# Multi-Criteria Route Planning with Little Regret

**Carina Truschel**
University of Konstanz, Germany

**Sabine Storandt**
University of Konstanz, Germany

## Abstract

Multi-criteria route planning arises naturally in real-world navigation scenarios where users care about more than just one objective – such as minimizing travel time while also avoiding steep inclines or unpaved surfaces or toll routes. To capture the possible trade-offs between competing criteria, many algorithms compute the set of Pareto-optimal paths, which are paths that are not dominated by others with respect to the considered cost vectors. However, the number of Pareto-optimal paths can grow exponentially with the size of the input graph. This leads to significant computational overhead and results in large output sets that overwhelm users with too many alternatives. In this work, we present a technique based on the notion of regret minimization that efficiently filters the Pareto set during or after the search to a subset of specified size. Regret minimizing algorithms identify such a representative solution subset by considering how any possible user values any subset with respect to the objectives. We prove that regret-based filtering provides us with quality guarantees for the two main query types that are considered in the context of multi-criteria route planning, namely constrained shortest path queries and personalized path queries. Furthermore, we design a novel regret minimization algorithm that works for any number of criteria, is easy to implement and produces solutions with much smaller regret value than the most commonly used baseline algorithm. We carefully describe how to incorporate our regret minimization algorithm into existing route planning techniques to drastically reduce their running times and space consumption, while still returning paths that are close-to-optimal.

**2012 ACM Subject Classification** Theory of computation → Data structures design and analysis; Theory of computation → Shortest paths

**Keywords and phrases** Pareto-optimality, Regret minimization, Contraction Hierarchies

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2025.13

**Funding** *Carina Truschel*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 251654672 – TRR 161.

## 1 Introduction

In many real-world navigation tasks, the optimal route is not sufficiently defined by a single criterion. For example, cyclists might want to follow the shortest path in terms of distance but only if it does not include too many steep climbs. For electric vehicles, travel time might be a primary objective but limited energy consumption should also be taken into account. Furthermore, users might want to also consider criteria as road surface quality, curviness, or tolls among others. Multi-criteria route planning addresses these scenarios. Here, given a graph $G(V, E)$, the edge costs are $d$-dimensional vectors $c : E \to \mathbb{R}^d_{\geq 0}$, where, for example, $c_1$ encodes travel time, $c_2$ distance, $c_3$ gas or energy consumption, and so on. As for the one-dimensional case, the cost of a path $\pi$ in the graph is the summed cost of its traversed edges $c(\pi) := \sum_{e \in \pi} c(e)$. A path $\pi$ dominates another path $\pi'$, if $c_i(\pi) \leq c_i(\pi')$ holds for all cost dimensions $i = 1, \ldots, d$ and $\exists i : c_i(\pi) < c_i(\pi')$. There are three main types of route planning queries that are of interest for a given input source-target-node pair $s, t \in V$:

- *Full Pareto Set (FPS).* Compute the set of Pareto-optimal paths from $s$ to $t$.

- *Constrained Shortest Path (CSP).* Given budgets $B_2, \ldots, B_d$, compute the $s$-$t$-path $\pi$ with minimum cost $c_1(\pi)$ under the constraint that for $i = 2, \ldots, d$ we have $c_i(\pi) \leq B_i$.
- *Personalized Shortest Path (PSP).* For a given input weight vector $\alpha \in \mathbb{R}^d_{\geq 0}$, compute the $s$-$t$-path $\pi$ that minimizes the personalized path cost $\alpha^T c(\pi)$.

It is well known, that CSP and PSP solutions are always Pareto-optimal. Thus, given the FPS, the last two types of queries can easily be answered. However, the goal for CSP and PSP is usually to compute the respective path without having to explore all Pareto-optimal paths. While the CSP problem is known to be NP-hard, PSP can indeed be solved in polynomial time using Dijkstra's algorithm on the personalized edge costs $\alpha^T c$. In [11] computing FPS for multi-criteria shortest paths in time-dependent train networks is tackled. As an example for CSP, finding constrained shortest paths for electric vehicles considers the travel time while not exceeding a maximum number of recharging stops [22] or the energy consumption does not exceed the capacity of the battery [4]. However, preprocessing based techniques to accelerate PSP queries also store sets of Pareto-optimal paths between selected node pairs [14].

The main issue is that the number of Pareto-optimal solutions can become huge (even in the bi-criteria case) and thus oftentimes further filtering needs to be applied to make route planning algorithms efficient and to present a sensible selection of alternatives to the user. Different ideas were proposed in the literature to filter a given set of Pareto-optimal elements. The goal is to efficiently maintain a core of solutions that are still representative of the full set. One such method is to define a relaxation of the dominance criterion. In [17] a tailored relaxation was implemented to reduce the number of Pareto-optimal journeys in public transport planning. For bi-objective shortest path problems, [15] describe an interesting approach to approximate the Pareto frontier using pairs of paths and guarantees per objective. Filtering the Pareto set in multi-criteria public transit routing in [8] uses fuzzy logic to identify significant journeys. In [7] restricted Pareto sets are used to provide a subset of the full Pareto set which excludes outliers having undesirable trade-offs between the criteria. A general and quite powerful approach for dominance relaxation in two-dimensional sets was described in [21]. Here the idea is to enlarge the dominance area, which for strict dominance is simply the lower left quadrant of the point. The problem with existing methods is that most of them are tailored to the bi-criteria case. Furthermore, one has little control over the number of Pareto-optimal elements that survive the filtering process. It could still be too large or it might happen that almost all solutions are filtered out.

In this paper, we use the notion of *regret* as introduced in [18] to identify representative subsets of size $k$, where $k$ can be chosen as desired. Given a set of $d$-dimensional elements $S$ and a subset $S'$, a user's regret measures their disappointment when they have to choose the best option according to their preferences from $S'$ instead of $S$. More formally, a user $u$ has a preference or utility function $f_u : S \to \mathbb{R}_{\geq 0}$ and the regret of the user with respect to $S'$ is defined as $r_u(S, S') = 1 - \max_{s \in S'} f_u(s) / \max_{s \in S} f_u(s)$. By definition, we have $r_u(S, S') \in [0, 1]$.[1] The regret of $S'$ is defined by the most regretful user, that is $r(S, S') := \max_{u \in U} r(u)$. Typically, the set of users $U$ is the set of all functions in a given class, most commonly the class of all non-negative linear combinations, that is $f_u(s) = \alpha_u^T s$, where the weight vector $\alpha_u \in \mathbb{R}^d_{\geq 0}$ describes the importance of each dimension for the user. Figure 1 illustrates these concepts.

Given $k \in \mathbb{N}$, the optimization goal of regret minimization algorithms is to find the subset $S'$ of size at most $k$ with smallest regret. Obviously, $S'$ can be restricted to only contain Pareto-optimal points from $S$, as a dominated point can never have a higher utility than

[1] In case $\max_{s \in S} f_u(S) = 0$, we simply set $r_u = 0$.

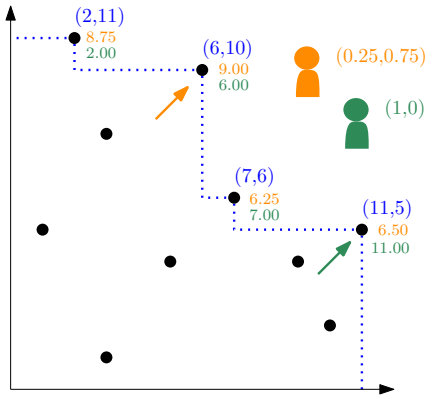


**Figure 1** Two-dimensional point set $S$ containing four Pareto-optimal elements (blue labels). Two example users (orange and green) assign each point a different utility based on their personal preferences. The respective values are provided for the Pareto-optimal elements. Among all points, the orange user assigns the highest value to point $(6, 10)$, namely $0.25 \cdot 6 + 0.75 \cdot 10 = 9$, while the green user prefers $(11, 5)$. For $S' = \{(2, 11), (6, 10)\}$, the orange user has a regret of 0, as their preferred choice from $S$ is contained in $S'$. For the green user the regret is $1 - \frac{6}{11} = 0.\overline{45}$.

the dominating point. Still, computing an optimal $S'$ poses an NP-hard problem [5]. We present a novel heuristic that computes subsets with small regret very quickly. This allows to integrate regret minimization in many applications where finding a representative subset of solutions is a frequent task, including multi-criteria route planning in road networks. Indeed, we show that leveraging regret filtering allows to compute constrained or personalized paths significantly faster, while ensuring that the resulting path (set) comes with small regret.

## 1.1 Further Related Work

A multitude of algorithms and heuristics has been proposed for all three main query types for multi-criteria route planning mentioned above. A* variants tailored to bi-objective search (BOA*) or multi-objective search (MOA*) have been demonstrated to greatly reduce the search space compared to Pareto-Dijkstra for FPS and CSP queries [24, 2, 20]. Methods to approximate the Pareto frontier were described in [15, 30]. In [29], an anytime approach was introduced that discovers a subset of Pareto-optimal paths quickly and then adds new solutions over time. A recent overview of the development of multi-objective route planning is provided in [27].

Even more pronounced speed-ups can be achieved if preprocessing is applied to the input graph. In [28], query answering for bi-criteria FPS using a contraction hierarchy (CH) data structure was reported to be two orders of magnitude faster than with BOA*. CH was also successfully applied to accelerate CSP [23, 13] as well PSP queries [12]. In [16], it was shown that filtering dominated points can be the bottleneck in bi-criteria CH construction as well as query answering. There, faster methods were proposed to compute Pareto-optimal path sets. But if the final set size remains large, the approach might still be too slow in practice. In all of these preprocessing-based techniques, sets of Pareto-optimal paths are precomputed and stored between selected pairs of nodes. While [12] allows to obtain approximate query results faster by only considering a subset of stored solutions on query time, non of the techniques apply filtering during the preprocessing, which results in data structures of substantial size.

## 1.2 Contribution

We show how multi-criteria route planning can be integrated with regret minimization. Compared to existing filtering methods, regret-based filtering has the advantage that it works for an arbitrary number of objectives and provides the user with the power to decide how many Pareto-optimal solutions shall be maintained to not exceed memory or running time resources. We first prove new theoretical properties of regret minimizing sets that are crucial for their application to route planning. As one main result, we show that the regret value does not deteriorate if we concatenate paths and combine their (filtered) solutions. This is important for many preprocessing-based techniques, in which this operation frequently occurs, for example, in multi-criteria contraction hierarchies (CH).

We describe how to construct the CH data structure with regret-based filtering as part of the preprocessing step (and thus significantly reduced space consumption), such that CSP and PSP queries can be answered much faster but with almost no loss in solution quality. As computing the solution subset with smallest regret poses an NP-hard problem, we also propose a novel and very efficient divide & conquer heuristic to compute high-quality subsets. In our experimental evaluation, we demonstrate that our new heuristic significantly outperforms the prevailing algorithm in solution quality, with only a small increase in running time. Furthermore, we show that using this heuristic as a subroutine for multi-criteria route planning results in well-performing preprocessing and query algorithms, even for a large number of cost dimensions.

# 2 Regret-Based Route Planning

The main idea is to use regret minimizing subsets to combat the combinatorial explosion of Pareto-optimal paths that is often observed in multi-criteria route planning algorithms. Regret is normally defined with respect to users that want to maximize their utility functions $f_u$. But in the context of route planning, users want to minimize their (perceived) path costs. Thus, we now redefine regret as follows $r_u(S, S') = 1 - \min_{s \in S} f_u(s) / \min_{s \in S'} f_u(s) \in [0, 1]$, where now $f_u$ expresses a penalty function. We remark that all regret minimizing algorithms can easily be adapted to also work with this definition.

## 2.1 Theoretical Bounds

Next, we prove a crucial property of the regret measure, namely that it does not stack when we combine partial solutions. This allows us to guarantee a good output quality when integrating regret-based pruning into various route planning algorithms, even if they need to combine partial solutions frequently. To concisely phrase our result, we introduce the following notations: With $P_{ab}$ we refer to the set of cost vectors of Pareto-optimal paths from $a$ to $b$, and with $P'_{ab}$ to a subset with regret $r_{ab}$. For two sets of cost vectors $P, Q$, we use $P \oplus Q := \{p + q | p \in P, q \in Q\}$ to denote the set of elements derived from pairwise addition.

▶ **Theorem 1.** *Let $P'_{sv}$ and $P'_{vt}$ be path sets with regret $r_{sv}$ and $r_{vt}$ with respect to $P_{sv}$ and $P_{vt}$, then the regret of the combined path set is upper bounded by $r(P_{sv} \oplus P_{vt}, P'_{sv} \oplus P'_{vt}) \leq \max\{r_{sv}, r_{vt}\}$.*

**Proof.** We use $P := P_{sv} \oplus P_{vt}$ and $P' = P'_{sv} \oplus P'_{vt}$ to abbreviate the notation. Let $u \in U$ be any user and $f_u(p) = \alpha^T c(p)$ their penalty function for paths $p$ with cost $c(p)$. Let $\pi := \arg\min_{p \in P} f_u(p)$ denote the preferred path of $u$ in the full path set, and $\pi' := \arg\min_{p \in P'} f_u(p)$ the best one in the subset. Further, we use $\pi_{sv} \in P_{sv}$ and $\pi_{vt} \in P_{vt}$

to denote the two subpaths of $\pi$ with $c(\pi_{sv}) + c(\pi_{vt}) = c(\pi)$. We have $f_u(\pi) = \alpha^T c(\pi) = \alpha^T c(\pi_{sv}) + \alpha^T c(\pi_{vt})$. With $r_{sv} = 1 - q_{sv}$ and $r_{vt} = 1 - q_{vt}$, we know that there exist paths $\pi'_{sv} \in P'_{sv}$ and $\pi'_{vt} \in P'_{vt}$ with $c(\pi_{sv})/c(\pi'_{sv}) \geq q_{sv}$ and $c(\pi_{vt})/c(\pi'_{vt}) \geq q_{vt}$, respectively. Thus, it follows that:

$$f_u(\pi') \leq \alpha^T c(\pi'_{sv}) + \alpha^T c(\pi'_{vt}) \leq \frac{\alpha^T c(\pi_{sv})}{q_{sv}} + \frac{\alpha^T c(\pi_{vt})}{q_{vt}}$$
$$\leq \frac{\alpha^T c(\pi)}{\min\{q_{sv}, q_{vt}\}} = \frac{f_u(\pi)}{\min\{q_{sv}, q_{vt}\}}$$

Accordingly, $f_u(\pi)/f_u(\pi') \geq \min\{q_{sv}, q_{vt}\}$ and therefore $r_u(P, P') \leq 1 - \min\{q_{sv}, q_{vt}\} = \max\{1 - q_{sv}, 1 - q_{vt}\} = \max\{r_{sv}, r_{vt}\}$. As this inequality holds for all users $u \in U$, the theorem follows. ◀

Of course, if we apply further filtering to a set that already was the result of prior filtering, the regret value might increase.

▶ **Lemma 2.** *Let $P, P', P''$ be path sets with $P' \subset P, P'' \subset P'$ and regrets $r(P, P') = 1 - q_1, r(P', P'') = 1 - q_2$, then $r(P, P'') \leq 1 - q_1 q_2$.*

**Proof.** For a user $u \in U$, let the preferred paths from $P, P', P''$ be $\pi, \pi', \pi''$, respectively. It follows that $f_u(\pi) \geq q_1 f_u(\pi')$ and $f_u(\pi') \geq q_2 f_u(\pi'')$. Combining the two inequalities yields $f_u(\pi) \geq q_1 q_2 f_u(\pi'')$. Thus, $f_u(\pi)/f_u(\pi'') \geq q_1 q_2$ and $r_u(P, P'') \leq 1 - q_1 q_2$. As this upper bound applies for all users, the lemma follows. ◀

Below, we will present route planning algorithms that aim to produce concise path sets $P'_{st}$ for a given $s$-$t$-query that induce little regret with respect to the FPS $P_{st}$. The impact on PSP and CSP queries is expressed in the following lemmas.

▶ **Lemma 3.** *For a PSP query from $s$ to $t$ with weight vector $\alpha$, returning $\min_{\pi' \in P'_{st}} \alpha^T c(\pi')$ yields a $\frac{1}{q}$-approximation with $q := 1 - r(P_{st}, P'_{st})$.*

**Proof.** Let $\pi \in P_{st}$ be the path with smallest personalized cost for the weight vector $\alpha$. As the regret for a subset is defined by the most regretful user, we know that $r(P_{st}, P'_{st}) \geq 1 - \frac{\alpha^T c(\pi)}{\alpha^T c(\pi')}$ and thus $\frac{\alpha^T c(\pi)}{\alpha^T c(\pi')} \geq q$. Rearranging this formula, we get $\alpha^T c(\pi') \leq \frac{1}{q} \alpha^T c(\pi)$. ◀

Accordingly, ensuring small regret automatically provides us with an approximation guarantee for PSP queries. The same is not possible for CSP queries, as here the user specifies hard constraints on the accumulated path costs in all but the first dimension. Thus, if we prune any solution from $P_{st}$, the CSP query might become infeasible. However, we can determine an upper bound on the accumulated constraint violation which is sufficient to guarantee a feasible solution.

▶ **Lemma 4.** *For a CSP query from $s$ to $t$ with bounds $B_2, \ldots, B_d$ for which $P_{st}$ contains a feasible path $\pi$, the set $P'_{st}$ contains a path $\pi'$ that obeys $\sum_{i=2}^{d} c_i(\pi') \leq \frac{1}{q} \sum_{i=2}^{d} B_i$ with $q := 1 - r(P_{st}, P'_{st})$.*

**Proof.** The universe $U$ of users $u$ that define the regret value contains all linear combinations of the cost dimensions. We now focus on $\alpha_u = (0, 1, \ldots, 1) \in \mathbb{R}^d_{\geq 0}$. Using this weight vector, we have $f_u(p) = \sum_{i=2}^{d} c_i(p)$ for any path $p$. Accordingly, $r_u(P_{st}, P'_{st}) \geq 1 - \frac{\sum_{i=2}^{d} c_i(\pi)}{\sum_{i=2}^{d} c_i(\pi')}$ where $\sum_{i=2}^{d} c_i(\pi) \leq \sum_{i=2}^{d} B_i$ holds by the feasibility of $\pi$. The statement follows from applying the same rearrangement to the formula as in the proof of Lemma 3. ◀

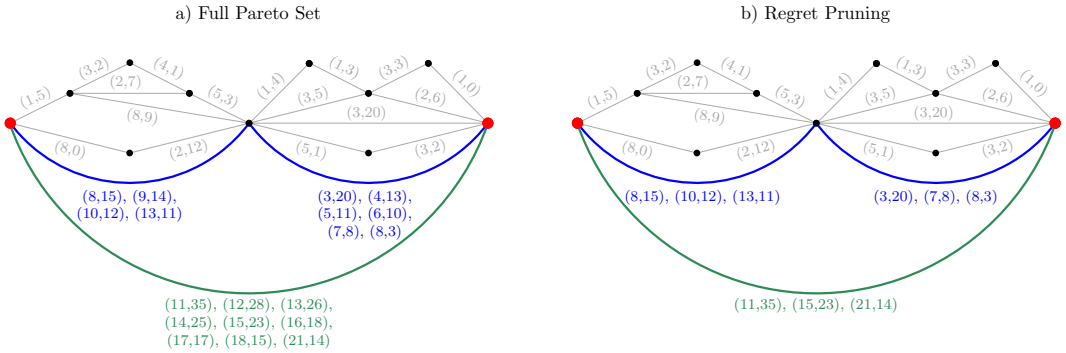


**Figure 2** Example road network (gray edges) with bi-criteria edge costs. The blue and green edges indicate shortcut edges. Left: Each shortcut has a set of cost vectors that encodes the aggregated costs of all Pareto-optimal paths between its endpoints. Right: Applying regret-based pruning to the cost vector sets $S(e)$ per shortcut edge with $k = 3$ results in labels of fixed size.

We remark that for the bi-criteria case, feasible CSP queries can easily be guaranteed by always maintaining the solution with smallest cost in the second dimension in $P'_{st}$.

## 2.2 Regret Minimizing Contraction Hierarchies

Contraction hierarchies (CH) were shown to be a very powerful technique for accelerating bi- and multi-objective route planning queries [9, 13, 12, 28, 16, 6, 3].

In the CH preprocessing phase, nodes are ordered by a ranking function $r : V \to [n]$. Then, so called shortcut edges are inserted between nodes $v, w$ if $r(v) < r(w)$ and there exists at least one Pareto-optimal path from $v$ to $w$ that does not contain a node of rank higher than $r(w)$. The shortcut edge $e = \{v, w\}$ represents all such Pareto-optimal paths from $v$ to $w$ and thus gets assigned the respective set of cost vectors $S(e)$. To obtain this shortcut set $E^+$ efficiently, a bottom-up approach is used. For original edges $e$, the set $S(e)$ initially only contains the cost vector associated with that edge. Then the nodes are considered in the order implied by the ranking and are contracted one-by-one. In the contraction step for node $v$, it is checked for all pairs of neighbors $u, w$ of $v$ whether $S(uv) \oplus S(vw)$ encodes Pareto-optimal paths from $u$ to $w$. If this is the case, a shortcut from $u$ to $w$ is added (if it not already exists) and $S(uw)$ is augmented with the relevant cost vectors from $S(uv) \oplus S(vw)$. Then, $v$ and its incident edges are temporarily deleted from the graph. Note that by construction it holds that after each contraction the Pareto-optimal path costs between the remaining nodes in the graph are the same as in the original graph. After all nodes have been contracted, all temporarily deleted nodes and edges are inserted back into the graph.

In the resulting CH-graph, queries are answered with a bi-directional run of the Pareto-Dijkstra algorithm (or a MOA* variant). Here, forward and backward search only relax edges $(v, w)$ incident to a node $v$ if $r(w) > r(v)$. This significantly reduces the search space but can be proven to nevertheless ensure correct query answering. To not only obtain the cost vectors of Pareto-optimal paths but also the paths themselves, a cost vector $c \in S(uw)$ stores references to $c_1 \in S(uv)$ and $c_2 \in S(vw)$ with $c_1 + c_2 = c$. This allows to recursively unpack a path that contains shortcut edges to a path that consists solely of original edges.

The memory consumption of the CH graph and the query performance crucially depend on the number of shortcut edges and especially the sizes of the cost vector sets $S(e)$ associated with them. A simple approach to reduce the set sizes is to apply regret-based filtering to all $S(e)$ independently with some fixed size upper bound $k$. By virtue of Theorem 1, the regret

for the result of any FPS query is upper bounded by the maximum regret obtained for any shortcut. But if one is interested not only in the path costs but also the paths themselves, the approach is insufficient. This is because we can no longer guarantee that the path unpacking procedure is successful, as for $c \in S(uw)$ with $c_1 + c_2 = c$ and $c_1 \in S(uv), c_2 \in S(vw)$ either $c_1$ or $c_2$ (or both) could have been pruned from their respective sets. Therefore, we use the following bottom-up approach instead: We process the edges/shortcuts $e = \{u, w\}$ ordered by $\min\{r(u), r(w)\}$. For a shortcut $e = \{u, w\}$ we consider all pairs of edges $\{u, v\}, \{v, w\}$ with $r(v) < r(u)$ and construct $S(e)$ as the union of $S(uv) \oplus S(vw)$. Then, we apply regret-based pruning to $S(e)$. In this way, we guarantee that each cost vector that remains in $S'(e)$ is represented by two cost vectors in $S'(uv)$ and $S'(vw)$, respectively. Figure 2 demonstrates the difference of maintaining the Pareto-optimal cost vector sets versus applying regret-based pruning on the set of cost vectors with a fixed size $k = 3$.

## 2.3 Route Planning Queries with Little Regret

For PSP queries there is no need to apply label pruning on query time, as the personalized weight vector reduces the edge costs to scalar values of which only the minimum needs to be maintained. But for FPS and CSP queries, label sizes tend to become huge during the search. A simple way to integrate regret pruning into answering FPS queries is to apply it whenever forward and backward search meet in bi-directional search. For query answering with CH, bi-directional search is the standard and it is also used as a standalone to reduce query time and space consumption [2]. For a node $v$ with a forward label $P_{st}$ and a backward label $P_{vt}$, computing $P_{sv} \oplus P_{vt}$ can be very time consuming if both sets are large. First applying regret-based pruning to both sets to reduce them to a size of $k$, respectively, allows to restrict the number of relevant elements to inspect to $k^2$. According to Theorem 1, the resulting regret is upper bounded by the maximum of the individual regrets.

However, we can also use regret pruning within the forward or backward (or in unidirectional) search as follows. Whenever a node label size $|P_{sv}|$ exceeds a threshold $t$, e.g. $t = 2k$, we apply pruning to reduce the size back to $k$ and then only proceed with the reduced label. Here, the regret might increase as shown in Lemma 2. However, this approach allows us to limit the space consumption of a query to $\mathcal{O}(tn)$ and the cost of an edge relaxation to $\mathcal{O}(t)$.

# 3 Regret Minimization Algorithms

The regret minimizing set problem (RMS) was introduced by [18], inspired by the application of multi-criteria decision making. Presenting a huge set of possibilities to an agent is often infeasible. Therefore, the goal is to select a (small) subset of possible decisions that are representative for the whole set and only communicate those to the agent. The approach is also used for database compression. A recent survey by [25] provides an extensive overview of several aspects of the regret minimization problem.

In [5, 1], RMS was proven to be NP-hard for dimensions $d \geq 3$. Several heuristic and approximation algorithms were designed for RMS that work for arbitrary $d$. The algorithm by [18] greedily adds points to the solution subset until reaching the specified output size. The point selection leverages linear programming. The GeoGreedy algorithm by [19] greedily derives a solution subset based on geometric computations. The Sphere algorithm by [26] relies on sampling utility functions in order to find points with high scores. Similarly, the HittingSet approach introduced by [1] samples utility functions and reduces the RMS problem to a hitting set problem based thereupon. [18] also introduce the Cube algorithm. It partitions the multi-dimensional space into hypercubes by constructing $\lfloor (k - d + 1)^{d-1} \rfloor$

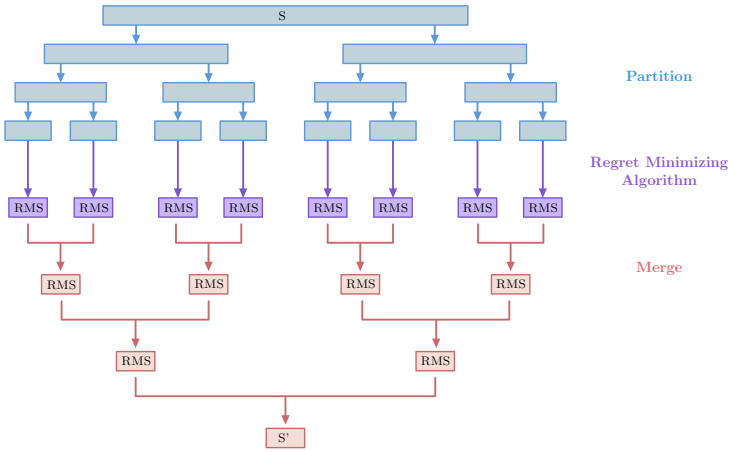


**Figure 3** Overview of the HRMS algorithm for obtaining representative subsets with little regret.

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 $k$ and a regret ratio of at most $\frac{d-1}{t+d-1}$. The running time of the Cube algorithm is in $\mathcal{O}(knd)$ with the space consumption being in $\mathcal{O}(kd+n)$. As this algorithm is easy to implement, scales very well in $d$, and comes with quality guarantees, it is applicable in practice.

## 3.1 Hierarchical Regret Minimization

We propose a novel hierarchical regret minimizing algorithm HRMS that produces representative subsets of size $k$ for arbitrary dimension $d$. The divide and conquer approach partitions the input set into smaller subsets and applies any 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 obtain desired trade-offs between running time and solution quality. Figure 3 illustrates the basic algorithmic concept. Algorithm 1 shows the respective pseudo-code.

Here, $\text{RMS}(P, k')$ is supposed to return a subset of $P$ of size $k'$ with (hopefully) small regret. In our implementation, we use the Cube algorithm [18] for this purpose, due to its simplicity and efficiency even for high dimensions. However, one could plug in any other RMS algorithm as well. $\textsc{Partition}(S, p)$ is supposed to partition $S$ into $p$ subsets, and $\textsc{Merge}(S'_A, S'_B, k')$ to merge two point sets into one and to reduce the resulting set to the desired size. We next discuss sensible implementations of these two important routines.

## 3.2 Partitioning and Merging

The goal of the $\textsc{Partition}$ method is to break the input set $S$ down into sufficiently small subsets of roughly similar size for efficient processing. A very simple partitioning method, is to randomly assign $\frac{n}{p}$ points to each of the $p$ subsets. However, it is advantageous to partition the input set in such way that the smaller subsets cover continuous parts of the multi-dimensional space. A good partition with regards to the HRMS algorithm yields subsets containing points from the Pareto front i.e. non-dominated points together with dominated points. During the

**Algorithm 1** HRMS.

**Input:** $S$, $k$
**Output:** $S' \subset S$ with $|S'| \leq k$
$\mathcal{P}(S) = \text{PARTITION}(S, p)$
$k' = kd$
**for** *each subset* $P \in \mathcal{P}(S)$ **do**
  $S' = \text{RMS}(P, k')$
  Add $S'$ to the list of intermediate sets
**while** $\exists$ *pair of intermediate sets* $S'_A$ *and* $S'_B$ **do**
  $S'_M = \text{MERGE}(S'_A, S'_B, k')$
  Add $S'_M$ to the list of intermediate sets
Define $S'_L$ as the last intermediate set
$S' = \text{RMS}(S'_L, k)$
**return** $S'$

conquer phase of the algorithm we handle each subset separately and reduce the number of overall points during the merging phase. In the worst case, all Pareto-optimal points are in one subset. Then, during the conquer phase we would exclude many such points in favor of dominated points from other subsets. Thus, it is ideal to have Pareto-optimal points in each of the $p$ subsets to ensure that the RMS algorithm is able to retrieve all of them. Our idea to achieve this is to divide the input space into $p$ equally-sized wedges between the x- and y-axes and to assign points to the wedges they are contained in. This likely ensures that each of the wedges covers a wide range of point values and also includes some Pareto-optimal points. Figure 4 depicts an exemplarily partitioning of the input space in $d = 2$ and $d = 3$ dimensions.

The goal of the MERGE operation is to combine two point sets $S'_A$ and $S'_B$ but only keep the best $k$ points among them. A naive approach would again be to just randomly sample $k$ points from the union $S'_A \cup S'_B$. However, this method is unlikely to produce good results. Since we already use an RMS algorithm to obtain the intermediate sets $S'_A$ and $S'_B$, a natural way to merge would be to rerun said RMS algorithm on their union. In practice, this merging technique significantly increases the running time of the HRMS algorithm, though. A third method is to greedily remove points from the union until the desired output size is reached. Ideally, we would like to always select the point whose removal increases the regret the least. But computing the regret ratio requires to solve a linear program [18] and is thus also time intensive. As a compromise, we propose a sorted merge algorithm. It initially sorts $S'_A$ and $S'_B$ decreasingly in each of the $d$ dimensions. Then, the first $\lfloor \frac{k}{2d} \rfloor$ points from the sets $S'_A$ and $S'_B$ corresponding to the decreasing order in the first dimension are retrieved. This procedure is repeated for each dimension to obtain the output set of given size. In this way, we select the promising points in each dimension.

## 3.3 Running Time Analysis

Considering the running time of the HRMS algorithm, we define $t_{\text{PART}}$ as the running time required by the partitioning method, $t_{\text{RMS}}$ as the maximum running time of the chosen RMS algorithm executed on a subset $S'$, and $t_{\text{MERGE}}$ be the running time of the chosen merging technique. With the partition method producing $p$ subsets, the total running time is in $\mathcal{O}(t_{\text{PART}} + p \cdot (t_{\text{RMS}} + t_{\text{MERGE}}))$. Wedge-based partitioning takes $\mathcal{O}(nd)$. The Cube algorithm

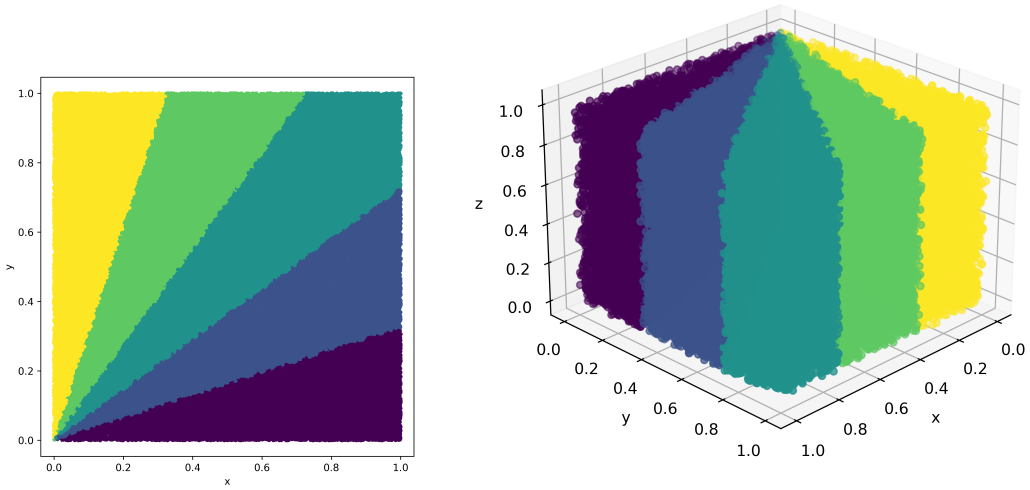


**Figure 4** Exemplary partitioning of uniformly distributed input sets in $d = 2$ and $d = 3$ dimensions. The wedges divide the input space into $k = 5$ subsets based on the angle of the points to the x-axis.

executed on a point subset $P$ with a desired output size of $k' = kd$ takes $\mathcal{O}(k'|P|d)$. Thus, the total time for processing all $p$ subsets $P \in \mathcal{P}(S)$ is in $\mathcal{O}(knd^2)$. The merging calls take $\mathcal{O}(dn \log n)$ to go from $p$ subsets to $p/2$ subsets. Thus, the total merging time is in $\mathcal{O}(dn \log n \log p)$. As we have $p \leq n$, we get an overall running time in $\mathcal{O}(nd(dk + \log^2 n))$ which is close to the theoretical running time of the Cube algorithm. We will see that the two algorithms indeed also have similar practical running times, while HRMS produces sets with significantly smaller regret than Cube.

## 4 Experimental Evaluation

For regret minimizing set computation, we implemented the Cube and the HRMS algorithm in C++. Furthermore, we implemented the multi-objective CH approach with and without regret pruning, as well as the described query answering variants. Experiments were conducted on a single core of a 4.5 GHz AMD Ryzen 9 7950X 16-Core Processor with 188 GB of RAM.

### 4.1 Regret Minimization Results

First, we investigate the performance of our proposed HRMS algorithm and compare it to the Cube baseline. We chose the Cube algorithm since it scales well in arbitrary dimension and allows for a straightforward implementation. For our experiments, we use input sets $S$ consisting of $n$ points in $d$ dimensions and obtain subsets $S' \subset S$ of size of $k = 2d$. Generating the input sets is done by randomly sampling $d$ values from the uniform distribution in the range $[0, nd)$ for each point. Running times and regret ratios are always averaged over 100 generated instances per tested value of $n$.

In order to evaluate the solution quality of the HRMS algorithm, we need to obtain the regret $r(S, S')$ of the solution subset $S'$ which is defined by the most regretful user. To this end, we uniformly sample $1,000$ users represented by their weight vectors $\alpha_u \in [0, 1]^d$. We approximate the regret of the solution subset $S'$ by computing the regret $r_u(S, S')$ for each


**Figure 5** Approximating the maximum regret ratio for a given subset $S' \subseteq S$ is done by randomly sampling $10^2, \ldots, 10^6$ user weight vectors $\alpha_u \in [0,1]^d$ and obtaining the maximum value among the regret ratios of all users in the sample.

sampled user $u$ and extract the maximum among them. A similar approach is done in [1] by randomly sampling 20,000 user weight vectors. We experimentally explore the effect of the sample size of users in the set $U$ on the resulting solution quality in Figure 5. Comparing sample sizes ranging from $|U| = 10^2, \ldots, 10^6$ we conclude that the obtained regret ratios for the solution subsets $S'$ are sufficiently similar among the sample sizes when ensuring that the extreme values 0.0 and 1.0 are represented at least once per dimension. Since computing the regret ratio for a given user weight vector $\alpha_u$ involves traversing the entire input set $S$, we use a sample size of 1,000 to efficiently approximate the regret ratio for our experiments.

Prior to the experiments, we want to evaluate the geometric partitioning with respect to the resulting subset sizes and asses how close this angle-based partitioning comes to an ideal partitioning of the input space. The partitioning divides the input set into $p$ subsets based on the position of points in the $d$-dimensional space. For the statistical assessment in Figure 6 input sizes of $n = 10^2, \ldots, 10^6$ in $d = 2$ dimensions and the output size $k = 4$ and intermediate output size $k' = 8$ are used to obtain $p$ subsets. The value of $p$ is determined such that $\frac{n}{p} > k'd = 16$. Per tested value of $n$, the partitioning described in Section 3.2 is executed on 100 instances. Figure 6 reports on the average distribution of resulting subset sizes with the minimum, maximum and median size of subsets $P \in \mathcal{P}(S)$ in the partition of the input set. Additionally, the number of subsets $p$ is also provided for each input size. An ideal partitioning yields equally-sized subsets each containing exactly $\frac{n}{p}$ points. This value depending on $n$ and $p$ is denoted as *ideal* (red markers).

The difference between the minimum and maximum subset sizes is fairly small considering input sets of up to 1 million points being divided into $p = 2^{15}$ subsets. The smallest subsets contain 2 points and the largest 72 points. However, most observations in the distribution are much closer to the ideal subset size. The interquartile range for all input sizes always encloses the ideal subset size and the $25^{\text{th}}$ percentile is 12 points below the ideal value and the $75^{\text{th}}$ percentile is 20 points above the ideal value. Moreover, the median of actual subset sizes from the partitioning is within 3 points of the ideal value. For $n \geq 1000$ the variation reduces to one point or less. The median is closer to the lower quartile value thus more subsets are slightly smaller than the median subset size.

Overall, the distribution of subset sizes is consistent throughout all input sizes. As the value of $p$ is equal for some input sizes e.g. $n = 6 \cdot 10^5, \ldots, 10^6$, the ideal subset size varies, consequently shifting the box plots to a slightly higher subset size.

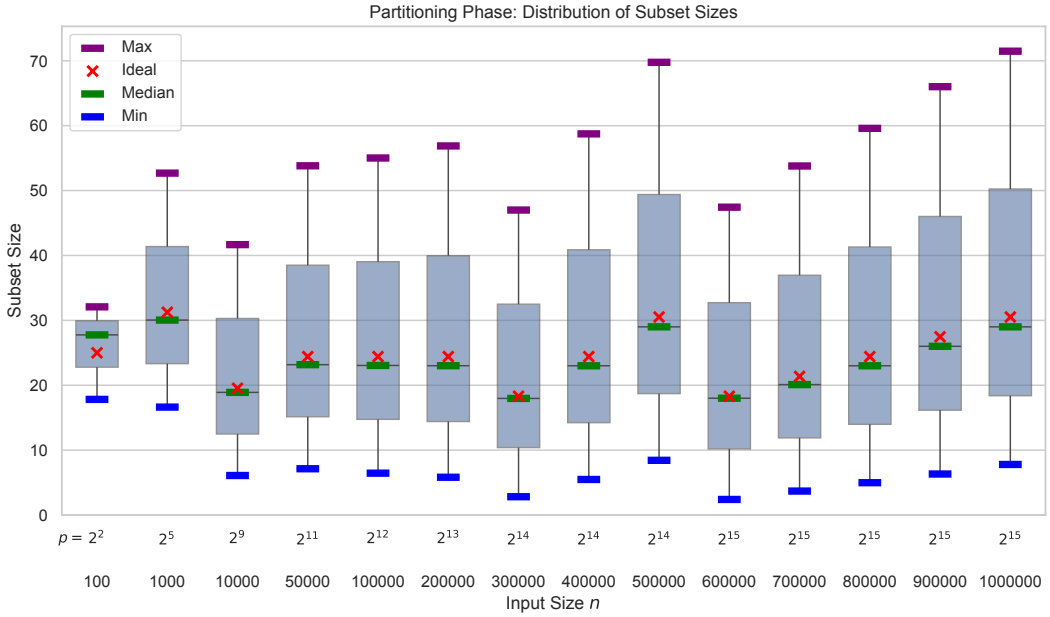


**Figure 6** Statistical evaluation of the subset sizes resulting from the geometric partitioning of the input set into $p$ subsets using the angle of points to the $x$-axis. Depicted are input sizes $n = 10^2, \ldots, 10^6$ in $d = 2$ dimensions using an output size of $k = 4$ and intermediate $k' = 8$. The number of subsets $p$ is determined to ensure $\frac{n}{p} > k'd = 16$. The ideal value of $\frac{n}{p}$ points encapsulates the goal of equally-sized subsets (red markers).

In conclusion, the geometric partitioning based on angles to the $x$-axis produces subsets which are fairly equally-sized and close to the ideal subset size for a given value of $p$. Further, the advantages of dividing the input space geometrically in contrast to a random partitioning prevail as the former ensures an even coverage of the Pareto-optimal points among the subsets.

Figure 7 compares the regret ratios of the HRMS algorithm using the geometric partitioning and the random partitioning. The former divides the input space into equally-sized wedges between the x- and y-axes whereas the latter divides the input set into $p$ subsets using the indices of points within the set. The solution quality resulting from the geometric partitioning is clearly preferable compared to the random partitioning. In $d = 2$ dimensions,

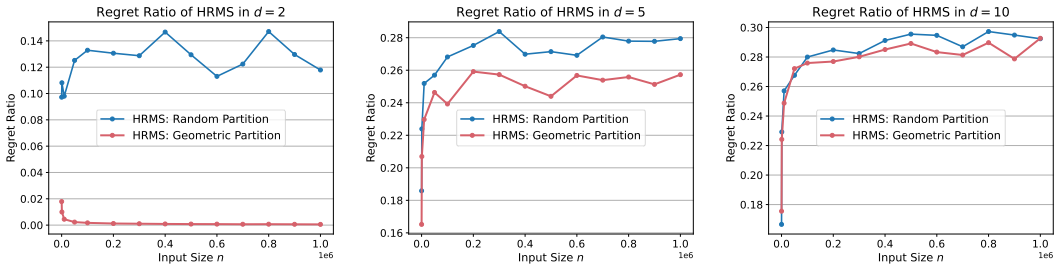


**Figure 7** Comparison of the average regret ratios produced by the HRMS algorithm using the geometric partitioning based on wedges between the x- and y-axes and the random partitioning into $p$ subsets. The input sets in $d = 2, 5, 10$ dimensions consist of $n = 10^2, \ldots, 10^6$ points.

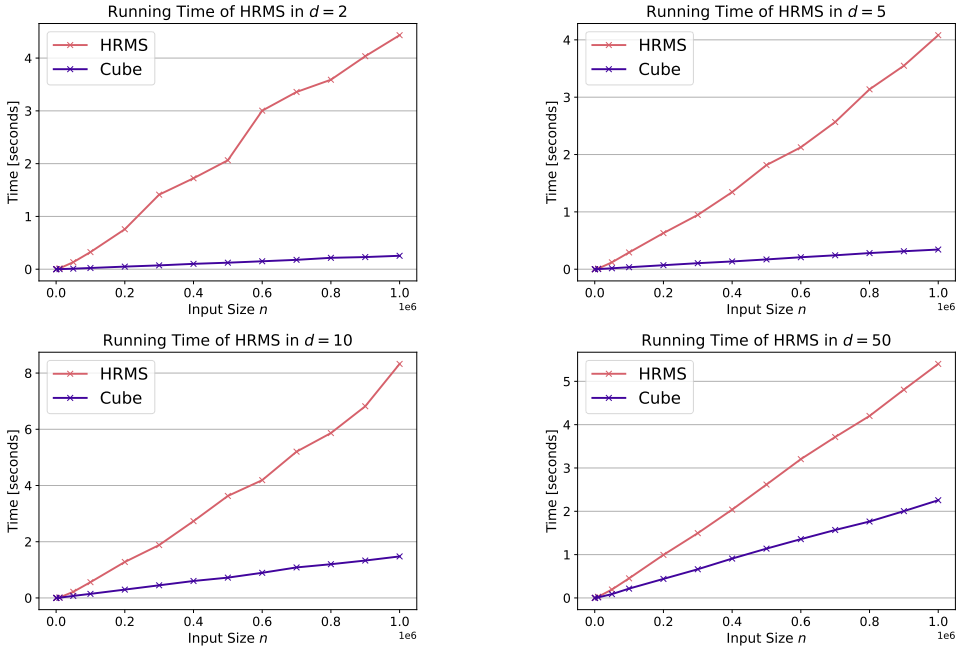


**Figure 8** Average running times for the HRMS algorithm compared to the Cube algorithm on generated input sets of sizes $n = 10^2, \ldots, 10^6$ in $d = 2, 5, 10, 50$ dimensions.

the improvement in regret ratio is up to a factor of 227 on average to the random partitioning. The beneficial behavior of the geometric approach was observed consistently up to $d = 10$ dimensions. Only for higher dimensions both versions of HRMS provide similar regret ratios. Thus, we conclude that the geometric partitioning is indeed preferred over the random partitioning in practice matching the previous theoretical considerations.

Considering the dimensional scalability of the HRMS algorithm, we use input sets of up to $n = 10^6$ points in $d = 2, \ldots, 50$ dimensions with the selection of $d = 2, 5, 10, 50$ being depicted in Figure 8 (running time) and Figure 9 (solution quality). For all tested dimensions, we observe the HRMS algorithm requiring slightly more time than the Cube algorithm executed directly on the input set which is in compliance with our theoretical analysis. However, as the number of dimensions increases, the practical running time of HRMS algorithm gets closer towards the running times of the Cube algorithm. In terms of the solution quality, the HRMS algorithm consistently outperforms the Cube algorithm by providing significantly lower regret ratios among all tested dimensions. The most significant improvement by the HRMS algorithm of up to a factor of 174 on average is obtained in $d = 2$ dimensions compared to Cube. In $d = 5$, $d = 10$ and $d = 50$ dimensions, the average improvement of HRMS compared to Cube is by factors of 1.49, 1.40 and 1.48, respectively. However, the HRMS algorithm optimizes for 50 objectives simultaneously and provides small representing subsets that still ensure that the most regretful user is about 79% happy with the provided solution instead of the full $10^6$ alternatives in the input set. Hence, we conclude that the geometric partitioning together with the hierarchical merging of the HRMS algorithm is consistently advantageous in terms of the solution quality as opposed to the state-of-the-art algorithm Cube.

Further, we evaluate the ratio of Pareto-optimal points in the solution subsets of the HRMS algorithm compared to the Cube algorithm in Figure 10 for $d = 2$ and $d = 3$ dimensions and up to $n = 10^6$ input points. In both dimensions, the HRMS algorithm

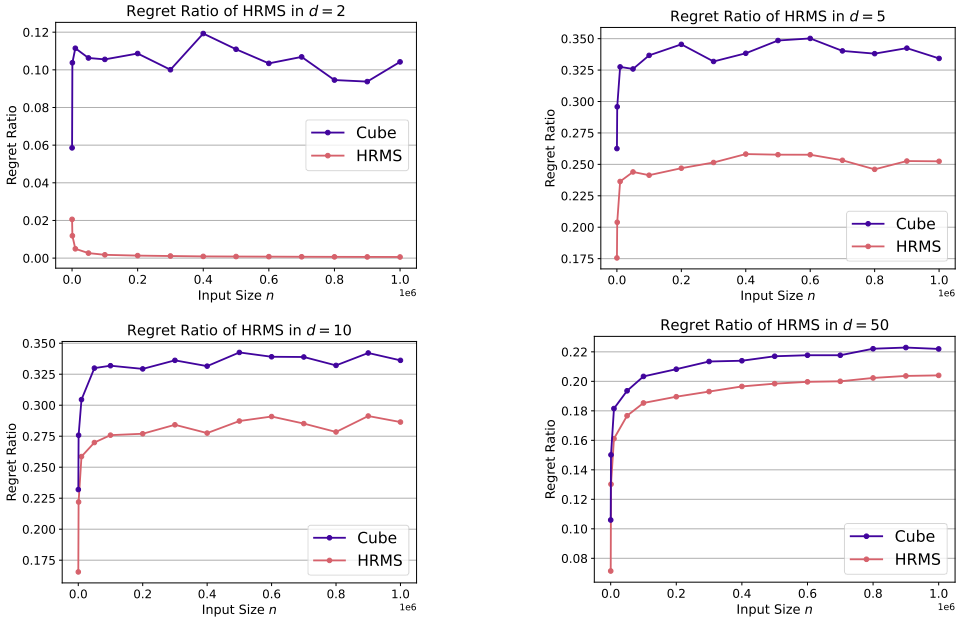


**Figure 9** Average regret ratios for the HRMS algorithm compared to the Cube algorithm on generated input sets of sizes $n = 10^2, \ldots, 10^6$ in $d = 2, 5, 10, 50$ dimensions.

retrieves more Pareto-optimal points in its output set than the Cube algorithm. More precisely, for two dimensions the HRMS provides a Pareto ratio of 87% while the Cube only achieves 70%. In three dimensions, the Pareto ratios are 91% for the HRMS and 77% for the Cube algorithm. Hence, the improvement in solution quality provided by the HRMS algorithm is due to retrieving more Pareto-optimal points from the input set $S$ than the Cube algorithm. During the partitioning phase we emphasize on dividing the input space in such way that each subset ideally contains a small fraction of the Pareto frontier. Further, the sorted merge during the hierarchical merging ensures to select points having high values in each of the dimensions which corresponds to the concept of Pareto-optimality.

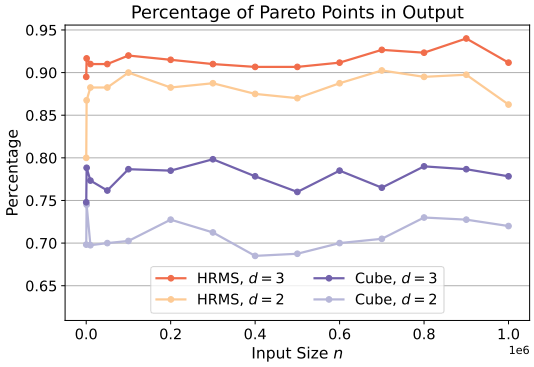


**Figure 10** Comparison of the ratio of Pareto-optimal points in the solution subsets of the HRMS algorithm and the Cube algorithm in $d = 2$ and $d = 3$ dimensions, using input sizes of $n = 10^2, \ldots, 10^6$.

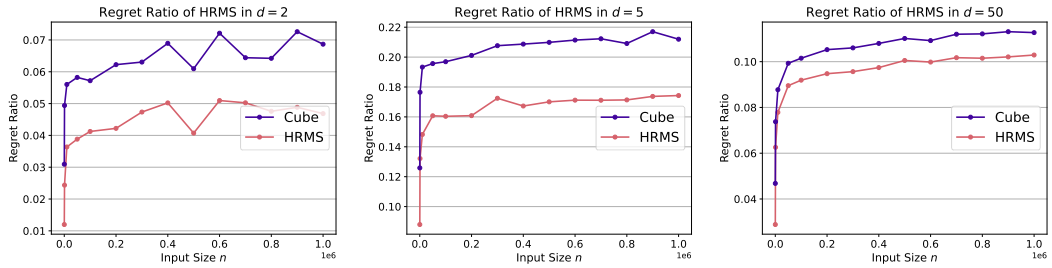


**Figure 11** Gaussian input: Comparison of the regret ratios obtained by the HRMS algorithm and the Cube algorithm on input sets of size $n = 10^2, \ldots, 10^6$ in $d = 2, 5, 50$ dimensions using Gaussian distributed inputs.

We now want to evaluate the HRMS algorithm on input sets following different distributions which are closer to realistic input scenarios. To this end, we execute the algorithms on Gaussian distributed input sets. For the Gaussian generator, we sample $d$ values from the normal distribution with mean $\mu = \frac{nd}{2}$ and standard deviation $\sigma = \frac{\mu}{4}$ ensuring that all values are positive integers. The corresponding regret ratios for the HRMS algorithm are displayed in Figure 11. Again, the HRMS algorithm consistently produces lower regret ratios than the Cube algorithm on all tested dimensions up to 50. On average among the input sizes, we gain a maximal improvement factor of 2.6 in $d = 2$ dimensions compared to the Cube algorithm. Over all dimensions and input sizes, the average improvement of the HRMS algorithm is by a factor of 1.24.

## 4.2 Multi-Objective Route Planning Results

For studying the impact of regret-based filtering in multi-objective route planning, we used an established benchmark, namely the German road network with $d = 10$ real cost dimensions[2] such as the distance, travel time, positive height difference and energy consumption for electric vehicles [12]. Each cost dimension was normalized to the interval $[0, 1]$. To study the scalability of the different methods, we selected cutouts with the number of nodes ranging from 10000 to 3 million.

### 4.2.1 Preprocessing

To allow for a fair comparison between classical multi-objective CH and our proposed variant with regret-based filtering, we first constructed a metric-independent CH-graph, also called Customizable Contraction Hierarchy (CCH) [10]. Here, shortcuts are assigned between all pairs of neighbors when a node is contracted. Afterwards, we use the described bottom-up procedure to equip the edges and shortcuts with sets of cost vectors that encode Pareto-optimal paths between their endpoints. We either use the mode *full* where we only filter dominated points, or the mode *regret* where we additionally use HRMS to reduce the cost vector size to at most $k$. If not stated otherwise, we use $k = 5$.

In Table 1, the statistics of our benchmark instances are provided together with the experimental results for the first preprocessing step, namely the CCH construction. Note that we actually stop after contracting 99.7% of the nodes. Leaving this "core" of uncontracted nodes is a common method for multi-objective CH construction (see e.g. [12]), as the

[2] `https://www.dropbox.com/s/tclrjdkfhabu27h/ger10.zip?dl=0`

**Table 1** Road network instances with number of nodes $n$ and number of edges $m$. The right columns indicate the time for the CCH construction as well as the number $|E^+|$ of inserted shortcuts.

| | properties | | CCH | |
|---|---|---|---|---|
| | $n$ | $m$ | time | $\lvert E^+\rvert$ |
| RN1 | 10000 | 21498 | 17ms | 19958 |
| RN2 | 100000 | 214362 | 0.3s | 207900 |
| RN3 | 1000000 | 2125904 | 3.8s | 2070908 |
| RN4 | 3064263 | 6468394 | 20.2s | 6347312 |

**Table 2** Results for the vector assignment step when maintaining all Pareto-optimal cost vectors (CH full) or with pruning to $k = 5$ vectors (CH regret). The columns *avg* and *max* show the average and maximum number of cost vectors per shortcut in the final graph.

| | CH full | | | CH regret | | |
|---|---|---|---|---|---|---|
| | time | avg | max | time | avg | max |
| RN1 | 0.1s | 1.78 | 606 | 30ms | 1.19 | 5 |
| RN2 | 1.5m | 5.80 | 30709 | 0.4s | 1.23 | 5 |
| RN3 | 2.7h | 7.45 | 161260 | 4.2s | 1.22 | 5 |
| RN4 | - | - | - | 19.9s | 1.21 | 5 |

shortcuts introduced late in the process usually span large portions of the graph and thus usually also encode a huge number of Pareto-optimal paths. Preventing their insertion by stopping early helps to reduce the query time, although the query algorithm has to consider all nodes in the core. We observe that this preprocessing step is very fast and the number of shortcuts that are inserted is less than the number of original edges.

In the second preprocessing step, the edge cost vectors are assigned. We set a timeout of 5 hours for this step. In Table 2, we see that for the *full* setting in which all Pareto-optimal solutions are maintained, we could only stay within that time limit for the graphs with up to 1 million nodes. While the average number of vectors per edge is not huge even for RN3, the maximum number grows quite severely. This not only consumes a lot of space but also increases the preprocessing time. Applying regret pruning, however, allows to perform this step very quickly on all networks as the number of vectors to process per shortcut is limited.

### 4.2.2 Personalized Shortest Path (PSP) queries

Having fewer cost vectors per shortcut also positively affects the query time, as fewer vectors need to be considered when relaxing an edge. We first evaluate PSP queries, where the user inputs source and target as well as a preference vector $\alpha$. We select 100 random source-target-pairs in each graph and choose $d + 2$ different preference vectors for each pair: One random weight vector, one vector with all entries set to $1/d$ (that means, all dimensions are equally important) and the $d$ unit vectors. Therefore, in total we have 1200 test queries per graph in our setup with $d = 10$. As shown in Table 3, the running time of the Dijkstra baseline is in the order of seconds on the larger graphs. Using the full CH approach, query times are significantly reduced as only a small percentage of nodes and edges are considered on query time. But the acceleration also decreases steeply with increasing graph size, from around 940 for RN1 to around 45 for RN3. The main reason is that the average number of vectors that have to be relaxed per edge is quite large for RN3, which increases the query time. Using CH with regret filtering for every shortcut, that ratio is always upper bounded

**Table 3** PSP query times. The ratio denotes the number of cost vectors considered in the query divided by the number of edges that were relaxed in the query. For the Dijkstra baseline, this value is always 1.

| | Dijkstra | CH full | | CH regret | |
|---|---|---|---|---|---|
| | time | time | ratio | time | ratio |
| RN1 | 47ms | 0.05ms | 25.7 | 0.02ms | 3.6 |
| RN2 | 0.5s | 9.17ms | 196.1 | 0.20ms | 4.2 |
| RN3 | 5.3s | 0.12s | 257.3 | 8.14ms | 4.4 |
| RN4 | 17.6s | - | - | 28.59ms | 4.4 |

**Table 4** Observed maximum regret values for varying values of $d$ and $k$ on the RN3 instance using 1200 queries each.

| $d$ | $k=3$ | $k=5$ | $k=10$ |
|---|---|---|---|
| 5 | 0.0409 | 0.0544 | 0.0204 |
| 10 | 0.0625 | 0.0572 | 0.0419 |
| 50 | 0.0638 | 0.0519 | 0.0481 |
| 100 | 0.0749 | 0.0495 | 0.0474 |

by $k$. Note that the ratio of vectors per edge in a query comes indeed close to $k$ and is thus much larger than the average cost vector number per edge which is provided in Table 2. This is due to the fact that shortcuts between nodes with higher contraction rank are more likely to be considered on query time, but these are also the ones which tend to have the most cost vectors. Nevertheless, the cost vector size is drastically reduced compared to CH full and therefore the query answering stays fast even in the larger graphs, with a speed-up of over 600 for RN3 and RN4.

But of course the question is how the result quality is compromised by the regret-based filtering of the solutions. Over all instances, the observed maximum regret was at most 0.0613 and the average an order of magnitude smaller, namely around 0.0029. Thus, the obtained approximation factor (see Lemma 3) for PSP queries is upper bounded by 1.065 in our experiments, and on average solutions were within 2-3% of the optimal cost. This demonstrates that regret pruning allows to get close-to-optimal query results for PSP while significantly reducing the preprocessing and query time, as well as the space consumption.

To further shed light on the interplay of the dimension $d$ and the maximum number of cost vectors $k$ per shortcut, Table 4 shows regret values on the RN3 instance where cost vectors were created with entries u.a.r. in $[0, 1]$. We observe that the maximum regret stays small even if we only preserve $k = 3$ vectors per shortcut. The average regret was again at least one order of magnitude smaller in all scenarios.

### 4.2.3 Constrained Shortest Path (CSP) queries

Finally, we investigate whether regret pruning is also useful in the context of CSP queries, where the user specifies upper bounds on all but the first cost dimension and aims at retrieving the path which obeys these bounds and has minimum cost in the first dimension. To construct sensible queries, we proceed as follows: We first select $d'$ out of the $d$ available cost dimensions randomly. Then, for a source-target-pair, we compute the shortest path cost $c_i$ for all but the first cost dimension individually and set the bound $B_i = c_i \cdot (1 + \varepsilon), i = 2, \ldots, d'$ for some $\varepsilon > 0$. If not stated otherwise, we use $\varepsilon = 0.05$. We consider the following query

**Table 5** CSP query results for different query algorithms and dimensions on the two smallest instances.

| | RN1 | | | RN2 | |
|---|---|---|---|---|---|
| | $d' = 2$ | $d' = 3$ | $d' = 5$ | $d' = 2$ | $d' = 3$ |
| PD | 25.0ms | 100.2ms | 2.7s | 2.1s | 22.0s |
| PDr | 22.1ms | 59.5ms | 0.5s | 0.8s | 2.7s |
| PCHD | 1.2ms | 13.0ms | 0.3s | 16.2ms | 57.3ms |
| PCHDr | 0.2ms | 0.4ms | 1.4ms | 1.3ms | 2.5ms |

algorithms: Pareto-Dijkstra (PD), Pareto-Dijkstra with regret pruning (PDr), Pareto-CH-Dijkstra (PCHD) and Pareto-CH-Dijkstra with regret pruning (PCHDr). For the variants with regret pruning, we applied HRMS whenever the number of labels assigned to a node is equal to or exceeds $2k$ to bring it back down to $k$. We use $k = 10 \cdot d$ in our experiments. Setting $k$ too low can actually be detrimental to faster query answering, as pruning too many labels might lead to additional operations if labels that dominate many others are removed. Thus, it is sensible to choose $k$ proportional to the dimension. Labels are pruned in all algorithms if a cost entry in any dimension exceeds the respective bound. Table 5 summarizes our findings on the RN1 and RN2 instances. In the 100 queries per experiment, we never observed that PD found a feasible solution but the regret-based variants did not. However, the costs in the first dimension were on average 2-5% larger than for the baseline. But this is a small increase compared to the large performance gain. For RN2 with larger $d'$ and RN3 and RN4 the PD(r) baselines were too slow to conduct sufficiently many queries (single queries took hours). But PCHD(r) are applicable. For example, for RN4 and $d' = 2$, query times for PCHD were in the order of several minutes, while PCHDr took at most 5 seconds.

## 5 Conclusions and Future Work

We demonstrated in this paper, that the regret-based pruning of multi-dimensional solution sets can be a very beneficial ingredient in multi-objective route planning, as it helps to significantly reduce query times and space consumption without having a severe impact on the solution quality. The efficiency of our new hierarchical algorithm for computing regret minimizing subsets, which produces solutions of high quality, is crucial in achieving these outcomes.

The proposed methods should be easy to integrate with existing heuristics for multi-objective search as MOA*. Nevertheless, it would be interesting to explore their interplay further and to apply methods as dimensionality reduction, used for faster dominance checks in MOA*, also for regret pruning.

## References

1 Pankaj K. Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. Efficient Algorithms for k-Regret Minimizing Sets. In *16th International Symposium on Experimental Algorithms (SEA 2017)*, pages 7:1–7:23, 2017. doi:10.4230/LIPIcs.SEA.2017.7.

2 Saman Ahmadi, Guido Tack, Daniel D Harabor, and Philip Kilby. Bi-objective search with bi-directional A*. In *Proceedings of the International Symposium on Combinatorial Search*, volume 12, pages 142–144, 2021. doi:10.1609/socs.v12i1.18563.

3 Gernot Veit Batz and Peter Sanders. Time-dependent route planning with generalized objective functions. In *European Symposium on Algorithms*, pages 169–180. Springer, 2012. doi:10.1007/978-3-642-33090-2_16.

4 Moritz Baum, Julian Dibbelt, Dorothea Wagner, and Tobias Zündorf. Modeling and engineering constrained shortest path algorithms for battery electric vehicles. *Transportation Science*, 54(6):1571–1600, 2020. doi:10.1287/trsc.2020.0981.

5 Sean Chester, Alex Thomo, S. Venkatesh, and Sue Whitesides. Computing k-regret minimizing sets. *Proceedings of the VLDB Endowment*, 7(5):389–400, 2014. doi:10.14778/2732269.2732275.

6 Marek Cuchỳ, Jiří Vokřínek, and Michal Jakob. Multi-objective electric vehicle route and charging planning with contraction hierarchies. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 114–122, 2024. doi:10.1609/icaps.v34i1.31467.

7 Daniel Delling, Julian Dibbelt, and Thomas Pajor. Fast and exact public transit routing with restricted pareto sets. In *2019 Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 54–65. SIAM, 2019. doi:10.1137/1.9781611975499.5.

8 Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F Werneck. Computing multimodal journeys in practice. In *International Symposium on Experimental Algorithms*, pages 260–271. Springer, 2013. doi:10.1007/978-3-642-38527-8_24.

9 Daniel Delling and Dorothea Wagner. Pareto paths with sharc. In *International Symposium on Experimental Algorithms*, pages 125–136. Springer, 2009. doi:10.1007/978-3-642-02011-7_13.

10 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *Journal of Experimental Algorithmics (JEA)*, 21:1–49, 2016. doi:10.1145/2886843.

11 Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. In *International Workshop on Experimental and Efficient Algorithms*, pages 347–361. Springer, 2008. doi:10.1007/978-3-540-68552-4_26.

12 Stefan Funke, Sören Laue, and Sabine Storandt. Personal routes with high-dimensional costs and dynamic approximation guarantees. In *16th International Symposium on Experimental Algorithms (SEA 2017)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.SEA.2017.18.

13 Stefan Funke and Sabine Storandt. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In *2013 Proceedings of the Fifteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 41–54. SIAM, 2013. doi:10.1137/1.9781611972931.4.

14 Stefan Funke and Sabine Storandt. Personalized route planning in road networks. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2015. doi:10.1145/2820783.2820830.

15 Boris Goldin and Oren Salzman. Approximate bi-criteria search by efficient representation of subsets of the pareto-optimal frontier. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 149–158, 2021. doi:10.1609/icaps.v31i1.15957.

16 Demian Hespe, Peter Sanders, Sabine Storandt, and Carina Truschel. Pareto sums of pareto sets. In *31st Annual European Symposium on Algorithms (ESA 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.60.

17 Matthias Müller-Hannemann and Mathias Schnee. Finding all attractive train connections by multi-criteria pareto search. In *Algorithmic Methods for Railway Optimization: International Dagstuhl Workshop, Dagstuhl Castle, Germany, June 20-25, 2004, 4th International Workshop, ATMOS 2004, Bergen, Norway, September 16-17, 2004, Revised Selected Papers*, pages 246–263. Springer, 2007. doi:10.1007/978-3-540-74247-0_13.

18 Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J Lipton, and Jun Xu. Regret-minimizing representative databases. *Proceedings of the VLDB Endowment*, 3(1-2):1114–1124, 2010. doi:10.14778/1920841.1920980.

19 Peng Peng and Raymond Chi-Wing Wong. Geometry approach for k-regret query. In *IEEE 30th International Conference on Data Engineering (ICDE 2014)*, pages 772–783, 2014. doi:10.1109/ICDE.2014.6816699.

20 Zhongqiang Ren, Richard Zhan, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Enhanced multi-objective A* using balanced binary search trees. In *Proceedings of the international symposium on combinatorial search*, volume 15, pages 162–170, 2022. doi:10.1609/socs.v15i1.21764.

21 Nicolás Rivera, Jorge A Baier, and Carlos Hernández. Subset approximation of pareto regions with bi-objective a. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10345–10352, 2022. doi:10.1609/aaai.v36i9.21276.

22 Sabine Storandt. Quick and energy-efficient routes: computing constrained shortest paths for electric vehicles. In *Proceedings of the 5th ACM SIGSPATIAL international workshop on computational transportation science*, pages 20–25, 2012. doi:10.1145/2442942.2442947.

23 Sabine Storandt. Route planning for bicycles – exact constrained shortest paths made practical via contraction hierarchy. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, pages 234–242, 2012. doi:10.1609/icaps.v22i1.13495.

24 Carlos Hernández Ulloa, William Yeoh, Jorge A Baier, Han Zhang, Luis Suazo, and Sven Koenig. A simple and fast bi-objective search algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 143–151, 2020. doi:10.1609/icaps.v30i1.6655.

25 Min Xie, Raymond Chi-Wing Wong, and Ashwin 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. doi:10.1007/s00778-019-00570-z.

26 Min Xie, Raymond Chi-Wing Wong, Jian Li, Cheng Long, and Ashwin 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. doi:10.1145/3183713.3196903.

27 Mingzhou Yang, Ruolei Zeng, Arun Sharma, Shunichi Sawamura, William F Northrop, and Shashi Shekhar. Towards pareto-optimality with multi-level bi-objective routing: A summary of results. In *Proceedings of the 17th ACM SIGSPATIAL International Workshop on Computational Transportation Science GenAI and Smart Mobility Session*, pages 36–45, 2024. doi:10.1145/3681772.3698215.

28 Han Zhang, Oren Salzman, Ariel Felner, TK Satish Kumar, Carlos Hernández Ulloa, and Sven Koenig. Efficient multi-query bi-objective search via contraction hierarchies. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 452–461, 2023. doi:10.1609/icaps.v33i1.27225.

29 Han Zhang, Oren Salzman, Ariel Felner, Carlos Hernández Ulloa, and Sven Koenig. A*pex: Efficient anytime approximate multi-objective search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 17, pages 179–187, 2024. doi:10.1609/socs.v17i1.31556.

30 Han Zhang, Oren Salzman, TK Satish Kumar, Ariel Felner, Carlos Hernández Ulloa, and Sven Koenig. A* pex: Efficient approximate multi-objective search on graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pages 394–403, 2022. doi:10.1609/icaps.v32i1.19825.