Hierarchical Algorithms for Regret Minimizing Sets

Sabine Storandt[a,*] **and Carina Truschel**[a,**]

[a]University of Konstanz, Germany
ORCID (Sabine Storandt): https://orcid.org/0000-0001-5411-3834, ORCID (Carina Truschel): https://orcid.org/0009-0009-7582-7209

Abstract. Regret minimizing algorithms tackle the problem of finding representative subsets for large data sets whilst optimizing for multiple objectives. This is done by minimizing the regret of any possible user with respect to the objectives. The regret is measured as the distance between the best-suited point for the user in the subset and the best-suited point for the user in the entire data set. We introduce the novel hierarchical regret minimizing set algorithm (HRMS) using the divide and conquer paradigm to efficiently approximate regret minimizing sets in arbitrary dimension. The HRMS algorithm arranges a pre-existing regret minimizing algorithm in a hierarchy to obtain intermediate regret minimizing sets which are merged until the final solution set is obtained. Further, we introduce the Pareto post-processing technique which replaces dominated points in the output set of any given regret minimizing algorithm. Our experiments on generated data and benchmark data sets in up to 30 dimensions show that the novel HRMS algorithm significantly outperforms the state-of-the-art in terms of the solution quality whilst yielding similar practical running times.

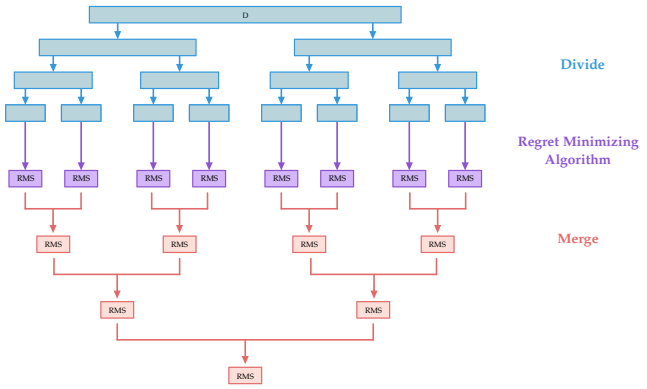


Figure 1. High level schema describing the procedure of the hierarchical regret minimizing algorithm. The input set is divided into smaller subsets for which we obtain regret minimizing sets by applying a known regret minimizing algorithm. The intermediate solution sets are merged in pairs using a filter until the final output set is obtained.

1 Introduction

Computing regret minimizing sets involves finding subsets that represent the entire data set such that the happiness of all possible users is maximized. In other words, we choose a set of points from the data set to minimize the user's regret when deciding for a point from the subset instead of deciding for one from the entire data set.

Consider the following example where Alice is looking to buy a new laptop online. In order to find the laptop which suits her preferences best (e.g. considering the objectives price, screen quality, CPU, and battery life) she would need to spend countless hours of searching the world wide web. Fortunately, an online shop provides a small selection of good laptops where she simply decides for the best-suited laptop according to her preferences. The goal of the online shop is to present for example 10 laptops to Alice such that she does not regret deciding for one laptop among the offered ones even if she would know every single laptop currently on the market. Alice's regret is measured as the ratio of perceived suitability of the laptop from the online shop and the best-suited laptop in the world. Ideally, she finds that both laptops fulfill her requirements and she does not regret deciding for the one from the online shop. To ensure customer satisfaction, the online shop needs to provide a selection of laptops such that every possible user, not just Alice, finds a suitable laptop and none of the users regret buying from the online shop even if they knew all other available options.

In general, given a data set, we define the regret minimizing set as the subset representing the entire data set such that every possible user is sufficiently happy with being limited to the options in the subset during the decision making process. Our proposed algorithm uses the hierarchical schema depicted in Figure 1 to efficiently identify such representative, small subsets in higher dimensions whilst minimizing the regret of any possible user.

1.1 Related Work

Nanongkai et al. [4] introduced the regret minimizing set problem (RMS) in 2010. In the past years, several versions of the problem and different algorithms have been presented. The survey done by Xie et al. [8] provides an extensive overview of various aspects of the regret minimization problem. Firstly, we differentiate between the RMS and the kRMS problem. The former tackles finding representative subsets that minimize the regret of any user towards the best-suited point in the subset and the best-suited point in the entire data set. Whereas the latter relaxes the requirement to minimizing the regret of any user towards the best-suited point in the subset and the k^{th}-best-suited point in the data set. Throughout this work we implicitly use $k = 1$ and simply denote RMS instead of 1RMS.

Chester et al. [3] prove that both the 1RMS and the kRMS problems are NP-hard by constructing a reduction from the SET-COVER

* Corresponding Author. Email: sabine.storandt@uni-konstanz.de
** Corresponding Author. Email: carina.truschel@uni-konstanz.de

problem to the 1- and k-regret minimization problems, respectively. Extending these findings, Agarwal et al. [1] conclude that the kRMS problem is NP-complete for instances with at least $d \leq 3$ dimensions. Hence, the approaches for the RMS problem are subdivided into exact algorithms for two dimensions and heuristic or approximation algorithms for d dimensions.

The Contour algorithm by Chester et al. [3] was the first exact approach for the kRMS problem in two dimensions. Using dynamic programming, the authors find a solution subset of size r by transforming the kRMS problem into a geometric problem in dual space. The algorithm proceeds with finding a convex chain closest to the top-k rank contour. The BiSearch algorithm by Cao et al. [2] is a randomized binary search algorithm based on the same transformation into dual space. Performing binary searches on candidate values of regret ratios that might be optimal leads to an improvement in running time compared to the Contour algorithm.

Heuristic algorithms tackle the NP-hard kRMS problem in arbitrary dimension without offering a theoretical guarantee on the regret ratio. The Greedy algorithm by Nanongkai et al. [4] is based on linear programming and similar to the approaches by Chester et al. [3] and Qiu et al. [6]. Using an LP, the algorithms greedily add points to the solution subset until reaching the specified output size. On the other hand, the GeoGreedy algorithm by Peng et al. [5] greedily derives a solution subset with geometric computations.

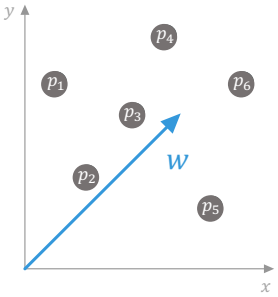
Lastly, the approximation algorithms provide solution subsets to the kRMS problem in arbitrary dimensions alongside a theoretical bound on the resulting regret ratio. Nanongkai et al. [4] introduce the efficient Cube algorithm tackling the 1RMS problem. The algorithm partitions the multi-dimensional space into hypercubes by constructing $t = \lfloor (r - d + 1)^{d-1} \rfloor$ equally-sized intervals in the first $d - 1$ dimensions. From each hypercube the point having the highest value in the last dimension is added to the solution subset. Thus, the Cube algorithm returns a subset of size at most r and a regret ratio of at most $\frac{d-1}{t+d-1}$. The running time of the Cube algorithm is in $\mathcal{O}(rnd)$ with the space consumption being in $\mathcal{O}(rd+n)$. Further, the Sphere algorithm by Xie et al. [7] improves the guarantee of the Cube algorithm by sampling a set of utility functions in order to find points with high scores on them. The authors ensure that the solution subset contains a point which is similar to the best-suited point in the entire data set for each utility function. Similarly, the HittingSet approach introduced by Agarwal et al. [1] samples utility functions and reduces the kRMS problem to the hitting set problem. Using an already existing algorithm, the authors solve the corresponding instance of the hitting set problem and produce the smallest subset not exceeding a given regret ratio.

1.2 Contribution

In this work, we propose the novel hierarchical regret minimizing algorithm HRMS approximating regret minimizing sets in arbitrary dimension. The divide and conquer approach partitions the input set into smaller subsets and applies a pre-existing regret minimizing algorithm on each of the subsets to obtain intermediate regret minimizing sets. These sets are then merged in a hierarchical manner until only the final regret minimizing set remains. The hierarchical regret minimizing algorithm allows for user-defined adjustments to its hierarchy in order to influence the practical running time and solution quality of the HRMS algorithm. Furthermore, we introduce a post-processing technique based on Pareto-optimal points which efficiently improves the solution quality of the output of any given regret minimizing algorithm.

2 Preliminaries

We consider sets of multi-dimensional points with each point having a non-negative attribute value per dimension. More formally, we define a set D of size $n \in \mathbb{N}$ in d dimensions consisting of points $p = (p_1, \ldots, p_d) \in \mathbb{R}^d_+$ where p_i denotes the value in the i^{th} dimension. Figure 2 depicts an example data set D of size $n = 6$ in two dimensions. Modeling the user is done by defining a weight vector $w = \langle w_1, \ldots, w_d \rangle \in \mathbb{R}^d_+$ according to the user's preference in each dimension. Further, we define the user's utility function as the dot product of each point $p \in D$ and the weight vector w. The score of a point $p \in D$ w.r.t. the weight vector w is $\text{score}(p, w) = p \cdot w = \sum_{i=1}^{d} p_i w_i$. In Figure 2 we consider only the weight vector of one specific user, namely $w = \langle 0.5, 0.5 \rangle$. The score of each point in D w.r.t. w is given in the table on the right.



	p_i^1	p_i^2	$\text{score}(p_i, w)$
p_1	1.0	6.0	3.50
p_2	2.0	3.0	2.50
p_3	3.5	5.0	4.25
p_4	4.5	7.5	6.00
p_5	6.0	2.0	4.00
p_6	7.0	6.0	6.50

Figure 2. Two-dimensional data set D of size $n = 6$ with the user weight vector w depicted on the left. The table shows the values in the 1st and 2nd dimension together with the score of each point on the weight vector w.

Based on the score we construct a descending ordering on D such that the first ranked point is the one having the highest score in D. Accordingly, we denote the k^{th} ranking point in D on the weight vector w as $D^{(k,w)}$. The ordering on D in Figure 2 is $(p_6, p_4, p_3, p_5, p_1, p_2)$ indicating that $p_6 = D^{(1,w}$ is the highest ranking point with respect to w.

Regarding a subset $R \subseteq D$ of points, we define its gain as the score of the highest ranking point in R with respect to w as $\text{gain}(R, w) = \text{score}(R^{(1,w)}, w)$. Consider the subset $R = \{p_1, p_3, p_4\} \subseteq D$ in Figure 2, the $\text{gain}(R, w)$ is equal to the score of the highest ranked point in R w.r.t. the weight vector w which is p_4.

Measuring how closely a subset R approximates the data set D for a given user w is done by considering the gain of the subset relative to the gain of the entire set. In other words, we compare the user's happiness with the best-ranking option in the subset R with the best choice in the entire data set D. The following distance metric called regret ratio captures exactly this.

Definition 1 (Regret Ratio)**.** *Given a subset $R \subseteq D$ and a weight vector w, the regret ratio is*

$$\text{regret ratio}(R, w) = \frac{\max(0, \text{gain}(D, w) - \text{gain}(R, w))}{\text{gain}(D, w)}$$

For the subset $R = \{p_1, p_3, p_4\}$ we obtain a $\text{regret ratio}(R, w) = 0.0769$ as point p_6 is the highest ranking in D and point p_4 is the highest ranking in R. In other words, the user described by the weight w has a low regret when choosing a point from the subset R instead of choosing one from D.

Concerning the RMS problem, we aim at minimizing the regret of the most unhappy user among all possible users. Therefore, we define the maximum regret ratio of a subset R as the regret ratio of the most unhappy user among all possible user weight vectors $w \in [0, 1]^d$.

Definition 2 (Maximum Regret Ratio)**.** *Let $\mathcal{L}$ denote all vectors in $[0,1]^d$. The maximum regret ratio of a subset $R \subseteq D$ is*

$$\text{regret ratio}(R) = \sup_{w \in \mathcal{L}} \text{regret ratio}(R, w)$$

For the example subset R the regret ratio is $\text{regret ratio}(R) = 0.351$ hence there exists at least one user w' for which the $\text{regret ratio}(R, w') = 0.351$. The user w' produces the maximum such ratio among all possible users and is the most unhappy one in our scenario.

Finally we define a regret minimizing set $R_{r,D}$ as the subset $R \subseteq D$ of size r having the smallest regret ratio among all subsets of the same size.

Definition 3 (Regret Minimizing Set)**.** *A regret minimizing set of size r on a data set D is*

$$R_{r,D} = \underset{R \subseteq D, |R|=r}{\arg\min} \text{regret ratio}(R)$$

Since the maximum regret ratio of the subset R is equal to 0.351 it is most likely not optimal. Instead, we aim to find a regret minimizing set of size 3 on the data set D. To this end, we compute the maximum regret ratio of all 20 possible subsets $R' \subseteq D$ of size 3. Among which we select the one with the smallest maximum regret ratio, namely $R' = \{p_1, p_4, p_6\}$. The subset R' has a maximum regret ratio of zero, consequently it is a regret minimizing set of size 3 on the data set D.

Definition 4 (RMS Problem)**.** *Given a data set D of n points in d dimensions and a non-negative integer r, return a regret minimizing set of size r on D.*

3 Hierarchical Regret Minimizing Algorithm

The hierarchical regret minimizing set algorithm (HRMS algorithm) follows the divide and conquer paradigm to efficiently compute a representative subset approximating the RMS problem in arbitrary dimension. Figure 1 depicts the high-level schema of the HRMS algorithm partitioned into a dividing phase, the application of a regret minimizing algorithm and a merging phase.

Firstly, the input set is recursively divided into sufficiently small subsets in order to conquer any large data set. For each of the subsets we use an already existing regret minimizing algorithm to obtain regret minimizing sets (short RMS) of a fixed size. Lastly, the HRMS algorithm merges pairs of regret minimizing sets in a hierarchical structure until only one set remains. The merging is done by filtering a subset of points from the pair of regret minimizing sets thus maintaining a fixed output size throughout the hierarchy.

The input to Algorithm 1 is a data set D consisting of n points in d dimensions and a non-negative integer r indicating the output size. In order to use any regret minimizing algorithm during the conquer phase, we define the maximum depth x of the hierarchy such that each of the 2^x subsets contains at least $r+1$ points. Otherwise, applying an RMS algorithm to compute a regret minimizing set of size r is superfluous as we could simply choose all points within the subset. Using said RMS algorithm on all 2^x subsets, we obtain 2^x regret minimizing sets of size r. The HRMS algorithm continues by merging the regret minimizing sets in pairs of two in each layer of the hierarchical structure depicted in Figure 1. During the merge, the output size r remains as we do not want to further decrease the number of points. Since we have x depth layers, the merging of pairs is performed x times until depth $i = 1$ where the final regret minimizing set of size r results from the last merge.

Algorithm 1: Hierarchical Regret Minimizing Algorithm

```
Define the maximum depth x such that n / 2^(x+1) ≤ r.
Divide D into 2^x subsets, each containing at least r + 1 points.
for each subset S ⊆ D do
    R = RMS(S)
    Add R to the list of intermediate sets in depth x.
for each depth i = x, ..., 1 do
    for each pair of intermediate sets R_A and R_B do
        R_M = MERGE(R_A, R_B)
        Add R_M to the list of intermediate sets in depth i − 1.
Define R as the intermediate set in depth 0.
return R
```

Considering the running time of the HRMS algorithm, dividing the input set of size n into 2^x subsets is done in $\mathcal{O}(n)$. Let t_{RMS} be the running time of the chosen RMS algorithm executed on the subset S of size r. The RMS algorithm is executed on each of the 2^x subsets thus it requires $\mathcal{O}(2^x \cdot t_{\text{RMS}})$ time. Let t_{MERGE} be the running time of the chosen merging technique for merging intermediate sets R_A and R_B each of size r. At depth i we merge 2^i intermediate sets in pairs of two. Hence, we merge 2^{i-1} pairs at each depth $i = x, \ldots, 1$. Overall the merging of pairs of intermediate sets is in $\mathcal{O}(2^x \cdot t_{\text{MERGE}})$. The total running time of the HRMS algorithm is in $\mathcal{O}(n + 2^x \cdot (t_{\text{RMS}} + t_{\text{MERGE}}))$. However, note that the height x of the hierarchy is in $\mathcal{O}(\log n)$ due to the recursive splitting into two halves. Hence, the term 2^x is in $\mathcal{O}(n)$.

Let s_{RMS} be the space consumption of the chosen RMS algorithm once executed on the subset S and let s_{MERGE} be the space consumption of the selected merging technique. Maintaining the 2^x subsets during the divide phase is in $\mathcal{O}(n)$. Executing the RMS algorithm on all subsets requires $\mathcal{O}(2^x \cdot s_{\text{RMS}})$ space and the merging phase requires $\mathcal{O}(2^x \cdot s_{\text{MERGE}})$ space. Consequently, the space consumption of the HRMS algorithm is in $\mathcal{O}(n + 2^x \cdot (s_{\text{RMS}} + s_{\text{MERGE}}))$.

3.1 Merging of Intermediate Regret Minimizing Sets

During the merging phase, we repeatedly combine regret minimizing sets R_A and R_B each of size r to form a new regret minimizing set R_M of size r. Therefore, we need to select r points from the union $R_A \cup R_B$ whilst not compromising the regret ratio of R_M. A naive approach would be to randomly sample r points from the union $R_A \cup R_B$ to form the set R_M during one merging step. This approach works efficiently on any given input but does not provide the best approximation for the regret ratio. On the other hand, since we already used an RMS algorithm to obtain the intermediate sets R_A and R_B, we can simple rerun said RMS algorithm on the union $R_A \cup R_B$ to obtain the set R_M of size r. However, in practice this merging technique increases the running time of the HRMS algorithm heavily and thus is not recommend. Instead, we propose the *sorted merge* which uses the observation in Section 2 that points rank higher for a specific user weight vector if their values in each dimension are higher than the values of other points. For arbitrary dimension d we conclude that points having high values in either one of the d dimensions are more likely to rank higher for any user weight vector than points having lower values in the same dimension. The sorted merge initially sorts R_A and R_B decreasingly in each of the d dimensions. Then,

the first $\lfloor \frac{r}{2d} \rfloor$ points from the sets R_A and R_B corresponding to the decreasing order in the first dimension are retrieved. This procedure is repeated for each dimension and we obtain the set R_M consisting of at most r points. The sorted merge efficiently merges two RMS as we only sort sets of size r which is typically much smaller than the input size n. Additionally, we select the most promising points in each dimension such that as many users as possible retrieve high scores from points in R_M leading to a lower regret ratio.

3.2 *Suggested Configuration of the HRMS Algorithm*

The depth x of the hierarchical framework is defined by the user such that for an input of $x = 0$ the HRMS algorithm is performed with the maximum depth defined in Algorithm 1. For any input $x > 0$, the hierarchy is trimmed to the specified depth. For the RMS algorithm used within the framework of HRMS we choose the Cube algorithm described in Section 1.1 due to its efficient running time in arbitrary dimension and approximation guarantee. Lastly, for the merging phase we select the sorted merge described in Section 3.1.

The running time of the Cube algorithm when executed on input sets of size $\frac{n}{2^x}$ to obtain intermediate regret minimizing sets is $t_{\text{RMS}} \in \mathcal{O}(r\frac{n}{2^x}d)$ with the space consumption $s_{\text{RMS}} \in \mathcal{O}(rd)$. The sorted merge sorts the intermediate RMS of size r in each dimension using MergeSort, thus the running time is $t_{\text{MERGE}} \in \mathcal{O}(dr \log r)$ and the space consumption $s_{\text{MERGE}} \in \mathcal{O}(r)$.

Consequently, for the proposed configuration of HRMS the running time is in $\mathcal{O}(rnd + 2^x dr \log r)$ with the space consumption being in $\mathcal{O}(n + 2^x rd)$. As mentioned previously, the term $2^x \in \mathcal{O}(n)$.

4 Pareto Post-Processing

We propose a post-processing technique based on Pareto-optimal points that is applicable on the output of any regret minimizing algorithm. A point p *dominates* another point q if the value of p in each dimension is equal or higher than the value of q and if p has a higher value than q in at least one dimension. If the point p is not dominated by any other point q in the data set D, then p is a *Pareto-optimal* point in D. The set of all non-dominated points is then called a *Pareto set*.

Consider the example data set $D = p_1, \ldots, p_6$ in Figure 3 where only two points are Pareto-optimal, namely p_4 and p_6. Any points situated in the blue shaded regions are dominated by either p_4 and/or p_6. For example, the point p_1 is dominated by p_4 due to p_4 having higher x- and y-values. On the other hand, point p_6 has the same y-value as p_1 but the x-value of p_6 is higher than the one of p_1. Thus, p_1 is also dominated by p_6. Since p_4 p_6 do not dominate each other and are not dominated by any other points in D, the set $\{p_4, p_6\}$ is the Pareto set in D.

Based on the aforementioned definition, we conclude that for any user weight vector w the point p with the highest score(p, w) is equally as good as all other points in all dimensions and additionally, p is superior to all other points in at least one dimensions leading to a higher score. Therefore, the point p having the highest score for the user defined by w is Pareto-optimal. Since this property holds for all possible user weight vectors, it seems reasonable to extract all Pareto-optimal points from the data set to form an ideal regret minimizing set with a maximum regret ratio of zero. In Figure 3 the Pareto set $\{p_4, p_6\}$ provides the best choice for any possible user and thus we could simply return the Pareto set as the regret minimizing set. However, this approach has the following two shortcomings rendering it unsuitable for the RMS problem. Firstly, the size of the Pareto set is not fixed and in the worst case the entire data set might be Pareto optimal. Thus, the Pareto set would not provide a sufficiently small subset to represent the entire data set. Secondly, determining all Pareto-optimal points in a data set, in the form of a Pareto filter, is costly as we need to compare each point to all other points. Therefore, computing the Pareto set of a given data set is not applicable for the regret minimization problem in practice.

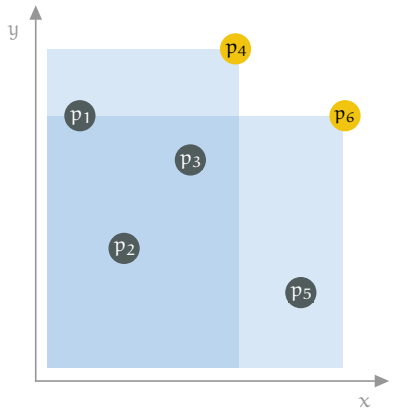


Figure 3. In this example data set D two points are Pareto-optimal (yellow), namely p_4 and p_6. The other four points lie within the blue shaded regions and thus are dominated by either one or both of the Pareto-optimal points.

The following post-processing technique in Algorithm 2 based on Pareto-optimal points overcomes the deficiencies of the aforementioned Pareto filtering. Given a regret minimizing set R, we traverse all points $q \in R$ and determine if q is dominated by some point $p \in D$. If we find a better choice in D in the form of the point p, we simply replace the point q by the point p within the regret minimizing set R. We continue traversing D with this newly determined point p. After one iteration for a given point $q \in R$, we either know that q is Pareto-optimal in D and it remains in R or it is replaced by a Pareto-optimal point from D. Hence, this post-processing technique ensures that the resulting set contains only Pareto-optimal points. Since p dominates q, the maximum regret ratio of R does not increase and replacing q with p in the set R does not worsen its solution quality.

Algorithm 2: Pareto Post-Processing

```
for each point q ∈ R do
    for each point p ∈ D do
        if q is dominated by p then
            Replace q with p in R.
return R
```

Further, we replace at most r points in R, thus, the output size does not increase during the post-processing, overcoming the issue of uncontrollable output sizes for the Pareto filter. It is worth mentioning, that one dominating point that replaces a dominated point in R might also dominate other points in R. In this case, the dominating point is only added once and the output size decreases. However, the shrinking of the output set due to replacing several dominated points by one dominating point is acceptable for RMS problems since we want to find small representative subsets.

Since the post-processing performs one traversal of the entire data set D for each point in R comparing the values in all d dimensions, its running time is in $\mathcal{O}(rnd)$ which is significantly better than the running time of $\mathcal{O}(dn^2)$ for the Pareto filter. Since the Pareto post-

processing works directly on the solution set but requires the entire data set to determine dominating points, its space consumption is in $\mathcal{O}(n)$.

Due to its simplicity, the Pareto post-processing is suitable for the approximate regret minimizing algorithms Cube and HRMS discussed in this work but also the Sphere [7] algorithm and the Hitting Set [4] approach described in Section 1.1.

5 Experimental Evaluation

We implemented the HRMS algorithm according to the configuration described in Section 3.2 and the Cube algorithm [4] in C++. The experiments were conducted on a single core of a 4.5 GHz AMD Ryzen 9 7950X 16-Core Processor with 188 GB of RAM. Running times and regret ratios are always averaged over 100 generated instances per tested value of n.

5.1 Approximating the Regret Ratio

In order to evaluate the solution quality of the HRMS algorithm, we need to obtain the maximum regret ratio of the solution subset $R \subseteq D$ among all possible user weight vectors as described in Definition 2. To this end, we uniformly sample $1,000$ user weight vectors $w \in [0,1]^d$ ensuring that the extreme values 0.0 and 1.0 are represented at least once in each dimension. We approximate the maximum regret ratio of a subset R by computing the regret ratio(R, w) for each sampled vector w and extract the maximum among all weight vectors. In Figure 4 we experimentally explore the effect of the sample size of user weight vectors on the resulting approximated regret ratio of a given subset R. We conclude that the obtained regret ratios are sufficiently similar even when comparing samples of size 100 and $100,000$. Since computing the regret ratio for a given user weight vector involves traversing the entire input set D, we use a sample size of $1,000$ to efficiently approximate the regret ratio for our experiments. Agarwal et al. [1] use a similar approach by randomly sampling $20,000$ user weight vectors to approximate the regret ratio of a given subset. Note that whenever we compare the regret ratios of multiple algorithms, the same sample of user weight vectors is used for all considered solution subsets to ensure comparability of the obtained regret ratios.

Figure 4. Approximating the maximum regret ratio for a given subset $R \subseteq D$ is done by randomly sampling $100, \ldots, 100,000$ user weight vectors $w \in [0,1]^d$ and obtaining the maximum value among the regret ratios of all users in the sample.

5.2 Generated Data Sets

For our experiments, we use input sets D consisting of n points in d dimensions. Generating the input sets is done by randomly sampling d values from the uniform distribution in the range $[0, nd)$ for each point in the input set. The output size in the following is $r = \max(20, 2d)$ to ensure that the sorted merge for the HRMS algorithm retrieves at least one point from each the intermediate set in each dimension.

Firstly, we assess the average running times and regret ratios obtained by executing the HRMS algorithm and Cube algorithm on input sets of up to $n = 10^6$ points in $d = 2, \ldots, 30$ dimensions with the selection of $d = 2, 4, 30$ being depicted in Figure 5. Using the hierarchical framework of the HRMS increases the running time slightly compared to Cube algorithm in $d = 2$ and $d = 4$ dimensions. Here, calling Cube on D is faster than calling it on $2^x = 8$ subsets and performing the sorted merge during HRMS. However, with increasing dimensions of the input, the divide and conquer approach of HRMS proves useful as its running time is lower for $d > 10$. Hence, subdividing the input into smaller subsets efficiently tackles the problem of approximating the RMS. In general, the running times of HRMS and Cube are similar. Thus, using the hierarchical framework does not introduce a large overhead for small values of d and yields an improvement in running time for higher dimensions.

In all tested dimensions $d = 2, \ldots, 30$ the HRMS algorithm consistently produces better regret ratios than the Cube algorithm. Especially for $d = 2$ the improvement in regret ratio is up to a factor of 8052 and on average among all n by a factor of 33. Hence, leveraging the hierarchical framework we gain significant improvements in solution quality whilst not compromising on the efficiency in terms of running time. The HRMS algorithm uses the Cube algorithm to obtain intermediate regret minimizing sets and proceeds by merging the sets. Further, it yields better results than a single call to the Cube algorithm on the entire data set D. Hence, we conclude that the improvement in solution quality is due to the hierarchical structure of HRMS together with the chosen merging technique. For $d = 4$ the HRMS yields regret ratios which are lower by a factor of up to 560 than Cube and on average provides an improvement by a factor of 1.7. With increasing number of dimensions, the problem of finding a small representative subset such that every possible user is sufficiently happy becomes increasingly more complex. Already adding two objectives from $d = 2$ to $d = 4$ increases the regret ratio of 0.002 to 0.18, respectively. In $d = 30$ dimensions, the HRMS produces solution subsets which are better than the results of Cube by up to a factor of 16 and on average by a factor of 1.1. However, the HRMS algorithm optimizes for 30 objectives simultaneously and provides representing subsets that still make the most unhappy user about 77% happy during the decision making process with being limited to a subset of size 60 representing $n = 10^6$ possibilities.

Further, we evaluate the impact of the depth x in the hierarchy of HRMS on its running time and regret ratio in $d = 2$ dimensions. For depths $x = 3, \ldots, 10$ the HRMS exhibits faster running times than Cube as depicted in Figure 6. As for the remaining depth values up to the maximum depth $x = 15$, the running time of HRMS exceeds the time required by Cube. Regarding the solution quality in Figure 7, for depths $x = 1, 2, 3$ the regret ratio of HRMS are significantly lower than the regret ratios of Cube with $x = 3$ providing the overall lowest regret ratios in two dimensions. Hence, we chose $x = 3$ for the experiments in higher dimensions. Note that for $d \geq 4$ using the maximal depth for the hierarchical framework provides consistently better solution qualities of HRMS compared to Cube. However, the

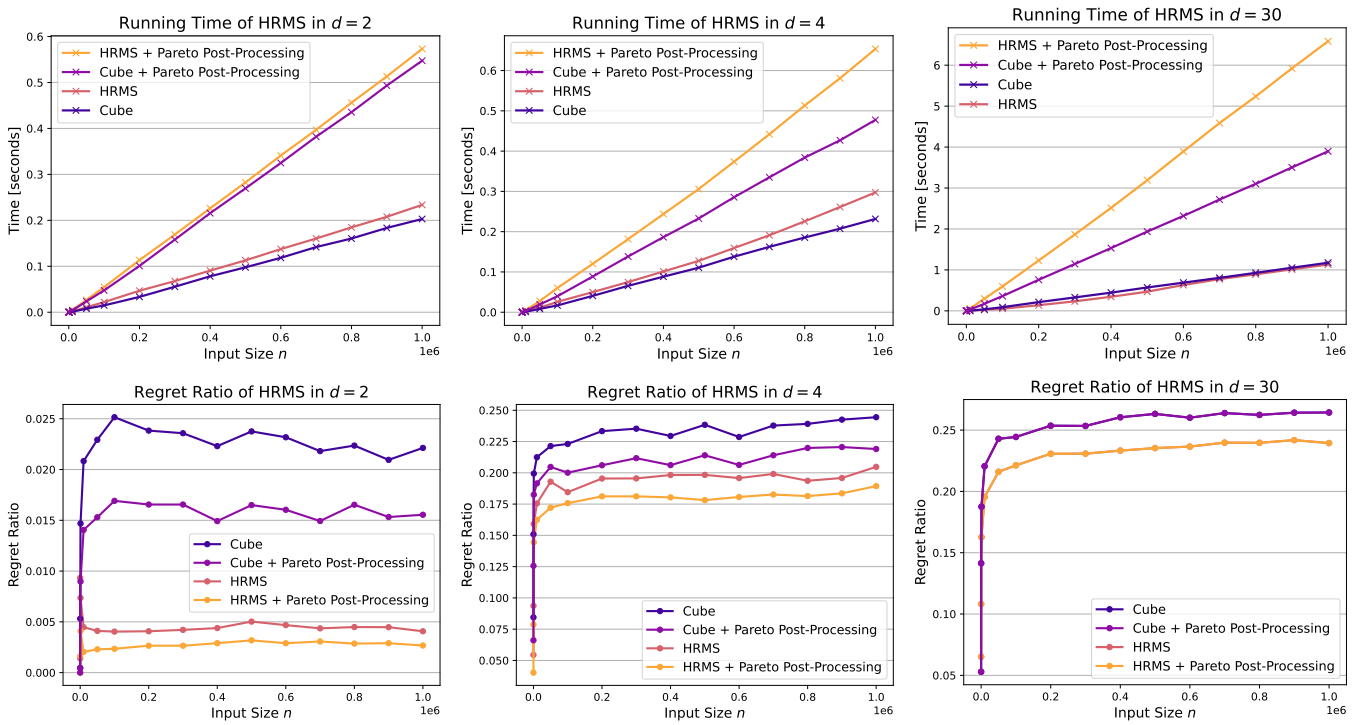


Figure 5. Average running times and maximum regret ratios for the HRMS algorithm and the Cube algorithm on inputs of sizes up to $n = 10^6$ in $d = 2, 4, 30$ dimensions using the depth $x = 3$ for the hierarchical framework. Additionally, the resulting running times and regret ratios after applying the Pareto Post-Processing on both algorithms are reported.

running time of HRMS using the maximal depth is higher as when using depths $x = 3$ or $x = 1$. Thus, we suggest the depth value $x = 3$ to optimize for both the running time and the solution quality of HRMS.

In Figure 8 we explore the reasoning why calling the Cube algorithm on D provides higher regret ratios than using the HRMS algorithm which calls Cube on smaller subsets and performs the merging of intermediate sets. Therefore we measure the ratio of Pareto-optimal points in the output sets of HRMS and Cube for $d = 2, \ldots, 5$. In every dimension, the HRMS algorithm includes more Pareto-optimal points in its output than Cube. Recall that all Pareto-optimal points together as the Pareto set yield a regret ratio of zero but the size of the Pareto set might be much larger than the output size r. Hence, the improvement in solution quality by HRMS is due to retrieving more Pareto-optimal points from D than Cube. During the sorted merge, we emphasize on selecting points having high values in each of the dimensions which corresponds to the concept of Pareto-optimality.

Further, we observe an increase in Pareto-optimal points in the output sets with higher dimensionality of the input. In general, more points in higher dimensional data sets are Pareto-optimal and for $d > 5$ the ratio for both algorithms gets closer to 1.0. However, both algorithms do not return subsets which consist entirely of Pareto-optimal points. Hence, applying the Pareto Post-Processing to further improve the solution quality of the algorithms is applicable.

The Pareto Post-Processing (PPP) replaces dominated points in the output set by points that provide better value combinations from the input data set D. Figure 5 depicts the running times and regret ratios obtained after applying the PPP in $d = 2, 4, 30$ dimensions. The PPP consistently yields improvements in the solution quality of both algorithms whilst introducing only a slight increase in running time. The largest improvement is obtained for Cube in $d = 2$ where PPP

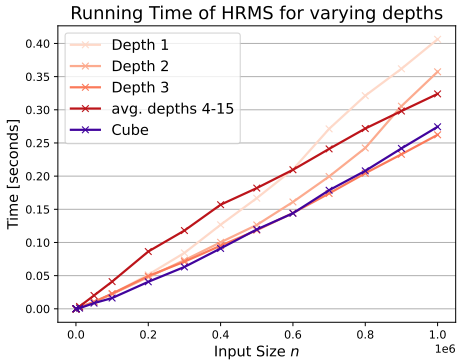


Figure 6. Average running times for varying depths of the hierarchical framework ranging from $x = 1$ to the maximal depth $x = 15$ for input sets of sizes up to $n = 10^6$ in $d = 2$ dimensions and an output size of $r = 20$.

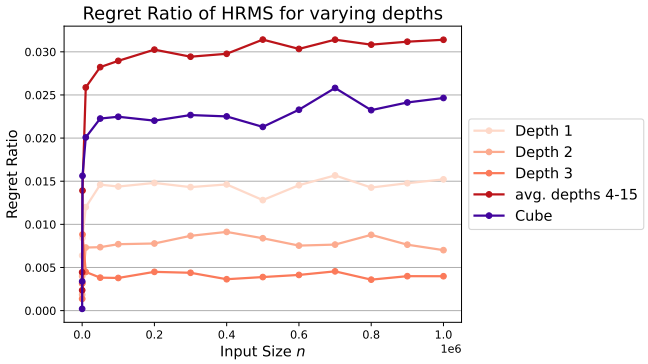


Figure 7. Average regret ratios for varying depths of the hierarchical framework ranging from $x = 1$ to the maximal depth $x = 15$ for input sets of sizes up to $n = 10^6$ in $d = 2$ dimensions with an output size of $r = 20$.

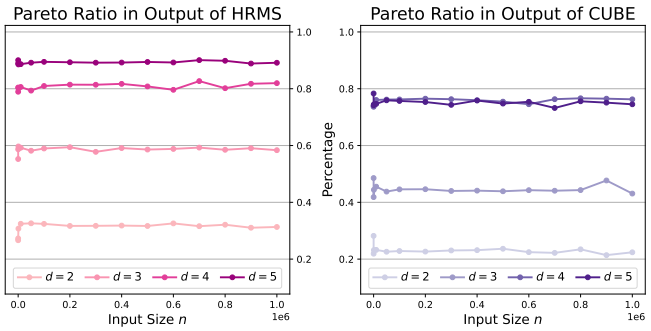


Figure 8. Comparison of the ratio of Pareto-optimal points in the output subset of the HRMS algorithm and the Cube algorithm in $d = 2, \ldots, 5$ dimensions for the output size $r = 20$ and depth value $x = 3$ for the HRMS.

yields solution subsets with regret ratios being better by up to a factor of 11.8 than the result of Cube. Since the output of HRMS already contains more Pareto-optimal points, the PPP provides better results by up to a factor of 5.8. For $d > 10$ and especially for $d = 30$, the number of Pareto-optimal points in D increases with the number of dimensions. Thus, both algorithms already return subsets which consist of only Pareto-optimal points. Thus, the PPP does not improve on the solution quality since it does not replace any dominated points. We conclude that the PPP is applicable for any regret minimizing algorithm in $d \leq 10$ dimensions where it efficiently improves on the already obtained regret ratio.

5.3 *Benchmark Data Sets*

In addition to the generated data, we evaluate the hierarchical regret minimizing algorithm on two real-world benchmark data sets typically used for RMS problems. The first data set *Basketball* consists of $n = 21,961$ points in $d = 5$ dimensions and as provided by Agarwal et al. [1]. The points in the data set represent basketball players and the five dimensions correspond to the player statistics *points*, *rebounds*, *blocks*, *assists* and *fouls*. An exemplary objective for this scenario is to find a representative subset of 20 players to be invited to an event such that every attendant of the event is sufficiently happy with meeting those players with respect to the performance statistics.

Table 1. Average running times in milliseconds and regret ratios of the output sets generated by the HRMS algorithm using maximal depth $x = 7, \ldots, 10$ (depending on the output size r) and the Cube algorithm on the *Basekteball* data set.

r	Running Time				Regret Ratio			
	HRMS		Cube		HRMS		Cube	
		+ post		+ post		+ post		+ post
10	4.17	8.21	0.62	2.62	0.2108	0.1947	0.2181	0.2181
20	10.39	18.55	1.29	6.30	0.1860	0.1836	0.2150	0.2150
50	8.36	28.58	1.41	6.43	0.0749	0.0539	0.2158	0.2158
100	16.71	57.66	6.00	18.89	0.0533	0.0513	0.1918	0.1917

The results are shown in Table 1 for output sizes $r = 10, \ldots, 100$ where the HRMS algorithm uses the maximal depth of the hierarchical framework. In terms of running time, the HRMS is slightly slower than the Cube algorithm. The Pareto Post-Processing imposes an increase in running time for both algorithms. As for the regret ratios, the HRMS consistently provides smaller values than Cube on all output sizes. Especially for $r = 50$ and $r = 100$ the HRMS returns subsets which produce significantly lower regret ratios by factor of 2.9 and 3.6, respectively, than the Cube algorithm. Applying the PPP further improves the solution quality of HRMS whereas the results for the Cube algorithm remain mostly unchanged. Further, for $r = 100$ the HRMS requires 16.71 ms for a regret ratio of 0.0533 and the Cube algorithm together with the PPP takes 18.89 ms resulting in a regret ratio of 0.1917.

Table 2. Average running times in milliseconds and regret ratios of the output sets generated by the HRMS algorithm using maximal depth $x = 10, \ldots, 13$ (depending on the output size r) and the Cube algorithm on the *El Nino* data set.

r	Running Time				Regret Ratio			
	HRMS		Cube		HRMS		Cube	
		+ post		+ post		+ post		+ post
10	50.04	82.65	8.80	24.64	0.0741	0.0741	0.0753	0.0753
20	106.27	171.62	19.31	57.84	0.0644	0.0642	0.0747	0.0747
50	76.77	241.24	19.24	57.86	0.0577	0.0551	0.0740	0.0740
100	135.10	460.10	65.28	175.47	0.0553	0.0520	0.0647	0.0641

The second benchmark data set is called *El Nino* as used by Agarwal et al. [1]. The data set contains weather data of $n = 178,079$ points where the $d = 5$ dimensions represent measurements such as *wind speed* and *water temperature* of buoys placed in the Pacific ocean. Comparing the running times depicted in Table 2 of HRMS and Cube, we observe similar patterns as with the previous data set. Cube performs faster than HRMS on all output sizes but all executions were well below half of one seconds. On the other hand, HRMS produces lower regret ratios than Cube by factors of up 1.3. The PPP achieves minor improvements for both algorithms. Thus, we conclude that the output sets of HRMS and Cube already contain mostly Pareto-optimal points such that the PPP cannot further improve their solution quality. For $r = 100$ we observe that HRMS is faster and obtains a lower regret ratio than Cube together with PPP as for the *Basketball* data set.

6 Conclusion and Future Work

We introduce the novel hierarchical regret minimizing algorithm HRMS tackling the regret minimization problem in arbitrary dimension. As a divide and conquer approach, the HRMS algorithm is capable of leveraging pre-existing regret minimizing algorithms together with new merging techniques to efficiently approximate regret minimizing sets. The hierarchical framework of HRMS allows for user-defined adjustments such as the definition of the depth of the hierarchy as well as the selection of well-established regret minimizing algorithms best-suited for the specific scenario. Based on our experiments, we provide a suggested configuration of the HRMS algorithm to allow for effortless integration. Further, we develop a post-processing technique (PPP) based on Pareto-optimal points which efficiently improves the solution quality and is applicable for any regret minimizing algorithm. The experimental evaluation shows that our HRMS algorithm significantly outperforms the Cube algorithm [4] in terms of regret ratio not only on generated data up to $d = 30$ dimensions but also on established benchmark data sets.

Future work will target further adjustments to the hierarchical framework of HRMS such as varying the depth of the merging phase and extending the internally used output size for intermediate sets to $r' = dr$. It would be interesting to explore more intricate partitioning methods during the divide phase of the algorithm to further improve on its solution quality. Additionally, the HRMS algorithm allows for parallelization of the calls to the regret minimizing algorithm within the framework such that 2^x subsets are processed in parallel as well as the merging per level. Naturally, we want to incorporate other regret minimizing algorithms e.g. Sphere [7] into the HRMS algorithm and evaluate the performance of our approach.

References

[1] P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri. Efficient Algorithms for k-Regret Minimizing Sets. In *16th International Symposium on Experimental Algorithms (SEA 2017)*, pages 7:1–7:23, 2017.

[2] W. Cao, J. Li, H. Wang, K. Wang, R. Wang, R. C.-W. Wong, and W. Zhan. k-Regret Minimizing Set: Efficient Algorithms and Hardness. In *20th International Conference on Database Theory (ICDT 2017)*, pages 11:1–11:19, 2017.

[3] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Computing k-regret minimizing sets. *Proceedings of the VLDB Endowment*, 7(5):389–400, 2014.

[4] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-minimizing representative databases. *Proceedings of the VLDB Endowment*, 3(1-2):1114–1124, 2010.

[5] P. Peng and R. C.-W. Wong. Geometry approach for k-regret query. In *IEEE 30th International Conference on Data Engineering (ICDE 2014)*, pages 772–783, 2014.

[6] X. Qiu, J. Zheng, Q. Dong, and X. Huang. Speed-up algorithms for happiness-maximizing representative databases. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data*, pages 321–335, 2018.

[7] M. Xie, R. C.-W. Wong, J. Li, C. Long, and A. Lall. Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In *Proceedings of the 2018 International Conference on Management of Data*, pages 959–974, 2018.

[8] M. Xie, R. C.-W. Wong, and A. Lall. An experimental survey of regret minimization query and variants: bridging the best worlds between top-k query and skyline query. *The VLDB Journal*, 29(1):147–175, 2020.