$. This exposes a weakness of regret minimizing sets: they are forced to fit a very rigid criterion for user satisfaction, that a "happy" user is one who obtains his absolute top choice. However, for an analyst curious to know who is a high point-scoring basketball player, is he really dissatisfied with the second choice, LeBron James, as a query response rather than Durant?

In practice, one often does not get the theoretical top choice, anyway. To change the scenario a bit, consider a dataset of hotels and a user searching for one that suits his preferences. The absolute top theoretical choice may not suit him especially well at all. It could be fully booked. Or, he may have been dissatisfied when he stayed there previously. For a user like him, the regret minimizing set is rigidly constructed on an intangibly poor choice, even if that choice was theoretically far superior.

To alleviate these problems, we soften the happiness criterion to a second or third or fourth "best" point, smoothening out the outliers in the dataset. As a result, on this small example, we can select another player (Durant) and come within 14% of everyone's second choice. More impressively, with just eight data points (from the entire basketball dataset and six dimensions, not just the eight players in Table 1 and two dimensions in this example), we can be within 10% of everyone's third choice, but only within 30% of everyone's top choice. This *k-regret minimizing set* can more succinctly represent the entire dataset than just 1-regret can.

1.2 Contributions

After introducing *k-regret minimizing sets*, we focus on two broad questions: (how) can one compute a k-regret minimizing set? We use a diverse toolkit to resolve these questions, including geometric duality, plane sweep, dynamic programming, linear programming, randomization, and reduction. In particular, we:

- generalize *regret ratio*, a top-1 concept, to *k-regret ratio*, a top-k concept, for quantifying how well a subset of a dataset appeals to users (Section 2);
- resolve a conjecture by Nanongkai et al. [18], that finding 1-regret minimizing sets is an NP-hard problem, and extend the result for k-regret minimizing sets (Section 3);
- introduce an $\mathcal{O}(n^2 r)$ plane sweep, dynamic programming algorithm to compute the size-r k-regret minimizing subset S of a two-dimensional dataset D for any k (Section 4); and
- introduce a randomized greedy linear programming algorithm for the NP-hard case of arbitrary dimension (Section 5) that we show performs very well on experiments (Section 6).

2. K-REGRET MINIMIZING SETS

In this section, we give the definitions needed for this paper. The definitions build towards our introduction of *k-regret ratio*, a generalisation of *regret ratio* [18], which measures how far from a k'th "best" tuple is the "best" tuple in a subset. We conclude the section with the statement of our problem, kRMS. Throughout, we use the example in Table 2 to demonstrate application of these definitions.

To begin, we describe the basic notation, which is summarized in Table 3. We assume a dataset D of n points in d dimensions, with each dimension scaled to the range $[0, 1]$. We denote the i'th point by p^i and the j'th coordinate of p^i by p^i_j. We are particularly interested in evaluating a subset $R \subseteq D$ as an approximation to D. We consider "linear scoring semantics" wherein a user specifies a vector of real-valued attribute weights, $\mathbf{w} = \langle \mathrm{w}_0, \ldots, \mathrm{w}_{d-1}\rangle$ and a point $p \in D$ has a score on $\mathbf{w}$: $\mathrm{score}(p, \mathbf{w}) = \sum_{i=0}^{d-1} p_i \mathrm{w}_i$. If one breaks ties arbitrarily, D can be sorted with respect to $\mathbf{w}$ in

symbol	definition
D	An input dataset
n	$\|D\|$, the number of input points
d	The number of dimensions
k	The rank of a point on a given query
p^i	The i'th point in D
p^i_j	The j'th coordinate of p^i
R	A subset of D
r	$\|R\|$, the number of elements in R
$\mathbf{w}$	A user weight vector
$\mathbf{w}_i$	The i'th weight vector in a set
w_i	The i'th component of vector $\mathbf{w}$
$D^{(k,\mathbf{w})}$	The k'th ranked point in D on $\mathbf{w}$
$R^{(k,\mathbf{w})}$	The k'th ranked point in R on $\mathbf{w}$

Table 3: Notation commonly used throughout this paper

descending order of score, producing a list $(D^{(1,\mathbf{w})}, \ldots, D^{(n,\mathbf{w})})$. That is to say, we denote by $D^{(k,\mathbf{w})}$ (or by $R^{(k,\mathbf{w})}$) the point $p \in D$ (or $p \in R$) with the k'th highest score, and refer to it as the k-ranked point on $\mathbf{w}$. From the example in Table 2, $D^{(1,\mathbf{w}_1)}$ is Durant, since he has the highest score on $\mathbf{w}_1$ (.90) and $D^{(2,\mathbf{w}_1)}$ is James, since he has the second highest score.

We now introduce the terms specific to k-regret minimizing sets. First, for each vector of user weights, $\mathbf{w}$, we define the $k$gain of a subset $R \subseteq D$, denoted $k\mathrm{gain}(R, \mathbf{w})$ as an alias for *the score of the k-ranked point among R*:

DEFINITION 2.1 ($k$gain).

$$k\mathrm{gain}(R, \boldsymbol{w}) = \mathrm{score}(R^{(k,\boldsymbol{w})}, \boldsymbol{w}).$$

Returning to Table 2, $2\mathrm{gain}(\{Randolph, Stoudemire\}, \mathbf{w}_0) = 0.68$, because Stoudemire is the second-ranked point in the set, and $2\mathrm{gain}(\{James, Randolph, Stoudemire\}, \mathbf{w}_0) = 0.77$, since the score for James on $\mathbf{w}_0$ demotes Stoudemire to second-ranked.

We compare subsets of D based on their 1gain, relative to the $k$gain of the original dataset. Given $\mathbf{w}$ and a subset $R \subseteq D$, we define our distance metric for subsets, the *k-regret ratio* as follows:

DEFINITION 2.2 (k-REGRET RATIO). *Given a subset $R \subseteq D$ and a vector of weights, $\boldsymbol{w}$, the k-regret ratio is:*

$$k\text{-regratio}(R, \boldsymbol{w}) = \frac{\max(0, k\mathrm{gain}(D, \boldsymbol{w}) - 1\mathrm{gain}(R, \boldsymbol{w}))}{k\mathrm{gain}(D, \boldsymbol{w})}.$$

Note that the ratio must fall in the range $[0, 1]$. The bottom part of Table 2 shows the calculation of k-regret ratios for different subsets. As an example, the 1-regret ratio of {Randolph,Stoudemire} on $\mathbf{w}_1$ is 0.14 because the highest score in the dataset on $\mathbf{w}_1$ is 0.90 and the highest score among Randolph and Stoudemire is 0.77. So {Randolph,Stoudemire} is within $(0.90-0.77)/0.90$ of the $k = 1$-ranked response to $\mathbf{w}_1$.

Our objective in this problem is to minimize the worst case; so, we measure the *maximum k-regret ratio* for a subset, considering all possible weight vectors, $\mathbf{w} \in [0, 1]^d$.

DEFINITION 2.3 (MAXIMUM k-REGRET RATIO). *Let $\mathcal{L}$ denote all vectors in $[0, 1]^d$. Then the* maximum k-regret ratio *for a subset $R \subseteq D$ is:*

$$k\text{-regratio}(R) = \sup_{\boldsymbol{w} \in \mathcal{L}} k\text{-regratio}(R, \boldsymbol{w}).$$

To finish the example in Table 2, we determine the maximum k-regret ratio for each of the subsets by looking for the highest value among the five vectors. (We know that checking just these five vectors is sufficient based on ideas we present in Section 4.) So, the maximum 1-regret ratio for the three singleton sets, {Durant}, {Stoudemire}, and {Randolph} is 0.34, 0.23, and 0.32, respectively, of which the value for {Stoudemire} is the smallest.

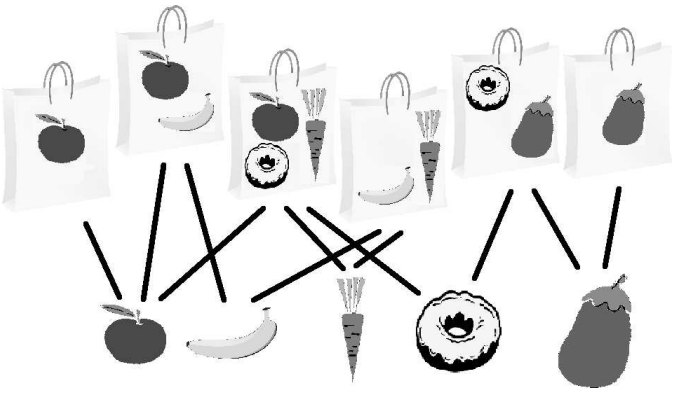


Figure 1: An example of the set cover problem. Choose at most r of the grocery bags above such that the union of all r grocery bags contains everything in the five-item grocery list below.

This brings us to the primary objective in this paper, to produce a fixed-size *k-regret minimizing set*. Given an integer r and a dataset D, discover a subset $R \subseteq D$ of size r that achieves the minimum possible maximum k-regret ratio, as {Stoudemire} did above.

DEFINITION 2.4 (k-REGRET MINIMIZING SET). *A k-regret minimizing set of order r on a dataset D is:*

$$R_{r,D} = \mathrm{argmin}_{R \subseteq D, |R|=r} k\text{-regratio}(R).$$

Problem Definition 1. [kRMS] Given a set D of n points in d dimensions and an integer r, return a k-regret minimizing set of size r on D.

As a couple of final remarks, first note that kRMS is a class of problems, and each value of k is a distinct problem. If $k = 1$, all these definitions, including 1RMS, reduce to analogous ones introduced by Nanongkai et al. [18]. Finally, we merge two ideas, top-k queries and k-regret minimizing sets, each of which uses k with different meaning. Because top-k queries are the more established and familiar field, we preserve their notation, replacing the use of k in Nanongkai et al. [18] with the variable r.

3. REGRET MINIMIZATION IS HARD

We begin by resolving a conjecture by Nanongkai et al. [18] that 1RMS is hard.

THEOREM 1. 1RMS *is NP-Hard.*

We prove Theorem 1 by means of a reduction from the SET-COVER decision problem. The SET-COVER decision problem is, from a finite collection of subsets T of a finite universe $\mathcal{U}$, to determine if one can select r subsets from T so that every element of $\mathcal{U}$ is included. SET-COVER was shown to be NP-Complete by Karp [11]. Note that in the following definition, $\mathcal{P}(\mathcal{U})$ denotes the power set of $\mathcal{U}$.

Problem Definition 2. [SET-COVER] Given three inputs, a finite universe $\mathcal{U}$, a set $T \subseteq \mathcal{P}(\mathcal{U})$, and an integer r, is there a size r subset $S = \{S_0, \ldots, S_{r-1}\}$ of T such that $\bigcup_{S_i \in S} S_i = \mathcal{U}$?

id	a	b	c	d	e
d_0	1	0	0	0	0
d_1	0	1	0	0	0
d_2	0	0	1	0	0
d_3	0	0	0	1	0
d_4	0	0	0	0	1
p_0	.2	0	0	0	0
p_1	.2	.2	0	0	0
p_2	.2	0	.2	.2	0
p_3	0	.2	.2	0	0
p_4	0	0	0	.2	.2
p_5	0	0	0	0	.2

Table 4: An instance of 1RMS corresponding to the set cover problem in Figure 1. Each axis point d_0 to d_4 corresponds to an element of $\mathcal{U}$ and each mapped point p_0 to p_5 corresponds to an element of T. Any subset of r points with a maximum regret of $1 - 1/|\mathcal{U}| = 0.8$ is a solution to the original set cover problem. If no such subset exists, no solution exists in Figure 1.

At a high level, we reduce an instance of SET-COVER=$(\mathcal{U}, T, r)$ to an instance of 1RMS=(D, r) as follows. We construct D with $n = |\mathcal{U}| + |T|$ points in $d = |\mathcal{U}|$ dimensions. The first $|\mathcal{U}|$ "axis" points correspond to elements of $\mathcal{U}$ and the last $|T|$ "mapped" points of D correspond to elements of T. The coordinate values are chosen appropriately, higher for axis points than for mapped points, so that a 1RMS solution on D then maps back to a solution of the SET-COVER problem.

The formal details follow shortly, but first we illustrate the reduction with an example of SET-COVER in Figure 1 and a corresponding $|\mathcal{U}|$-dimensional 1RMS instance in Table 4. For each element of $\mathcal{U}$, one creates a unique axis point (d_0 to d_4 in Table 4). For example, "banana" becomes $d_1 = (0, 1, 0, 0, 0)$ and "eggplant" becomes $d_4 = (0, 0, 0, 0, 1)$. For each subset S_i in T, one creates an additional point (p_0 to p_5 in Table 4). Each coordinate p_i^j is 0 if the j'th element is not in S_i and is $|\mathcal{U}|^{-1}$ if it is. For example, $\{\text{banana}, \text{carrot}\}$ becomes $p_3 = (0, .2, .2, 0, 0)$. This forms the dataset for 1RMS.

To be more precise, we build a reduction from the decision SET-COVER problem defined in Problem Definition 2. Given an instance $\mathcal{I}_{\text{SC}} = (\mathcal{U}, T, r)$ of SET-COVER, we produce an instance $\mathcal{I}_{\text{RMS}} = (D, r)$ of regret minimization as follows. For every element $e_i \in \mathcal{U}$, construct an *axis point* $d_i = (d_i^0, \ldots, d_i^{|\mathcal{U}|-1})$, where $d_i^j = 1$ if $i = j$ and $d_i^j = 0$ otherwise. For every element $S_i \in T$, construct a *mapped point* $p_i = (p_i^0, \ldots, p_i^{|\mathcal{U}|-1})$, where $p_i^j = |\mathcal{U}|^{-1}$ if the j'th element of $\mathcal{U}$ is in S_i and $p_i^j = 0$ if it is not. Let D be the set of all axis and mapped points so constructed. Let $\mathcal{I}_{\text{RMS}}$ be the instance of regret minimization produced with D and the same value of r as in $\mathcal{I}_{\text{SC}}$. This completes the reduction, which runs in polynomial time (Proposition 3.1).

PROPOSITION 3.1. *The reduction from SET-COVER to 1RMS takes $\mathcal{O}(|\mathcal{U}| + |T|)$ time for the construction of $\mathcal{I}_{\text{RMS}}$.*

PROOF OF THEOREM 1. We now use the reduction described above to prove Theorem 1. We show that there are only three possible maximum regret ratios that can be produced by an instance or 1RMS constructed with this reduction (Lemma 3.2), and a yes or no decision for $\mathcal{I}_{\text{SC}}$ each corresponds exactly to those cases (Lemma 3.3).

LEMMA 3.2. *Given $\mathcal{I}_{\text{RMS}} = (D, r)$, any subset $R \subseteq D$ has* $1\text{-regratio}(R) \in \{0, 1, 1 - d^{-1}\}$.

PROOF. First, note that $\forall \mathbf{w}$, the top-ranked point is an axis point. Precisely, if j is the largest coordinate of $\mathbf{w}$, then $D^{(1,\mathbf{w})} = d_j$, because $\forall p_i \in D, \sum_{h=0}^{d-1} w_h p_i^h \leq w_j$. (The coordinate values of each p_i were chosen specifically to guarantee this condition.)

Second, note that for any subset $R \subseteq D$, either 1) all axis points are in R; or 2) $\exists d_j \notin R$, where d_j is some j'th axis point. This second case refines further: 2a) $\exists d_j \notin R, \forall p_i \in R, p_i^j = 0$; and 2b) $\forall d_j \notin R, \exists p_i \in R$ with $p_i^j = d^{-1}$. We analyse each of the three cases in order.

In case 1), $1\text{-regratio}(R) = 0$, because every top-ranked point is an axis point, and every axis point is in R. In case 2a), let d_j be an axis point fulfilling the case condition. The weight vector $\mathbf{w}$ formed by setting the j'th coordinate to 1 and every other coordinate to 0, establishes $1\text{-regratio}(R, \mathbf{w}) = 1$, the maximum possible value for a 1-regret ratio, so $1\text{-regratio}(R) = 1$. In the final case, 2b), consider a weight vector $\mathbf{w}$ that maximizes the expression $1 - \frac{\mathbf{w} \cdot p_i}{\mathbf{w} \cdot d_j}$, for some $d_j \notin R$. Clearly, the j'th coordinate of $\mathbf{w}$ must have some non-zero value, c, or else $\mathbf{w} \cdot d_j = 0$. But any non-zero value on any other coordinate of $\mathbf{w}$ can only increase the numerator without increasing the denominator, since the denominator, $\mathbf{w} \cdot d_j$, has only the one non-zero coordinate. So, $\mathbf{w}$ must be constructed as in case 2a). Then, $1\text{-regratio}(R, \mathbf{w}) = 1 - \frac{cp_i^j}{c} = 1 - d^{-1}$. So, in conclusion, the maximum regret ratio must take on one of the three values: $\{0, 1, 1 - d^{-1}\}$. □

LEMMA 3.3. *An instance $\mathcal{I}_{\text{SC}}$ has a set cover of size r if and only if the corresponding instance $\mathcal{I}_{\text{RMS}}$ has a 1-regret minimizing set of size r with 1-regret ratio < 1.*

PROOF. Without loss of generality, assume that we do not have a trivial case of set cover where either $r \geq |\mathcal{U}|$ or $\exists u \in \mathcal{U} : \forall S_i \in S, u \notin S_i$, since both cases are easily resolved in polynomial time. Then:

If: Let R be a solution to $\mathcal{I}_{\text{RMS}}$ with a 1-regret ratio of $1 - d^{-1}$. Then $\mathcal{I}_{\text{SC}}$ has a set cover, namely the one containing the set S_i for every mapped point p_i and a set $S_h \in S$ containing j for each axis point $d_j \in R$.

Only if: Assume that there is a set cover S of size r on $\mathcal{I}_{\text{SC}}$. Every subset $S_h \in S$ has a corresponding mapped point $p_i \in D$ with $p_i^j = d^{-1}$ for all $j \in S_h$. So, since S covers every $u \in \mathcal{U}$, then for every dimension j, some p_i has a non-zero p_i^j. The regret ratio is maximized on the axes; so, the maximum regret ratio is at most $1 - d^{-1}$. But from Lemma 3.2, the only other possible value is 0, which corresponds to one of the trivial cases. □

Therefore, we have built a correct polynomial-time reduction from SET-COVER to 1RMS. This completes the proof of Theorem 1. □

To see an example of the case correspondence in Lemma 3.3, take any two points p and p' in Table 4. There will be some j'th dimension not "covered" by p nor p' and setting $\mathbf{w}$ identical to d_j will produce a regret ratio of 1. For $p = p_2$ and $p' = p_5$, for example, d_1 is one such axis point. Notice, too, that there are no two grocery bags in Fig. 1 that cover the grocery list. On the other hand, $R = p_1, p_3, p_4$ has a regret ratio of $1 - d^{-1} = 0.8$ and the set $\{s_1, s_3, s_4\}$ in Fig. 1 covers the complete grocery list.

COROLLARY 3.4. *kRMS is NP-Hard.*

PROOF SKETCH. The reduction proceeds analogously to that of Theorem 1, except that we create k axis points for every element of $\mathcal{U}$ rather than just one. Then, the rank of every mapped point is k, there is no benefit in selecting multiple copies of axis points, and the other details of the proof remain the same.

4. A CONTOUR VIEW OF REGRET

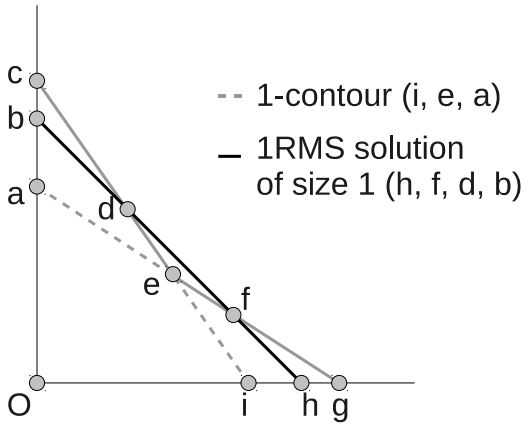


Figure 2: An illustration of 1RMS is dual space. Of the three dual lines, (i, c), (h, b), and (g, a), it is (h, b) that has the minimum maximum distance ratio with respect to the dotted, top-1 contour, (i, e, a). Therefore, it is the best solution of size 1. The maximum ratio for (h, b) occurs on the positive y-axis, where it is given by the length of (a, b) divided by the length of (O, b).

The previous section showed that it is NP-Hard to compute a k-regret minimizing set in arbitrary dimension. Now, we focus on the specific case of two dimensions, and offer an efficient, exact algorithm, consequently implying that kRMS$\in$P for the case $d = 2$. Figure 2 illustrates the main insight of this section, that solving kRMS is equivalent to a *dual-space* geometric problem of finding a "convex chain," like the black line in the figure, that is "closest" to the "top-k rank contour," the dotted line in the figure.

This section has two parts. In Section 4.1, we formally define *dual space*, *convex chains*, *top-k rank contours*, and our meaning of the term "closest." In Section 4.2, we present the algorithm.

4.1 Convex chains and contours

Throughout Section 4, we will work in the *dual space* of [6]. That is to say, all points in D are transformed into lines in a set $\mathcal{L}(D)$,[3] as in Figure 3. For a point $p_i \in D$ construct the line $l_i \in \mathcal{L}(D)$ as $p_i^0 x + p_i^1 y = 1$, (or, in slope-intercept form, $y = (1 - p_i^0 x)/p_i^1$).

For example, the *Stoudemire* point $(0.77, 0.77)$ in Figure 3, is transformed into the *dual line*, $y = 1.30 - x$. Equivalently, this transformation can be viewed as finding the line that is orthogonal to the vector $\langle 0.77, 0.77\rangle$ and passes through the y-axis at $1/0.77$ and through the x-axis at $1/0.77$. The three points *Durant*, *Stoudemire*, and *Randolph* are all depicted in dual space in Figure 2 as the lines (i, c), (h, b), and (g, a), respectively.[4]

An important property of this transform is that if two points p_i and p_j have the same score on a weight vector $\mathbf{w} = \langle x, y\rangle$, then l_i and l_j will intersect at the point (x, y). For example, in Table 2 it was shown that *Stoudemire* and *Randolph* have the same score, 0.77, on weight vector $\mathbf{w}_1 = \langle 0.72, 0.28\rangle$. In Figure 2, the two corresponding dual lines intersect at the point $f = (0.72, 0.28)$.

[3]When D is clear from the context we will use $\mathcal{L}$ to denote $\mathcal{L}(D)$ for notational convenience.

[4]Technically, these lines stretch infinitely in both directions, but algorithmically we are only interested in the positive quadrant.

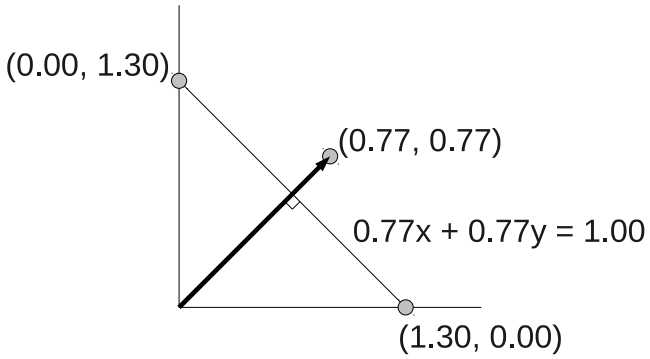


Figure 3: An illustration of the duality transform. The *Stoudemire* point at $(0.77, 0.77)$ is transformed into an orthogonal line with equation $0.77x + 0.77y = 1.0$.

In dual space, the analog of a set of points is a *convex chain* that begins on the positive x-axis and ends on the postive y-axis:

DEFINITION 4.1 (CONVEX CHAIN). *A* convex chain *of size r is a sequence of line segments, $(s_0, \ldots, s_{r-1})$, where the slope of any segment s_i is negative and less than that of s_{i+1}, and any two consecutive line segments have a common endpoint. We call the endpoints of the line segments the* joints *of the convex chain.*

For example, in Figure 2, $((h, f), (f, e))$ and $((i, d), (d, b))$ are both convex chains, although we are only interested in the second example because the first does not end on the y-axis. In contrast, $((g, e), (e, d), (d, b))$ and $((h, f), (e, a))$ are not convex chains because the first is *not convex* and the second is *not a chain*.

The other important concept for this section, borrowed from [6], is the *top-k rank contour*, which models the k'th best score for any weight vector:

DEFINITION 4.2 (TOP-k RANK CONTOUR [6]). *Given a set of lines $\mathcal{L}$, the* top-k rank contour, *denoted $\mathcal{C}_k$, is the set of points from each $l_i \in \mathcal{L}$ such that for any point $p \in \mathcal{C}_k$, exactly $k-1$ lines in $\mathcal{L}$ cross the segment $[O, p]$ (ties withstanding).*

A top-k rank contour is a chain, but is not necessarily convex. For example the top-1 contour in Figure 2 is the dotted chain, $((i, e), (e, a))$. The second best scores, instead, are given by the top-2 contour, the non-convex chain $((h, f), (f, e), (e, d), (d, b))$. The top-$k$ rank contour of n lines can be found in $\mathcal{O}(n \lg n)$ time [6].

We assess how "close" a convex chain is to the top-k rank contour, because this is the same as asking how well the corresponding set of points approximates the top-k choices in D. We explain this by the example of Figure 2. First, consider the weight vector, $\mathbf{w}_0 = \langle 1.0, 0.0\rangle$. In this direction, the *Stoudemire* line, (h, b), has a promixity to the contour of $|(i, h)|/|(O, h)| = 0.23$. On the other hand, the *Durant* line, (i, c), has a proximity of $|(i, i)|/|(O, i)| = 0.00$.

The proximity of the convex chain to the contour is the maximum such ratio, considering all directions. Although the *Durant* line had a distance ratio of 0.00 in the direction of $\mathbf{w}_0 = \langle 1.0, 0.0\rangle$, it has a *maximum* distance ratio of $|(a, c)|/|(O, c)| = 0.34$ in the direction of $\mathbf{w}_4 = \langle 0.0, 1.0\rangle$; so, this is the *distance* of the chain $((i, c))$ from the top-1 rank contour.

For contrast, the distance of the *Stoudemire* line was already maximized at $\mathbf{w}_0 = \langle 1.0, 0.0\rangle$; so, the chain $((h, b))$ is closer to the top-1 rank contour than $((i, c))$ and represents a better (in fact, optimal) 1RMS solution of size 1.

4.2 An algorithm for two dimensions

In this section, we present Algorithm 1 to find a convex chain of size r within $\mathcal{L}(D)$ whose maximum distance ratio from the top-k rank contour is minimized. This gives the kRMS solution in primal space. The algorithm is illustrated in Figure 4, continuing the example from Figure 2. We begin with a high-level description of the algorithm in Section 4.2.1, then give more detail in Section 4.2.2 by describing the data structure transitions.

4.2.1 Algorithm Description

Algorithm 1 Two-dimensional kRMS algorithm

1: **Input**: D; r
2: **Output**: $R \subseteq D$, with $|R| = r$ and minimum k-regret ratio
3: Compute $\mathcal{C}_k$.
4: Transform set of points D into set of lines $\mathcal{L}$.
5: Sort $\mathcal{L}$ by ascending x-intercept.
6: Initialize $\mathcal{Q}$ with intersection points of all $l_i, l_{i+1} \in \mathcal{L}$, sorted by angle from positive x-axis.
7: Initialize $\mathcal{P}$ as $n \times r$ matrix, cell (i, j) has path (l_i) and cost equal to the regret ratio of l_i on $\mathbf{w} = \langle 1.0, 0.0 \rangle$.
8: **while** $\mathcal{Q}$ is not empty **do**
9: Pop top element q off $\mathcal{Q}$.
10: Update data structures $\mathcal{L}, \mathcal{Q}, \mathcal{P}$ as per Section 4.2.2.
11: **end while**
12: Update each cell $(i, r-1)$ of $\mathcal{P}$ with cost at positive y-axis.
13: Find cell $(i, r-1)$ in $\mathcal{P}$ with lowest cost.
14: Init empty set R.
15: **for all** j on convex chain in cell $(i, r-1)$ **do**
16: Add p_j to R
17: **end for**
18: RETURN R

At a high level, the algorithm is a radial plane sweep and dynamic programming algorithm, which traces out and evaluates the possible convex chains in $\mathcal{L}$. A sweep line L rotates from the positive x-axis to the positive y-axis, stopping at any intersection points (as seen left to right in the five subsequent columns of sub-figures in Figure 4). While L rotates, we maintain a list of the best seen solutions so far. At each intersection point, we check whether to update our list of best seen solutions, then continue on. When L reaches the positive y-axis (the far right in Figure 4), the best of the best seen solutions is the final answer.

The plane sweep is achieved with two data structures: a sorted list of $\mathcal{L}$ and a priority queue $\mathcal{Q}$ containing *some* of the unprocessed joints of *some* of the convex chains. Meanwhile, of all convex chains encountered up to L, the best seen solutions are maintained in an $n \times r$ matrix $\mathcal{P}$, the dynamic programming data structure.

The set of lines $\mathcal{L}$ is always sorted with respect to the distance from the origin in the direction of L. As L rotates, $\mathcal{L}$ needs to be updated because the sort order changes. For example, just prior to L in Figure 4k, the order of lines is *Durant*, *Stoudemire*, *Randolph*, but just afterwards, the order of *Stoudemire* and *Randolph* flips.

The priority queue $\mathcal{Q}$ contains, sorted by increasing angle, intersection points that have been 'discovered.' An intersection point of lines l_i, l_j is discovered exactly when l_i and l_j are immediate neighbours in $\mathcal{L}$ *and* their intersection point lies between L and the positive y-axis. At the moment in Figure 4k, the second intersection point is discovered, because that is the first time that the *Randolph* and *Durant* lines become neighbours. This discovery process keeps the size of $\mathcal{Q}$ smaller and avoids exhaustively searching all $n(n-1)/2$ intersection points.

Finally, the matrix $\mathcal{P}$ contains a cell (i, j) for the best convex chain found up to L that ends in a segment of l_i and contains no more than j joints. Stored in each cell is the convex chain itself *and* the maximum distance ratio of that convex chain from the top-k rank contour, $\mathcal{C}_k$. In Figure 4, the bottom row corresponds to where l_i is Randolph. In Figure 4k, cell $(i, 0)$ will contain the cost value 0.32 and the chain $((g, f))$; cell $(i, 1)$ will contain the cost value 0.23 and the chain $((h, f), (f, f))$.

4.2.2 Data structure transitions

The primary processing in the algorithm is in the data structure transitions, which we describe now. Let L be at an arbitrary intersection point of lines l_i and l_j, denoted $p_{i,j}$. For simplicity of discussion, assume that only two lines intersect at any given point; it is straight-forward to handle lines not in general position.

$\mathcal{L}$

Because the lines are intersecting, we know they are immediately adjacent in $\mathcal{L}$. We swap l_i and l_j in $\mathcal{L}$ to reflect the fact that immediately after $p_{i,j}$, they will have opposite order as beforehand. In Figure 4, this happens once for each column.

$\mathcal{Q}$

Immediately after $p_{i,j}$, lines l_i and l_j have been swapped in $\mathcal{L}$. So, potentially, two new intersection points are discovered: viz., l_i and its new neighbour (should one exist) and l_j and its new neighbour (again, should one exist). Both these intersection points are added to the appropriate place in $\mathcal{Q}$, provided that they are between L and the positive y-axis. The point $p_{i,j}$ is removed $\mathcal{Q}$. In the second column of Figure 4, point e is discovered and inserted. The other intersection points, d and f, had been discovered at initialisation (the first column).

$\mathcal{P}$

Of the four paths, $(l_i, l_i), (l_i, l_j), (l_j, l_i), (l_j, l_j)$, through $p_{i,j}$, only three are valid. For example, consider point d in Figure 4h. The turn from *Stoudemire* onto *Durant* is concave, so the resultant chain is invalid. However, the paths straight through d, as well as the path turning from *Durant* onto *Stoudemire* produce valid convex chains.

To update the cost, it depends on the path chosen through $p_{i,j}$. For a line transitting through $p_{i,j}$, such as (l_i, l_i), the convex chain does not change, but the cost is updated to the larger of what the value was before and the distance ratio of $p_{i,j}$ relative to $\mathcal{C}_k$ in the direction of L. For the convex chain that turns, (l_i, l_j), the cost in cell (j, h) depends on the best route to get to $p_{i,j}$. Specifically, the best convex chain of size h to $p_{i,j}$ is either: 1) the chain incoming on l_j if the cost in cell (j, h) is smaller; or 2) the chain incoming on l_i if the cost in cell $(i, h-1)$ is smaller. Call the smaller cost mc. The cost for cell (j, h) then becomes the larger of mc and the distance ratio of $p_{i,j}$ in the direction of L.

For rows i and j, each of the $2r$ cells is updated in this manner. If $p_{i,j}$ happens to be a vertex of $\mathcal{C}_k$ (such as in the middle column of Figure 4), then every cell of every row is updated in this manner, not just those of lines l_i and l_j (as in the second and fourth columns of Figure 4).

As a last quick note, we now show Lemma 4.1, which bounds the running time and space requirements for Algorithm 1.

LEMMA 4.1. *Algorithm 1 finds a k-regret minimizing set of size r for $d = 2$ in $\mathcal{O}(rn^2)$ time and $\mathcal{O}(n^2)$ space.*

PROOF SKETCH. The space comes from storing the dynamic programming matrix, $\mathcal{P}$, which is the largest of the three data structures. The running time is dominated either by updating $\mathcal{O}(r)$ cells

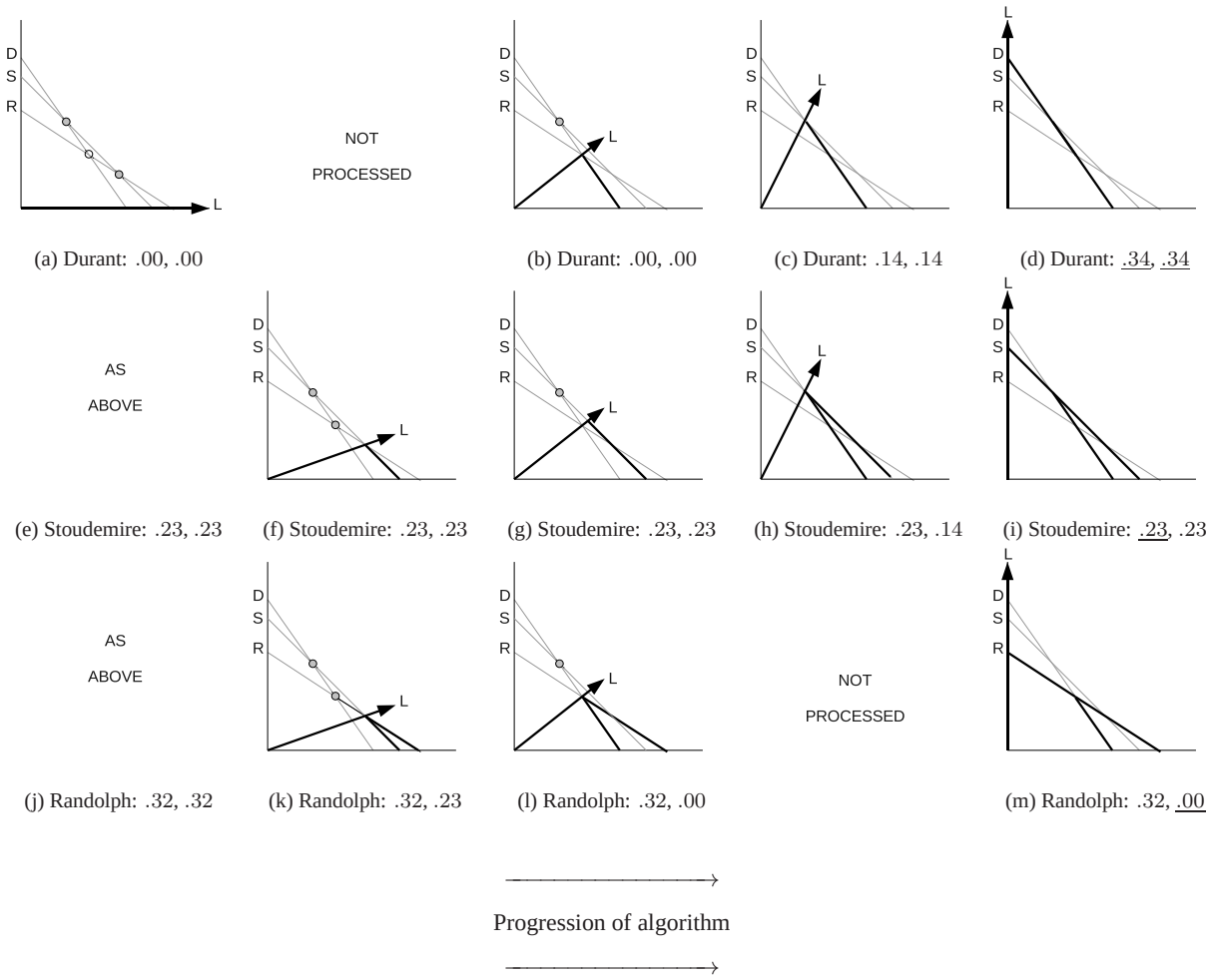


Figure 4: An illustration of Algorithm 1 on the lines in Figure 2. There are five *events* in this example, the initialisation, three intersection points, and termination, each depicted in a column and progressing chronologically towards the right. Each row from top to bottom shows depicts chains ending on the *Durant*, *Stoudemire*, and *Randolph* lines, respectively. For each figure, the best interim solutions, one with 0 joints and one with 1 joint, are shown in bold (although in some cases, like (f), these are identical). Below each figure is shown the *cost* for, first, the 0-joint solution and, second, the 1-joint solution. In the final column (at termination), the best costs are underlined.

of $\mathcal{P}$ for each of the (up to) $n(n-1)/2$ intersection points or by updating all nr cells of $\mathcal{P}$ for the up to $n-1$ vertices of $\mathcal{C}_k$.

5. A RANDOMIZED ALGORITHM FOR GENERAL DIMENSION

Having shown the hardness of regret minimization in Section 3, we know that one cannot aspire towards a fast, optimal algorithm for kRMS in arbitrary dimension. However, we can still aim for a fast algorithm to find sets with *low* k-regret ratio; in this section, we describe a randomized, greedy kRMS algorithm that achieves this goal.

After first recalling the 1RMS algorithm of Nanongkai et al. [18], we extend it for 2RMS in Section 5.1. Then, we show how by introducing random partitioning with repeated trials, we can produce an effective kRMS algorithm for arbitrary k (Section 5.2).

Algorithm 2 is a simple yet effective greedy algorithm that expands an interim solution R point-by-point with the local optimum. For each interim solution R, Linear Program 1 below is run on every point $p \in D$ to find the one that is responsible for the current maximum regret ratio. In the terminology of the previous section, each iteration from $1 < |R| \leq r$ finds the point on $\mathcal{C}_1$ farthest from R and adds that to the interim solution.

Linear Program 1 below finds, given an interim solution $R \subseteq D$ and a point $p \in D$, the weight vector $\mathbf{w}$ that maximizes the 1-regret ratio of R relative to $R \bigcup \{p\}$. The 1-regret ratio is proportional to x, which is upper-bounded in constraint (2).

LINEAR PROGRAM 1.

$$\text{maximize } x \text{ s.t.} \tag{1}$$
$$\mathbf{p} \cdot \mathbf{w} - \mathbf{p}' \cdot \mathbf{w} \geq x \quad \forall \mathbf{p}' \in R \tag{2}$$
$$w_i \geq 0 \quad \forall 0 \leq i < d \tag{3}$$
$$\mathbf{p} \cdot \mathbf{w} = 1 \tag{4}$$
$$x \geq 0 \tag{5}$$

To understand the algorithm, consider an example using the points in Table 2. Initally, R is set to $\{\text{Durant}\}$, since he maximizes the first attribute. We first execute Linear Program 1 with respect to

Algorithm 2 Greedy algorithm to compute 1RMS [18]

1: **Input**: D; r
2: **Output**: $R \subseteq D$, with $|R| = r$ and low 1-regret ratio
3: Let $R = \{p_i \in D\}$, where p_i is a point in D with highest p_i^0.
4: **while** $|R| < r$ **do**
5: Let q be first $p \in D$.
6: **for all** $p \in D \setminus R$ **do**
7: Let max_regret(p) be result of Linear Program 1 with input p, R.
8: **if** max_regret(p)>max_regret(q) **then**
9: Let q be p.
10: **end if**
11: **end for**
12: Let $R = R \bigcup \{q\}$.
13: **end while**
14: RETURN R

Stoudemire and find the positive, unit weight vector $\widetilde{\mathbf{w}}$ that maximizes $(0.77 - 1.00)w_0 + (0.77 - 0.66)w_1$: it is at $\langle 0, 1\rangle$ and produces a difference in scores of $x = 0.11$. The expression for Randolph, $(0.68 - 1.00)w_0 + (1.00 - 0.66)w_1$, is also maximized at $\langle 0, 1\rangle$, but with $x = 0.34$. For James, on the other hand, there is no feasible region, because he is dominated by Durant. So, the iteration of the algorithm concludes by greedily adding Randolph to $R = \{\text{Durant}\}$, since he produced the highest score of $x = 0.34$.

5.1 Extending 1RMS to 2RMS

To go from 1RMS to 2RMS, one needs to find points not top-ranked but 2-ranked and measure regret ratio with respect to them. Our 2RMS algorithm is largely unchanged from Algorithm 2 except for invoking Linear Program 2 instead. Linear Program 2 finds a weight vector $\mathbf{w}$ to maximize x, the regret ratio, and also determines with y in constraint (8) the amount by which the best point in $D \setminus R \setminus \{p\}$ outscores p on $\mathbf{w}$. So, if $y > 0$, then p is at best 2-ranked and an eligible candidate to add to R. On Line (8) of Algorithm 2, we specify an additional clause, that $y \geq 0$, to rule out 1-ranked points.

LINEAR PROGRAM 2.

$$\text{maximize } x - \varepsilon y \text{ s.t.} \tag{6}$$
$$\mathbf{p} \cdot \mathbf{w} - \mathbf{p}' \cdot \mathbf{w} \geq x \qquad \forall \mathbf{p}' \in R \tag{7}$$
$$\mathbf{p}'' \cdot \mathbf{w} - \mathbf{p} \cdot \mathbf{w} \leq y \quad \forall \mathbf{p}'' \in D \setminus R \setminus \{p\} \tag{8}$$
$$w_i \geq 0 \qquad \forall 0 \leq i < d \tag{9}$$
$$\mathbf{p} \cdot \mathbf{w} = 1 \tag{10}$$
$$x \geq 0 \tag{11}$$
$$y \geq -\varepsilon \tag{12}$$

Constraint (8) is of an existential nature; so, there may be more than one point that outscores p in the direction of $\mathbf{w}$, indicating that p is *not* 2-ranked. But if some other point p''' also outscores p on $\mathbf{w}$, then either p'' or p''' will better maximize x than does p and be chosen instead. Note that we include y in the objective function to ensure it takes the minimum valid value, making the 1-ranked case distinctive. The really small, positive real, ε, dampens the effect of y—we foremost want x to be maximized, even when it is paired with a large y.

To return to the example from before, consider again when $R = \{\text{Durant}\}$ and we evaluate $p =$ Stoudemire. As before, we maximize $(0.77 - 1.00)w_0 + (0.77 - 0.66)w_1$, the "distance" from Stoudemire to the current regret minimizing set, R. But now, also, $\widetilde{\mathbf{w}}$ must have a non-negative solution to $(0.68 - 0.77)w_0 + (1.00 - 0.77)w_1$ *or* to $(0.91 - 0.77)w_0 + (0.58 - 0.77)w_1$; otherwise, Stoudemire is the top-ranked point on the given query direction and we are computing the regret of $R \bigcup \{p\}$ relative to the top-ranked rather than second-ranked point. Stoudemire maximizes $x = 0.11$ at $\langle 0, 1\rangle$, this time also minimzing $y = 0$. The feasible region of Randolph, however, does not include $\langle 0, 1\rangle$, because both $(0.77 - 0.68)w_0 + (0.77 - 1.00)w_1$ and $(0.91 - 0.68)w_0 + (0.58 - 1.00)w_1$ are negative at $\langle 0, 1\rangle$. In fact, the feasible region is empty, because Randolph is never simultaneously at best 2^{nd}-ranked and higher ranked than Durant. Instead, Stoudemire is selected to augment R.

5.2 Extending 2RMS to kRMS

Algorithm 3 Greedy algorithm to compute kRMS

1: **Input**: D; r; k; T
2: **Output**: $R \subseteq D$, with $|R| = r$ and low k-regret ratio
3: Let $R = \{p_i \in D\}$, where p_i is point in D with highest p_i^0.
4: **while** $|R| < r$ **do**
5: Let q be first $p \in D$ and $\widetilde{\mathbf{w}} = \langle 0, \ldots, 0\rangle$.
6: **for all** $p \in D \setminus R$ **do**
7: **for all** i from 1 to T **do**
8: Randomly partition $D \setminus R \setminus \{p\}$ into $D_0, \ldots, D_{k-2}$
9: Let max_regret(p) be result of Linear Program 3 with input $p, R, D_0, \ldots, D_{k-2}$.
10: **if** max_regret(p) has all $x_j > 0$ **then**
11: **if** max_regret(p)>max_regret(q) **then**
12: Let q be p and $\widetilde{\mathbf{w}}$ be $\mathbf{w}$ from Linear Program 3.
13: **end if**
14: Break inner loop and go to next point.
15: **end if**
16: **end for**
17: **end for**
18: Let $S = \{q\}$ and $\mathbf{w}$
19: **for all** $p \in D \setminus R$ **do**
20: **if** $p \cdot \widetilde{\mathbf{w}} \geq q \cdot \widetilde{\mathbf{w}}$ **then**
21: Let $S = S \bigcup \{p\}$.
22: **end if**
23: **end for**
24: Let s be 'best' member of S with heuristic of choice.
25: Let $R = R \bigcup \{s\}$.
26: **end while**
27: RETURN R

To solve the more general kRMS problem, we make use of Proposition 5.1 and randomness. The idea is that we decompose each iteration of the kRMS problem into a set of 2RMS problems and optimize for a common solution.

PROPOSITION 5.1. *If $p = D^{(k,\mathbf{w})}$, then there exists a partitioning of D into $D_0, \ldots, D_{k-2}$ such that $\forall D_i, p = D_i^{(2,\mathbf{w})}$.*

To rephrase Proposition 5.1, if p is k-ranked on D with respect to $\mathbf{w}$, then we can split D into $k - 1$ partitions such that p will be 2-ranked on every one with respect to the same weight vector, $\mathbf{w}$. The key is that the partitions must each contain exactly one of the points higher ranked than p.

Without knowing apriori the weights of $\mathbf{w}$, it is challenging (and, we posit, an interesting open research direction) to construct such a partitioning. A random partitioning, however, may successfully separate the higher-ranked points into disjoint partitions and allow us to find $\mathbf{w}$ with Linear Program 3. Of course, a random partitioning may very well *not* produce such a separation, but then Linear

ID	Name	Source	n	d	|Sky|
AI	All-inclusives	yvrdeals.com	425	2	10
BB	Basketball	databasebasketball.com	21961	5	200
EN	El Nino	archive.ics.uci.edu/ml/datasets/El+Nino	178080	5	1183
WE	Weather	cru.uea.ac.uk/cru/data/hrg/tmc/	566262	13	16433
HH	Household	usa.ipums.org/usa/	862967	6	69
AT	Air Traffic	kt.ijs.si/elena_ikonomovska/data	115069017	4	87

Table 5: Statistics for the six datasets used in these experiments.

Program 3 will report that p is not k-ranked and we can keep trying new random partitionings.

LINEAR PROGRAM 3.

$$\text{maximize } x - \varepsilon \sum x_j \text{ s.t.} \tag{13}$$

$$\mathbf{p}\cdot\mathbf{w} - \mathbf{p}'\cdot\mathbf{w} \geq x \qquad \forall \mathbf{p}' \in R \tag{14}$$

$$\mathbf{p}''\cdot\mathbf{w} - \mathbf{p}\cdot\mathbf{w} \leq x_j \quad \forall \mathbf{p}'' \in D_j, 0 \leq j \leq k-2 \tag{15}$$

$$w_i \geq 0 \qquad 0 \leq i < d \tag{16}$$

$$\mathbf{p}\cdot\mathbf{w} = 1 \tag{17}$$

$$x \geq 0 \tag{18}$$

$$x_j \geq -\varepsilon \qquad 0 \leq j \leq k-2 \tag{19}$$

So, we have Algorithm 3 to solve the kRMS problem. It is similar to Algorithm 2, except Linear Program 3 is executed several times for each point, each after randomly partitioning $D \setminus R \setminus \{p\}$. If Linear Program 3 sets all $x_j > 0$, its solution is optimal; if it does not, either the partitioning was unlucky, R is still quite poor, or p cannot contribute to improving the interim R. So, we try another hopefully luckier partitioning until after a maximum number of trials that is dependent on k. There is a probability of .1 that 8 trials at $k = 3$ are all unlucky, for example. With Proposition 5.2, we can bound the number of partitioning trials with high probability; although, as will be evidenced in Section 6, 'unluckiness' is not necessarily so costly, anyway.

PROPOSITION 5.2. *If one repeatedly partitions into m parts a dataset D with at least m points of interest, the probability of not obtaining a repetition in which each partition contains a point of interest after $\frac{3.16m^{2m}}{m!m!} - \frac{2.16m^m}{m!}$ trials is $\leq .1$.*

PROOF SKETCH. The probability comes from the Chebyshev Inequality, given that the repeated partitioning is a Bernoulli Process with chance of success $\geq \frac{m!}{m^m}$.

To finish the running example, we use all the points in Table 1 (but normalized as before and now in all four dimensions). Let $R = \{\text{Durant}\}$ and $k = 3$ and consider the computation for the point $p =$ James. First, we partition the remaining points, say into $\{\{\text{Anthony}, \text{James}, \text{Nowitzki}\}, \{\text{Wade}, \text{Randolph}, \text{Bryant}\}\}$. Next, we find the vector $\widetilde{\mathbf{w}}$ that simultaneously solves 2RMS on each partition. This particular partitioning has no such vector, because James is not $\geq 2^{\text{nd}}$-ranked on the first partition for any vector on which he outranks Durant. So, the points are randomly re-partitioned. Eventually, some random partitioning will separate Wade and Bryant, at which point James will be ranked third on the weight vector $\langle 0, 0, 0.75, 0.25\rangle$, the vector for which he maximally outranks Durant while still being ranked at best third. However, ultimately, James will not be added to R on this iteration, because Anthony produces a larger jump in solution quality on the weight vector $\langle 0, 0, 0, 1\rangle$, one for which he is exactly third ranked.

Heuristics for kRMS. On a final note, once a maximal $\widetilde{\mathbf{w}}$ is discovered, there are k points in D that could be selected, for each has $k\text{-regratio}(p, \widetilde{\mathbf{w}}) = 0$. Of these, we choose the point with the largest sum of coordinates (line 24 of Algorithm 3). So, it can be that distinct points produce the same final solution.

6. EXPERIMENTAL EVALUATION

In this section, we empirically compare 1-regret ratio to k-regret ratio for values of $k \leq 4$. To do this, we implement the algorithms of Section 5 in C, using the MOSEK linear program (LP) solver,[5] and then look at performance over six real datasets with respect to *solution quality* and *execution time*. We also conduct an exploration of the impact of randomization on the solution quality.

6.1 Datasets

We run experiments against six real datasets, summarized in Table 5, which range up to roughly $100,000,000$ points and up to 13 dimensions. The all-inclusives (AI) dataset is the archetypal skyline dataset, trading off ratings and prices of hotels. The basketball (BB) dataset contains points for each player/team/season combination for *rebounds*, *assists*, *blocks*, *fouls*, and *points scored*. The El Nino (EN) dataset consists of oceanographic readings like *surface temperature* and *wind speed*, taken at buoys placed in the Pacific Ocean. And the household (HH) dataset contains US census data for expenses such as *electricity* and *mortgage*.

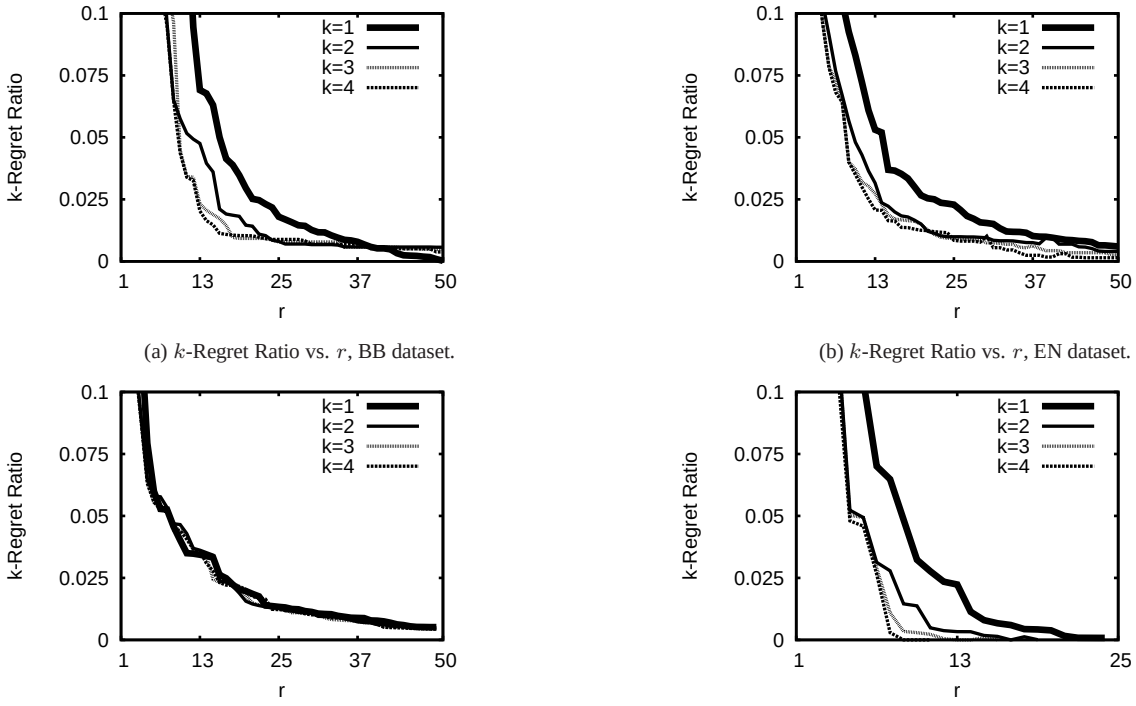
The two largest datasets are the Weather (WE) dataset, which consists of average monthly precipitation totals and elevation at over half a million sensor locations, and the Air Traffic (AT) dataset, which gives distances and arrival delays for over one hundred million American flights from 1987 to 2008.

For all six datasets, the attributes have been normalized to the range [0,1] by subtracting the smallest value and then dividing by the range. Missing values have been replaced with the lowest value found in the dataset. Non-skyline points have been pruned, because they will never form part of any solution.

6.2 Experiment descriptions

Our experiments compare performance in terms of quality of solution and execution time. Towards the former, we measure the maximum k-regret ratio after $1 \leq r \leq 50$ tuples have been reported. The purpose of this question is to determine how much better a subset of size r can approximate the k'th best tuples of an entire dataset than it can the top-1 tuples. We evaluate execution time by the average wall time (in milliseconds) of the LP subroutine, which is the primary algorithmic difference for distinct values of k. This is a more meaningful metric than the overall wall time, because the difference between the cost of running the LP n times and of running the entire algorithm depends primarily on T, the number of partitioning trials, which is highly tunable.

[5] http://www.mosek.com



(a) k-Regret Ratio vs. r, BB dataset.

(b) k-Regret Ratio vs. r, EN dataset.

(c) k-Regret Ratio vs. r, WE dataset.

(d) k-Regret Ratio vs. r, AT dataset.

Figure 5: *k-Regret ratio vs. r.* A point on the plot shows the k-regret ratio (y-axis) achieved by a k-regret minimizing set of size $|R|$ (x-axis). Each series gives a value of k.

We also run experiments to assess the effect of using randomness. Under a perfect partitioning, the LP will find a weight vector $\mathbf{w}$ with maximal k-regret ratio x. However, due to the use of randomness, it is is possible that T repeated 'unlucky' trials leads to missing this ideal solution, or even that mT repeated 'unlucky' trials leads to missing the m best LP solutions. We ask what percentage of the dataset must be missed by 'unlucky' randomness before losing 2.5% of the value of x. This is a measure of how costly unluckiness can be on the quality of the solution.

We run all these experiments on a machine with two 800MHz cores and 1GB RAM, running Ubuntu 13.04. We set the number of random trials, T, as per Proposition 5.2 (i.e., high enough to expect a 90% success rate). Thus for $k = 2$, $T = 1$; for $k = 3$, $T = 8$; and for $k = 4$, $T = 54$.

6.3 Discussion

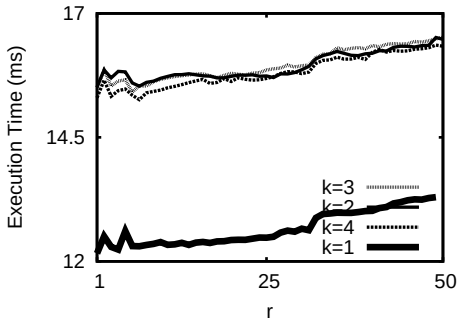
We first discuss the results of the solution quality experiments in Figure 5. On the AI dataset (not shown), for which we can compute optimal solutions, we find the following sets returned for manual comparison. At $k = 1$, $R = \{$(4.7,\$1367/pp),(3.4,\$847/pp),(4.6, \$1270/pp)$\}$; at $k = 2$, the third element is instead (3.9, \$1005/pp); and at $k \geq 3$, it requires only two resorts to satisfy every user. So, the sets at $k > 1$ appear more diverse at constrained sizes.

That observation is mirrored in the plots of 5a, 5b, and 5d, which plot k-regret ratio as a function of output size. We set the x-axis range up to 50 which is $\geq 25\%$ of the skyline of three datasets and is sufficient to reduce the k-regret ratio to less than half a percent. The exception is 5d, where the k-regret ratio can be reduced to 0 with 10 to 24 points, depending on the value of k. Also, we do not show k-regret ratios on the y-axis above 10%, because 90% is a reasonable minimum expectation for accuracy. On the WE dataset (5c), relative to the others, we see that even at $k = 1$, a low k-regret ratio is achievable with only d output points, indicating that there are fewer interesting query weights for this dataset. Across the other three datasets (and the two not shown), we see a big jump from $k = 1$ to $k = 2$, sometimes another jump from $k = 2$ to $k = 3$ (5a and 5d), and comparability between $k = 3$ and $k = 4$.

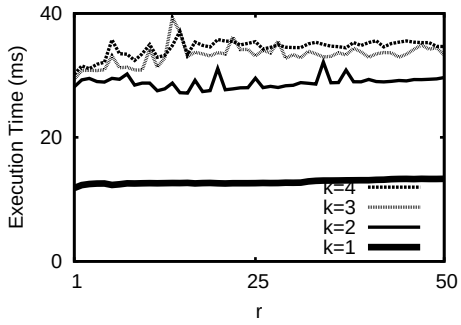
The initial differences can be quite substantial. On the BB dataset, at $k = 3$ and $k = 4$, we achieve a k-regret ratio of .02 with a set R about half the size as is required for $k = 1$. In contrast, on the WE dataset, the performance is equal, on account of $k = 1$ performing much stronger than it does on other datasets. The conclusion is that one can consistently achieve excellent dataset approximations by increasing $k > 1$, often substantially better than at $k = 1$, and that $k = 2$ and $k = 3$ produce the greatest relative dividends.

The plots in Fig. 6 show, as a function of output size, the average execution time for a single run of the pertinent linear program (LP). We vary the y-axis, time, to the range of the specific plot so that each series is easier to read. The x-axis is again bounded by $r \leq 50$, except for 6d. We observe an anticipated jump in execution time from $k = 1$ to $k = 2$, given the additional constraints added to Linear Program 2. However, upwards of $k = 2$, the cost does not increase much and the shape of the curves remains consistent.

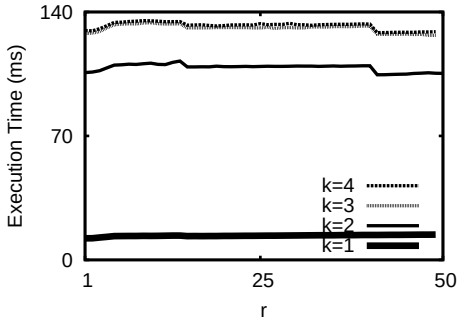
Finally, the results on randomness are in Figure 7.The plots show, as a function of output size, what percentage of the top 100 sub-optimal points produce a score close to optimal—characterizing the cost of selecting a sub-optimal point. The y-axis runs to its maximal possible value, 100%, in all plots and starts at the highest value that still clearly shows all series for the specific plot. The x-axis runs to 50, except on the HH dataset (7c), where a k-regret ratio of 0 is obtained by $r = 20$ for $k = 1$, and even earlier for the other three values of k. In all plots, we observe percentages that



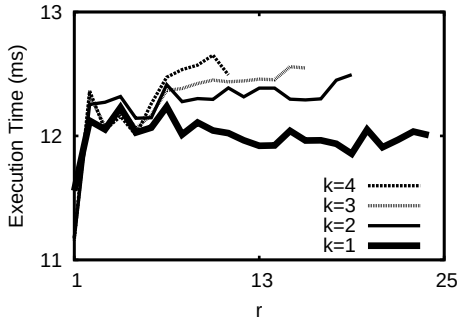
(a) Execution time vs. r, BB dataset.



(b) Execution time vs. r, EN dataset.



(c) Execution time vs. r, WE dataset.



(d) Execution time vs. r, AT dataset.

Figure 6: *Execution time vs.* r. The wall time to run one instance of Linear Program 1 (in the case of 1RMS) or Linear Program 3 (in the case of kRMS, $k > 1$), averaged over 1000 trials. The x-axis gives $|R|$ and the y-axis gives milliseconds. Each series gives a value of k.

are consistently high for $k > 2$. For example, at $k = 3$, for all values r, over 70% of the top 100 points produce a score within 2.5% of optimal. Therefore, there is a very strong resilience to "bad luck": $70T$ trials must all fail in order to lose 2.5%. The conclusion that we draw is that one can confidently lower T (and thus decrease running time) without incurring much loss in the quality of the solution.

7. RELATED WORK

The idea to represent an entire dataset by a few representative points for multi-criteria decision making has drawn much attention in the past decade, since the introduce of the Skyline operator by Börzsönyi et al. [2]. However, the susceptibility of the skyline operator to the curse of dimensionality is well-known. Chan et al. [4] made a compelling case for this, demonstrating that on the NBA basketball dataset (as it was at the time), more than 1 in 20 tuples appear in the skyline in high dimensions. Consequently, there have been numerous efforts to derive a representative subset of smaller size (e.g., [3, 13, 23, 24]), especially one that presents very distinct tuples (e.g., [8, 19]) or has a fixed size (e.g., [14, 15, 21]).

Regret minimizing sets are relatively new in the lineage of these efforts. When introduced by Nanongkai et al. [18], the emphasis was on proving that the maximum regret ratio is bounded by:

$$\frac{d-1}{(c-d+1)^{d-1}+d-1}.$$

Naturally, this bound holds for the generalisation introduced in this paper, since $k\text{-regratio}(R, \mathbf{w}) \leq (k-1)\text{-regratio}(R, \mathbf{w})$. As far as we know, this paper is the first to address computational questions around k-regret minimizing sets, certainly for $k > 1$.

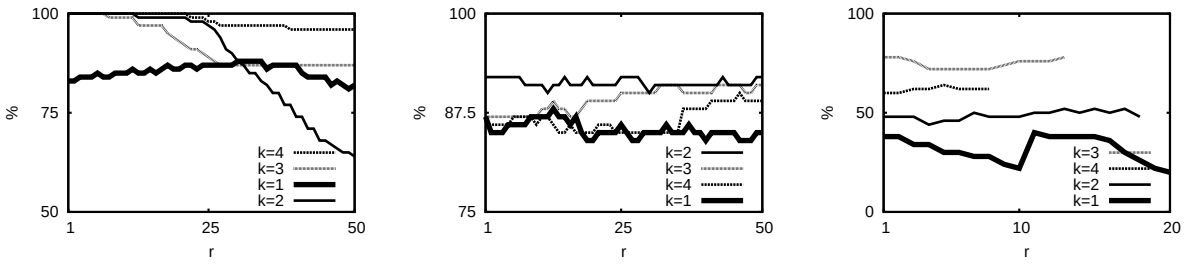
Regret minimizing sets presuppose that linear top-k queries are of interest, a class of queries that has been well studied and has been surveyed quite thoroughly by Ilyas et al. [10]. The use here of duality is fairly common (e.g., [7, 12, 20]) as is the emphasis on (layers of) lower envelopes (e.g., [5, 25]). Transforming points into dual space in two dimensions often leads to the employment of plane sweep algorithms [9] and the availability of many results on arrangements of lines. For example, Agarwal et al. [1] give bounds on the the number of edges and vertices that can exist in a chain (such as $\mathcal{C}_k$) through an arrangement. The dual-space top-k rank contours of Chester et al. [6], which were proposed to answer monochromatic reverse top-k queries [22], are central to our two dimensional algorithm. It is an interesting question whether duality can help in higher dimensions and also whether there exists a strong connection between reverse top-k queries and k-regret minimization as the application of these results may imply.

Lastly, *anytime* skyline algorithms can be halted mid-execution and output a non-optimal solution [16]. Regret minimizing sets are well suited to these interactive scenarios [17]; so, it is reasonable to believe that k-regret minimizing sets may be suitable as well.

8. CONCLUSIONS

The 1-regret minimizing set is a nice alternative to the skyline as a succinct representation of a dataset, but suffers from rigidly fitting the top-1 for every query. We generalised the concept to that of the *k-regret minimizing set*, which represents a dataset not by how closely it approximates every users' top-1 choice, but their top-k choice. Doing so permits simultaneously achieving a lower k-regret ratio while making the representative subsets much smaller.

In the special case of $d = 2$, we give an efficient, exact algorithm



(a) *Score loss* plot, BB dataset. (b) *Score loss* plot, EN dataset. (c) *Score loss* plot, HH dataset.

Figure 7: Q4) *Score loss*. For each value r (the x-axis), the y-axis gives the percentage of tuples producing scores on their own maximal weight vector within 2.5% of the k'th best score on the optimal maximal weight vector.

based on the dual space insight that the k-regret minimizing set corresponds to the convex chain closest to the top-k rank contour. The algorithm uses dynamic programming and plane sweep to search the space of convex chains. For general dimension, we first resolve a conjecture that computing a 1-regret minimizing set is NP-Hard and extend the result to k-regret minimizing sets. Then, we give a randomized, greedy algorithm based on linear programming to find a subset with *low* k-regret ratio. In comparison to computing subsets with low 1-regret ratio, we show that its solution quality is much stronger.

9. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers and Dr. Ashwin Lall for their helpful suggestions for improving the presentation in this article.

10. REFERENCES

[1] P. K. Agarwal, B. Aronov, and M. Sharir. On levels in arrangements of lines, segments, planes, and triangles. In *Proc. SOCG*, pages 30–38, 1997.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. ICDE*, pages 421–430, 2001.

[3] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *Proc. SIGMOD*, pages 503–514, 2006.

[4] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *Proc. EDBT*, pages 478–495, 2006.

[5] Y.-C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The onion technique: indexing for linear optimization queries. In *Proc. SIGMOD*, pages 391–402, 2000.

[6] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Indexing reverse top-k queries in two dimensions. In *Proc. DASFAA*, pages 201–208, April 2013.

[7] G. Das, D. Gunopulos, N. Koudas, and N. Sarkas. Ad-hoc top-k query answering for data streams. In *PVLDB*, pages 183–194, 2007.

[8] A. Das Sarma, A. Lall, D. Nanongkai, R. J. Lipton, and J. Xu. Representative skylines using threshold-based preference distributions. In *Proc. ICDE*, pages 387–398, 2011.

[9] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. In *Proc. STOC*, pages 389–403, 1986.

[10] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4):11:1–11:58, Oct. 2008.

[11] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[12] J. Lee, H. Cho, and S.-w. Hwang. Efficient dual-resolution layer indexing for top-k queries. In *Proc. ICDE*, pages 1084–095, 2012.

[13] J. Lee, G.-w. You, and S.-w. Hwang. Personalized top-k skyline queries in high-dimensional space. *Information Systems*, 34(1):45–61, Mar. 2009.

[14] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: the k most representative skyline operator. In *Proc. ICDE*, pages 86–95, 2007.

[15] H. Lu, C. S. Jensen, and Z. Zhang. Flexible and efficient resolution of skyline query size constraints. *TKDE*, 23(7):991–1005, 2011.

[16] M. Magnani, I. Assent, and M. L. Mortensen. Anytime skyline query processing for interactive systems. In *Proc. DBRank*, 2012.

[17] D. Nanongkai, A. Lall, A. Das Sarma, and K. Makino. Interactive regret minimization. In *Proc. SIGMOD*, pages 109–120, 2012.

[18] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-minimizing representative databases. *PVLDB*, 3(1):1114–1124, 2010.

[19] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *Proc. ICDE*, pages 892–903, 2009.

[20] P. Tsaparas, N. Koudas, and T. Palpanas. Ranked join indices. In *Proc. ICDE*, pages 277–288, 2003.

[21] A. Vlachou, C. Doulkeridis, and M. Halkidi. Discovering representative skyline points over distributed data. In *Proc. SSDBM*, pages 141–158, 2011.

[22] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Monochromatic and bichromatic reverse top-k queries. *TKDE*, 23(8):1215–1229, 2011.

[23] M. L. Yiu and N. Mamoulis. Multi-dimensional top-k dominating queries. *VLDBJ*, 18(3):695–718, June 2009.

[24] Z. Zhang, X. Guo, H. Lu, A. K. Tung, and N. Wang. Discovering strong skyline points in high dimensional spaces. In *Proc. CIKM*, pages 247–248, 2005.

[25] L. Zou and L. Chen. Pareto-based dominant graph: An efficient indexing structure to answer top-k queries. *TKDE*, 23(5):727–741, 2011.