# Preference-Based Trajectory Clustering – An Application of Geometric Hitting Sets

**Florian Barth**
Universiät Stuttgart, Germany

**Stefan Funke**
Universität Stuttgart, Germany

**Claudius Proissl**
Universität Stuttgart, Germany

## Abstract

In a road network with multicriteria edge costs we consider the problem of computing a minimum number of driving preferences such that a given set of paths/trajectories is optimal under at least one of these preferences. While the exact formulation and solution of this problem appears theoretically hard, we show that in practice one can solve the problem exactly even for non-homeopathic instance sizes of several thousand trajectories in a road network of several million nodes. We also present a parameterized guaranteed-polynomial-time scheme with very good practical performance.

**2012 ACM Subject Classification** Theory of computation → Discrete optimization; Theory of computation → Computational geometry

**Keywords and phrases** Route planning, personalization, computational geometry

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.15

**Supplementary Material** *Software (Source Code and Data)*: `https://doi.org/10.17605/osf.io/4qkuv`

**Funding** This work was in part supported by the Deutsche Forschungsgemeinschaft (DFG) within the priority program 1894: Volunteered Geographic Information: Interpretation, Visualization and Social Computing.

## 1 Introduction

It is well observable in practice that drivers' preferences are not homogeneous. If we have two alternative paths $\pi_1, \pi_2$ between a given source-target pair, characterized by 3 costs/metrics (travel time, distance, and ascent along the route) each, e.g., $c(\pi_1) = (27min, 12km, 150m)$, and $c(\pi_2) = (19min, 18km, 50m)$, there are most likely people who prefer $\pi_1$ over $\pi_2$ and vice versa. In previous works, people have tried to formalize these preferences. The most common model here assumes a linear dependency on the metrics. While typical real-world trajectories are not necessarily optimal in such a model, they are usually decomposable into very few optimal subtrajectories, see, e.g., [4]. This preference model allows for *personalized route planning*, where a routing query not only consists of source and destination but also a weighting of the metrics in the network.

Since this weighting of the metrics – often called *preference* – is hard to specify as a user of such a personalized route planning system, methods have been developed which infer the preferences from paths that the user has traveled before. The larger a set of paths is, though, the less likely it is that a single preference/weighting exists which explains all paths, i.e., for which all paths are optimal. One might, for example, think of different driving styles/preferences when commuting versus leisure trips through the country side. So a natural question to ask is, what is the minimum number of preferences necessary to explain a set of given paths in a road network with multiple metrics on the edges. This can also be


interpreted as a trajectory clustering task where routes are to be classified according to their purpose. In our example, one might be able to differentiate between commute and leisure. Or in another setting, where routes of different drivers are analyzed, one might be able to cluster them into speeders and cruisers depending on the routes they prefer.

The goal of this paper is the formulation of this natural optimization problem in the context of multicriteria routing and investigate the theoretical and practical challenges of solving this problem.

### Related Work

The linear preference model in the navigation context has been used in numerous works, e.g., [13, 11, 12, 9], to allow for personalized route planning services. Fewer papers deal with the inference of personal preferences like [17, 8]. The latter paper is also most related to this work as it introduced preference inference based on a linear programming formulation, which we will also instrument in our approach. Note, though, that while [8] already talked about the problem of minimizing the number of preferences to explain a set of trajectories, no serious attempt at getting optimal or close-to-optimal solutions has been made.

### Our Contribution

In this paper we show how to mathematically formulate the problem of finding the minimum number of preferences to explain a set of given paths in a road network with multiple edge metrics. We investigate both theoretical challenges as well as practical solvability and provide a guaranteed polynomial-time heuristic as well as an exact (but potentially superpolynomial-time) solution. Our experimental results show that for problem instances based on data from the OpenStreetMap project, we can compute optimal preference sets for thousands of paths within few minutes on road networks with several million nodes and edges.

## 2 Preliminaries

Our work is based on a *linear* preference model, i.e., for a given directed graph $G(V, E)$ we have for every edge $e \in E$ a $d$-dimensional cost vector $c(e) \in \mathbb{R}^d$, where $c_1(e), c_2(e), \ldots, c_d(e) \geq 0$ correspond to non-negative quantities like travel time, distance, non-negative ascent, ..., which are to be minimized. A path $\pi = e_1 e_2 \ldots e_k$ in the network then has an associated cost vector $c(\pi) := \sum_{i=1}^{k} c(e_i)$.

A preference to distinguish between different alternative paths is specified by a vector $\alpha \in [0, 1]^d$, $\sum \alpha_i = 1$. For example, $\alpha^T = (0.4, 0.5, 0.1)$ might express that the respective driver does not care much about ascents along the route, but considers travel time and distance similarly important. Alternative paths $\pi_1$ and $\pi_2$ are compared by evaluating the respective scalar products of the cost vectors of the path and the preference, i.e., $c(\pi_1)^T \cdot \alpha$ and $c(\pi_2)^T \cdot \alpha$. Smaller scalar values in our linear model correspond to a preferred alternative. An $st$-path $\pi$ (which is a path with source $s$ and target $t$) is optimal for a fixed preference $\alpha$ if no other $st$-path $\pi'$ exists with $c(\pi')^T \cdot \alpha < c(\pi)^T \cdot \alpha$.

Of course, this linear model is a bold simplification of actual drivers' preferences, yet it has been observed that real-world trajectories can typically be decomposed into very few optimal subtrajectories [4] (significantly less than for a single metric), so this linear model appears to be a reasonable approximation of reality.

## 2.1 Personalized Route Planning

Using the linear model it is easy to allow for *personalized route planning*, i.e., a query does not only consist of a source $s$ and a target $t$, but also specifies a preference $\alpha \in [0,1]^d$. The expected answer to such a query $(s,t,\alpha)$ is a path $\pi$ that is optimal for preference $\alpha$. Dijkstra's algorithm can be instrumented to answer such queries by evaluating the scalar product of the edge cost vector and $\alpha$ when relaxing an edge in the course of the algorithm.

Since Dijkstra's algorithm is not really useful for continent-sized road networks people have come up with speed-up schemes, see [13, 12, 11], yet it appears considerably more difficult to achieve the speed-up factors of respective schemes for the single metric case.

## 2.2 Preference Inference

From a practical point of view, it is very unintuitive (or rather: almost impossible) for a user to actually express his driving preferences as such a vector $\alpha$, even if he is aware of the units of the cost vectors on the edges. Hence it would be very desirable to be able to *infer* his preferences from paths he likes or which he has traveled before. [8] proposed a technique for preference inference, which essentially instruments linear programming to determine an $\alpha$ for which a given path $\pi$ is optimal or certify that none exists. Given an $st$-path $\pi$ in a road network, in principle, their proposed LP has non-negative variables $\alpha_1, \ldots, \alpha_d$ and one constraint for each $st$-path $\pi'$ which states, that "for the $\alpha$ we are after, $\pi'$ should not be preferred over $\pi$". So the LP looks as follows:

$$\begin{array}{rlll} \max & \alpha_1 & & \\ \forall st\text{-paths } \pi' : & \alpha^T(c(\pi) - c(\pi')) & \leq 0 & \textit{optimality constraints} \\ & \alpha_i & \geq 0 & \textit{non-negativity constraints} \\ & \sum \alpha_i = 1 & & \textit{scaling constraint} \end{array}$$

Note, that an objective function is not really necessary, as we only care about feasibility, that is, existence of an $\alpha$ satisfying all constraints. As such, this linear program is of little use, since typically there is an exponential number of $st$-paths, so just writing down the complete LP seems infeasible. Fortunately, due to the equivalence of optimization and separation, it suffices to have an algorithm at hand which – for a given $\alpha$ – decides in polynomial time whether all constraints are fulfilled or if not, provides a violated constraint (such an algorithm is called a *separation oracle*). In our case, this is very straightforward: we simply compute (using, e.g., Dijkstra's algorithm) the optimum $st$-path for a given $\alpha$. If the respective path has better cost (wrt $\alpha$) than $\pi$, we add the respective constraint and resolve the augmented LP for a new $\alpha$, otherwise we have found the desired $\alpha$. Via the Ellipsoid method [14] this approach has polynomial running time, in practice the dual Simplex algorithm has proven to be very efficient.

# 3 Driving Preferences and Geometric Hitting Sets

The approach from Section 2.2 can easily be extended to decide for a *set* of paths (with different source-target pairs) whether there exists a single preference $\alpha$ for which they are optimal (i.e., which explains this route choice). It does not work, though, if different routes for the same source-target pair are part of the input or simply no single preference can

explain all chosen routes. The latter seems quite plausible when considering that one would probably prefer other road types on a leisure trip on the weekend versus the regular commute trip during the week. So the following optimization problem is quite natural to consider:

> Given a set of trajectories $T$ in a multiweighted graph, determine a set $A$ of preferences of minimal cardinality, such that each $\pi \in T$ is optimal with respect to at least one $\alpha \in A$.

We call this problem *preference-based trajectory clustering* (PTC).

For a concrete problem instance from the real world, one might hope that each preference in the set $A$ then corresponds to a driving style like speeder or cruiser. Also note, that a real-world trajectory often is not optimal for a single $\alpha$ (prime example for that would be a round trip), yet, studies like in [4] show that it can typically be decomposed into very few optimal subtrajectories if multiple metrics are available.

In [8], a sweep algorithm is introduced that computes an approximate solution of PTC. It is, however, relatively easy to come up with examples where the result of this sweep algorithm is by a factor of $\Omega(|T|)$ worse than the optimal solution. We aim to improve this result by finding practical ways to solve PTC optimally as well as approximately with better quality guarantees. Our (surprisingly efficient) strategy is to explicitly compute for each trajectory $\pi$ in $T$ the polyhedron of preferences for which $\pi$ is optimal and to translate PTC into a geometric hitting set problem.

Fortunately, the formulation as a linear program as described in 2.2 already provides a way to compute these polyhedra. The constraints in the LP from 2.2 exactly characterize the possible values of $\alpha$ for which one path $\pi$ is optimal. These values are the intersection of half-spaces described by the optimality constraints and the non-negativity constraints of the LP. We call this (convex) intersection *preference polyhedron*. A preference polyhedron $P$ of path $\pi$ is $d-1$ dimensional, where $d$ is the number of metrics. This is because the LP's scaling constraint reduces the dimension by one and we can substitute $\alpha_d$ by $1 - \sum_{i<d} \alpha_i$. In the remainder of this Section, we discuss how to construct preference polyhedra from given paths and reformulate PTC as a minimum geometric hitting set problem.

## 3.1 Exact Polyhedron Construction

A straightforward, yet quite inefficient way of constructing the preference polyhedron for a given path $\pi$ is to actually determine *all* simple $st$-paths and perform the respective half-space intersection. Even when restricting to pareto-optimal paths and real-world networks, their number is typically huge. So we need more efficient ways to construct the preference polyhedron.

### 3.1.1 Boundary Exploration

With the tool of linear programming at hand, one possible way of exploring the preference polyhedron is by repeated invocation of the LP as in Section 2.2 but with varying objective functions. For sake of simplicity let us assume that the preference polyhedron is full (that is, $d-1$[1]) dimensional. We first determine $d-1$ distinct extreme points of the polyhedron to obtain a $d-1$-simplex as first (inner) approximation of the preference polyhedron. We then repeatedly invoke the LP with an objective function corresponding to the normal vectors of

[1] after elimination of $\alpha_d$

the facets of the current approximation of the polyhedron. The outcome is either that the respective facet is part of the final preference polyhedron, or a new extreme point is found which destroys this facet and induces new facets to be investigated later on. If the final preference polyhedron has $f$ facets, this approach clearly requires $O(f)$ invocations of the LP solver (and some effort to maintain the current approximation of the preference polyhedron as the convex hull of the extreme points found so far). While theoretically appealing, in practice this approach is not very competitive due to the overhead of repeated LP solving.

### 3.1.2 Corner Cutting Approach

We developed the following *Corner Cutting Approach (CCA)* which in practice turns out to be extremely efficient to compute for a given path $\pi$ its preference polyhedron, even though we cannot bound its running time in terms of the complexity of the produced polyhedron.

CCA is an iterative algorithm, which computes the feasibility polyhedron via a sequence of half-space intersections. Let $P_\pi$ be the preference polyhedron of path $\pi$. In the $i$-th iteration CCA computes a polyhedron $P_i$ with $P_\pi \subseteq P_i \subseteq P_{i-1}$. $P_0$ is the entire preference space with a number of corners equal to the number of metrics. Each of these corners is initially marked as unchecked.

In iteration $i$ CCA takes one corner $\alpha^{(i)}$ of $P_{i-1}$ that has not been checked before. If no such corner exists, CCA terminates and returns $P_{CCA} := P_{i-1}$. Otherwise, it marks the corner $\alpha^{(i)}$ as checked and computes the optimal path $\pi(\alpha^{(i)})$. If $(\alpha^{(i)})^T \left(c\left(\pi\right) - c\left(\pi(\alpha^{(i)})\right)\right) = 0$, the corner $\alpha^{(i)}$ belongs to $P_\pi$ and there is nothing to do. Otherwise, the constraint $\alpha^T \left(c\left(\pi\right) - c\left(\pi(\alpha^{(i)})\right)\right) \leq 0$ is violated by a part of $P_{i-1}$. We call this part $P'_i$. It is clear that $\alpha^{(i)} \in P'_i$. Finally, the polyhedron $P_i := P_{i-1} \backslash P'_i$ is computed by intersecting $P_{i-1}$ with the half-space $\alpha^T \left(c\left(\pi\right) - c\left(\pi(\alpha^{(i)})\right)\right) \leq 0$. This intersection may introduce new corners, which are marked as unchecked. Afterwards, the next iteration starts.

We prove that CCA indeed computes $P_\pi$. Let $P_{CCA}$ be the output of CCA. We first show that $P_\pi \subseteq P_{CCA}$. $P_\pi \subseteq P_0$ is trivially true. Furthermore, in each iteration $i$ it is clear that $P'_i \cap P_\pi = \emptyset$. Hence, for each iteration $i$, we have $P_\pi \subseteq P_i$ and therefore $P_\pi \subseteq P_{CCA}$. The other direction $P_{CCA} \subseteq P_\pi$ follows from the fact that $P_\pi$ is convex and that all corners of $P_{CCA}$ belong to $P_\pi$ as they are marked as checked.

CCA also terminates for finite graphs. Each optimal path $\pi(\alpha^{(i)})$ can add finitely many corners to $P_i$ only once. Since the number of optimal paths is finite the number of corners to be considered is finite as well.

The great advantage of CCA compared to the boundary exploration approach is the avoidance of the linear programming solver.

## 3.2 Minimum Geometric Hitting Set

Using the preference polyhedra we are armed to rephrase our original problem as a *geometric hitting set (GHS)* problem. In an instance of GHS we typically have geometric objects (possibly overlapping) in space and the goal is to find a set of points (a *hitting set*) of minimal cardinality, such that each of the objects contains at least one point of the hitting set. Figure 1 shows an example of how preference polyhedra of different optimal paths could look like in case of three metrics. In terms of GHS, our PTC problem is equivalent to finding a hitting set for the preference polyhedra of minimum cardinality, and the "hitters" correspond to respective preferences. In Figure 1 we have depicted two feasible hitting sets (white squares and black circles) for this instance. Both solutions are minimal in that no hitter can be removed without breaking feasibility. However, the white squares (in contrast to the black circles) do not describe a minimum solution as one can hit all polyhedra with less points.

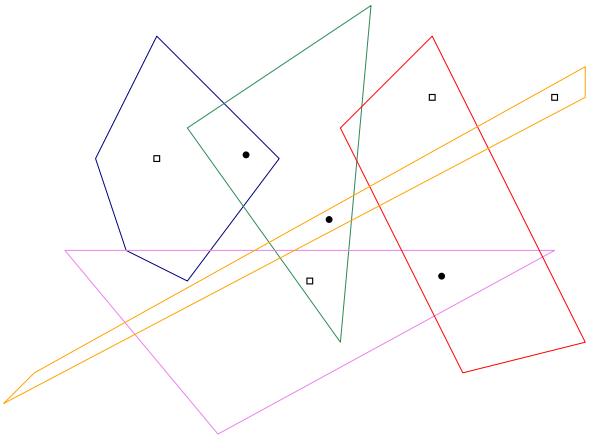

**Figure 1** Example of a geometric hitting set problem as it may occur in the context of PTC. Two feasible hitting sets are shown (white squares and black circles).

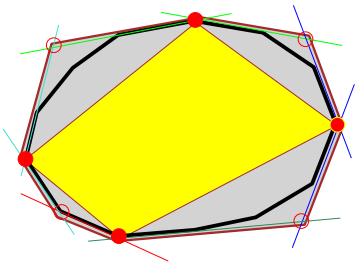

**Figure 2** Inner (yellow) and outer approximation (grey) of the preference polyhedron (black).

While the GHS problem allows to pick arbitrary points as hitters, it is not hard to see that it suffices to restrict to vertices of the polyhedra and intersection points between the polyhedra boundaries, or more precisely vertices in the arrangement of feasibility polyhedra. We describe the computation of these candidates for the hitting sets in Section 3.3.

The GHS instance is then formed in a straightforward manner by having all the hitting set candidates as ground set, and subsets according to containment in respective preference polyhedra. For an exact solution we can formulate the problem as an integer linear program (ILP). Let $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(l)}$ be the hitting set candidates and $\mathcal{U} := \{P_1, P_2, \ldots, P_k\}$ be the set of preference polyhedra. We create a variable $X_i \in 0, 1$ indicating whether $\alpha^{(i)}$ is picked as a hitter and use the following ILP formulation:

$$\min \sum_i X_i$$
$$\forall\, P \in \mathcal{U} : \sum_{\alpha^{(i)} \in P} X_i \geq 1$$
$$\forall\, i : X_i \in \{0, 1\}$$

While solving ILPs is known to be NP-hard, it is often feasible to solve ILPs derived from real-world problem instances even of non-homeopathic size.

## 3.3 Hitting Set Instance Construction via Arrangements of Hyperplanes

To obtain the actual hitting set instance, we overlay the individual preference polyhedra. This can be done via construction of the arrangement of the hyperplanes bounding the preference polyhedra. Each vertex in this arrangement then corresponds to a candidate for the hitting set. If $N$ is the total number of hyperplanes bounding all preference polyhedra in $D$-dimensional space, then this arrangement has complexity $O(N^D)$ and can be computed by a topological sweep within the same time bound [6]. For $K$ polyhedra with overall $N$ bounding hyperplanes we obtain a hitting set instance with $K$ sets and $O(N^D)$ potential hitter candidates.

**(a)** Before inserting new optimal path. **(b)** After inserting new optimal path.

**Figure 3** Example of preference polyhedra of optimal paths with the same source and target and with equal cost in the third metric.

## 3.4 Challenges

While our proposed approach to determine the minimum number of preferences to explain a set of given paths is sound, it raises two major issues. First, the complexity of a single preference polyhedron might be exponential (or just too large for actual computation), so just writing down the geometric hitting set instance becomes infeasible in practice. Second, solving a geometric hitting set instance to optimality is far from trivial. In the following we will briefly discuss these two issues and then in the next section come up with remedies.

### 3.4.1 Preference Polyhedron Complexity

In the following, we show that the complexity of preference polyhedra can be arbitrarily high. We proceed in two steps. In the first step we show that if we find $k$ optimal $st$-paths with two metrics we can construct a preference polyhedron with more than $k$ facets by adding a third metric. In the second step we show that for a single $st$-pair there can be arbitrarily many preference polyhedra with non-zero volume with two metrics.

Let us first assume we have two metrics and $k > 1$ optimal $st$-paths $\pi_1$ to $\pi_k$. Furthermore, we assume that each of the $k$ optimal paths has a preference polyhedron with non-zero volume (a polyhedron is a line in the two-metrics case). We refer to the $j$-th entry of cost vector $c(\pi_i)$ with $c(\pi_i)_j$. Let $x := \max_{1 \leq i \leq k} \; c(\pi_i)_1 + c(\pi_i)_2$ be the maximum of the sums of the cost vectors.

We now introduce a third metric with a constant cost of $3x$ for each path. It is clear that each of the $k$ optimal paths is optimal for the preference $\alpha_3 = [0, 0, 1]$. Hence, each preference polyhedron is now a triangle as shown in Figure 3a.

Finally, we create a new optimal path $\pi_{k+1}$ with the cost vector $c(\pi_{k+1}) := [2x, 2x, 2x]$. This path is clearly optimal for the preference $\alpha_3$. Since $\alpha_3^T \left(c(\pi_{k+1}) - c(\pi_i)\right)$ is strictly less than zero for each $1 \leq i \leq k$ the volume of the preference polyhedron of $\pi_{k+1}$ is non-zero. Furthermore, restricted to the first two metrics $\pi_{k+1}$ is not optimal. This directly follows from the definition of $x$. Hence, the preference polyhedron of $\pi_{k+1}$ shares a constraint with each of the $k$ other optimal paths as shown in Figure 3b.

We now discuss that with two metrics there can be arbitrarily many shortest paths with the same source and target that have a non-zero volume preference polyhedron.

Let us assume that there are $k + 1$ $st$-paths $\pi_0, \pi_1, \ldots, \pi_k$ with cost vectors $c(\pi_i) := \left[i^2, (k-i)^2\right]$. This could be easily realized with one-edge paths. A preference is in this case a tuple $[1 - \alpha, \alpha]$ with $0 \leq \alpha \leq 1$.

For any $0 \leq i < k$ we have $c(\pi_i) - c(\pi_{i+1}) = [-2i-1,\, 2(k-i)-1]$. With $\alpha = \frac{2i+1}{2k}$ we get

$$[1-\alpha,\, \alpha]^T \left(c(\pi_i) - c(\pi_{i+1})\right) = \left[\frac{2(k-i)-1}{2k},\, \frac{2i+1}{2k}\right]^T [-2i-1,\, 2(k-i)-1] = 0$$

Hence, for $0 < i < k$ the path $\pi_i$ is optimal for the range $\alpha \in \left[\frac{2i-1}{2k},\, \frac{2i+1}{2k}\right]$. Path $\pi_0$ is optimal for the range $\alpha \in \left[0,\, \frac{1}{2k}\right]$ and path $\pi_k$ is optimal for the range $\alpha \in \left[\frac{2k-1}{2k},\, 1\right]$.

The question remains whether the number of optimal $st$-paths can be exponential in the graph size. If this is the case, then it follows with the construction above that there are preference polyhedra with exponential complexity.

### 3.4.2 MGHS Hardness

The minimum hitting set problem (or equivalently the set cover problem) in its general form is known to be NP-hard and even hard to approximate substantially better than a $\ln n$ factor, see [2]. A simple greedy algorithm yields a $O(\log n)$ approximation guarantee, which in the general case is about the best one can hope for. For special instances, e.g., when the instance is derived from a geometric setting (as ours), better approximation guarantees can sometimes be achieved. While the exact solution remains still NP-hard even for seemingly simple geometric instances [7], quite strong guarantees can be shown depending on the characteristics of the objects to be hit. Sometimes even PTAS are possible, see, e.g. [16]. Unfortunately, apart from convexity, none of the favourable characterizations seem to be applicable in our case. We cannot even exclude infinite VC dimension in hope for a $O(\log OPT)$ approximation [5], as the preference polyhedra might have almost arbitrarily many corners.

## 4 Polynomial-Time Heuristics with Instance-based Lower Bounds

The previous section suggests that if we require worst-case polynomial running time, we have to resort to approximation of some kind. Both, generation of the geometric hitting set instance as well as solving of the instance might not be possible in polynomial time. We address both issues in this section.

### 4.1 Approximate Instance Generation

It appears difficult to show polynomial bounds on the size of a single preference polyhedron, so approximation with enforced bounded complexity seems a natural approach. There are well-known techniques like coresets [1] that allow arbitrarily (specified by some $\epsilon$) accurate approximation of convex polyhedra in space polynomial in $1/\epsilon$ (which is independent of the complexity of the original polyhedron). We follow the coreset approach and also make use of the special provenance of the polyhedron to be approximated.

For $d$ metrics, our polyhedron lives in $d-1$ dimensions, so we uniformly $\epsilon$-sample the unit $(d-2)$-sphere using $O((1/\epsilon)^{d-2})$ samples. Each of the samples gives rise to an objective function vector for our linear program, we solve each such LP instance to optimality. This determines $O((1/\epsilon)^{d-2})$ extreme points of the polyhedron in equally distributed directions. Obviously, the convex hull of these extreme points is *contained within* and with decreasing $\epsilon$ converges towards the preference polyhedron. Guarantees for the convergence in terms of $\epsilon$ have been proven before, but are not necessary for our (practical) purposes. We call the convex hull of these extreme points the *inner approximation* of the preference polyhedron.

What is interesting in our context is the fact that each extreme point is defined by $d-1$ half-spaces. So we can also consider the set of half-spaces that define the computed extreme points and compute their intersection. Clearly, this half-space intersection *contains* the preference polyhedron. We call this the *outer approximation* of the preference polyhedron.

Let us illustrate our approach for a graph with $d = 3$ metrics, so the preference polyhedron lives in the 2-dimensional plane, see the black polygon/polyhedron in Figure 2. Note that we do not have an explicit representation of this polyhedron but can only probe it via LP optimization calls. To obtain inner and outer approximation we determine the extreme points of this implicitly (via the LP) given polyhedron, by using objective functions $\max \alpha_1, \max \alpha_2, \min \alpha_1, \min \alpha_2$. We obtain the four solid red extreme points. Their convex hull (in yellow) constitutes the inner approximation of the preference polyhedron. Each of the extreme points is defined by 2 constraints (halfplanes supporting the two adjacent edges of the extreme points of the preference polyhedron). In Figure 2, these are the light green, blue, dark green, and cyan pairs of constraints. The half-space intersection of these constraints form the *outer approximation* in gray.

### 4.1.1 Sandwiching the Optimum Hitting Set Size

If – by whatever means – we are able to solve geometric hitting set instances optimally, our inner and outer approximations of the preference polyhedra yield upper and lower bounds to the solution size for the actual preference polyhedra. That is, if for example the optimum hitting set of the instance derived from the *inner* approximations has size 23 and the optimum hitting set of the instance derived from the *outer* approximations has size 17, we know that the optimum hitting set size of the actual exact instance lies between 17 and 23. Furthermore, the solution for the instance based on the inner approximations is also feasible for the actual exact instance. So in this case we would have an instance-based (i.e., not apriori, but only aposteriori) approximation guarantee of $23/17 \approx 1.35$. If for the application at hand this approximation guarantee is not sufficient, we can try improving by refining the inner and outer approximations. In the limit, inner and outer approximations coincide with the exact preference polyhedra.

## 4.2 Approximate Instance Solving

In this section, we discuss approximation algorithms to replace the ILP shown in Section 3.2, which is not guaranteed to run in polynomial time.

The geometric objects to be hit are the preference polyhedra of the given set of paths, the hitter candidates and which polyhedra they hit are computed via the geometric arrangement as described in Section 3.3.

### 4.2.1 Naive Greedy Approach

The standard greedy approach for hitting set iteratively picks the hitter that hits most objects, which have not been hit before. We call this algorithm *Naive Greedy* or short NG. It terminates as soon as all objects have been hit. The approximation factor of this algorithm is $O(\log n)$, where $n$ is the number of objects to be hit, see [15]. The information which preference hits which polyhedron comes from the arrangement described in Section 3.3.

After computing the cover, NG iterates over the picked hitters and removes them if feasibility is not violated. In this way the computed hitting set is guaranteed to be minimal (but not necessarily minimum).

### 4.2.2 LP-guided Greedy Approach

While naive greedy performs quite well in practice, making use of a precomputed optimal solution to the LP relaxation of the ILP formulation from Section 3.2 can improve the quality of the solution. We call this algorithm *LP Greedy* or LPG. It starts with an empty set $S^*$ and iterates over a random permutation of the cover constraints of the LP. Note that each of these constraints $C_i$ corresponds to a preference polyhedron $P_i$.

At each constraint $C_i$ LPG checks if $P_i$ is hit by at least one preference in $S^*$. If not, one of its hitters is randomly picked based on the weights of the LP solution and added to $S^*$. To be more precise, the probability of a hitter $\alpha$ to be drawn is equal to its weight divided by the sum of the weights of all hitters of $P_i$.

After iterating over all constraints it is clear that $S^*$ is a feasible solution. Finally, LPG iterates over $S^*$ in random order removing elements if feasibility is not violated. This ensures that $S^*$ is minimal. This process is repeated several times and the best solution is returned.

## 5 Experimental Results

In this section, we assess how real-world relevant the theoretical challenges of polyhedron complexity and NP-hardness of the GHS problem are, and compare exact solutions to our approximation approaches.

### 5.1 Experimental Setup

We run our experiments on a server with two intel Xeon E5-2630 v2 running Ubuntu Linux 20.04 with 378GB of RAM. The running times reported are wall clock times in seconds. Some parts of our implementation make use of all 24 CPU threads. We cover two different scenarios by extracting different graphs from OpenStreetMap of the German state of Baden-Württemberg. This first graph is a road network with the cost types *distance, travel time for cars* and *travel time for trucks*. It contains about 4M nodes and 9M edges. The second graph represents a network for cyclists with the cost types *distance, height ascent*, and *unsuitability for biking*. The latter metric was created based on the road type (big road $\Rightarrow$ very unsuitable) and bicycle path tagging. The cycling graph has a size of more than double of the road graph with about 11M nodes and 23M edges. The Dijkstra separation oracle was accelerated using a precomputed multi-criteria contraction hierarchy from [9]. For the Sections 5.3 to 5.5, we used a set of 50 preferences chosen u.a.r. per instance and created different quantities of paths with those preferences. We therefore know an apriori upper bound for the size of the optimal hitting set for our instances. In Section 5.6, we show that the number of preferences used does not change the characteristics of our approaches.

### 5.2 Implementation Details

Our implementation consists of multiple parts. The routing and the computation of the (approximate) preference polyhedra of paths is implemented in the rust programming language (compiled with rustc version 1.51) and uses GLPK [3] (version 4.65) as a library for solving LPs. We intentionally refrained from using non-opensource solutions like CPLEX or GUROBI, as they might not be accessible to everyone. The preference polyhedra are processed in a C++ implementation (compiled with g++ version 10.2) which uses the CGAL library [18, 19] (version 5.0.3) to compute the arrangements with exact arithmetic and output the hitting set instances. We transform the hitting set instances into the ILP formulation from Section 3.2

and solve this ILP formulation and its LP relaxation with GLPK. Finally, we implemented the two greedy algorithms described in Section 4.2.1 and 4.2.2 which solve the hitting set instances in C++. Source code and data under `https://doi.org/10.17605/osf.io/4qkuv`.

## 5.3 Geometric Hitting Set Instance Generation

First, we assess the generation of the GHS instances via preference polyhedra construction and computation of the geometric arrangement. In our tables, we refer to the corner cutting approach as "exact" and the approximate approach as "inner-$k$"/ "outer-$k$" where $k$ is the number of directions that were approximated. Since the hitter candidates from the geometric arrangement contain a lot of redundancies (which slows down in particular the (I)LP solving), e.g., some hitters being dominated by others, we prune the resulting candidate set in a straightforward set minimization routine. The preference polyhedra construction as well as the set minimization routine are the multithreaded parts of our implementation. Table 1 shows run times and the average polyhedron complexity for a problem instance with 10,000 paths. In this instance, approximating in twelve directions takes more time than the exact calculation with CCA. This is due the polyhedron complexity being very low in practice. It shows that CCA is a valid approach in practice. Set minimizing is particularly expensive for the inner approximations since considerably more candidates are not dominated.

**Table 1** Statistics about instance generation: average number of polyhedron corners, polyhedra construction time (multithreaded), construction time for arrangement, time for set minimization (multithreaded). Car graph with 10,000 paths. Time in seconds.

| Algo. | Polyh. Compl | Polyh. Time | Arr. Time | SetMin Time |
|---|---|---|---|---|
| Inner-12 | 3.8 | 70.6 | 352.3 | 1450.0 |
| Exact | 4.7 | 54.9 | 356.7 | 414.8 |
| Outer-12 | 4.5 | 70.6 | 361.4 | 473.0 |

**Table 2** Instance generation and solving for various polyhera approximations. Car graph with 1,000 paths. Time in seconds.

| Algo. | Polyh. Time | Arr. Time | ILP Sol. | ILP Time |
|---|---|---|---|---|
| Inner-4 | 6.7 | 4.4 | 110 | >3600 |
| Inner-8 | 8.4 | 5.5 | 50 | >3600 |
| Inner-16 | 11.3 | 4.8 | 45 | 15.5 |
| Inner-32 | 16.5 | 5.0 | 42 | 3.7 |
| Inner-64 | 26.9 | 4.9 | 39 | 1.8 |
| Inner-128 | 48.8 | 5.0 | 36 | 2.3 |
| Exact | 6.5 | 5.2 | 36 | 0.7 |
| Outer-128 | 48.8 | 5.1 | 36 | 0.8 |
| Outer-64 | 26.9 | 5.1 | 36 | 0.8 |
| Outer-32 | 16.5 | 5.0 | 36 | 0.6 |
| Outer-16 | 11.3 | 5.2 | 36 | 1.8 |
| Outer-8 | 8.4 | 5.0 | 36 | 1.6 |
| Outer-4 | 6.7 | 5.9 | 35 | 11.5 |

## 5.4 Geometric Hitting Set Solving

We now compare the two greedy approaches from Section 4.2 with the ILP formulation of Section 3.2. For the ILP solver, we set a time limit of 1 hour after which the computation was aborted and the best found solution (if any) was reported. The results for two instances on the bicycle graph are shown in Table 3. The naive greedy approach has by far the smallest run time but it also reports worse results than the LP-based greedy which was able to find the optimal solution for exact and outer approximations of the two instances. The ILP solver always reports the optimal solution but it might take a very long time to do so. Especially, the inner approximations seem to yield hard GHS instances before they converge to the exact problem instance. A state of art commercial ILP solver might improve run time drastically.

**Table 3** Comparison of GHS solving algorithms on two instances on the bicycle graph. The times (given in seconds) for the LP-Greedy and ILP algorithm include solving the LP-relaxation first.

| Algorithm | Paths | Greedy Solution | Greedy Time | LP-Greedy Solution | LP-Greedy Time | ILP Solution | ILP Time |
|---|---|---|---|---|---|---|---|
| Inner-12 | 5000 | 210 | 4.2 | 181 | 40.8 | – | >3600.0 |
| Exact | 5000 | 54 | 2.9 | 48 | 24.8 | 48 | 68.4 |
| Outer-12 | 5000 | 57 | 3.1 | 48 | 30.5 | 48 | 95.5 |
| Inner-12 | 10000 | 421 | 10.9 | 399 | 144.6 | – | >3600.0 |
| Exact | 10000 | 58 | 7.6 | 50 | 73.5 | 50 | 188.9 |
| Outer-12 | 10000 | 59 | 8.1 | 50 | 81.1 | 50 | 114.6 |

## 5.5 Varying Polyhedron Approximation

In Table 2 we show that increasing the number of directions in the approximation approach makes its results converge to the exact approach. Interestingly, the outer approximation converges faster than the inner approximation. It typically yields good lower bounds already for four approximation directions and higher. In Table 2 it yields a tight lower bound for eight directions while the inner approximation only achieves a tight upper bound with 128 directions.

## 5.6 Dependence on the number of preferences

So far, we have only considered PTC problem instances that were derived from 50 initial preferences. To ensure that the fixed number of preferences does not bias our results, we also ran instances generated with 10, 20, 100 and 1000 preferences. Tables 4 and 5 hold the results for instances with 1,000 paths on both graphs. As expected, we found larger optimal solutions as the number of preferences increased. Although, the solution size did increase only to 53 and 177 for the car and bicycle graph, respectively. We attribute this to the probably bounded inherent complexity of the graphs and cost types used. We also note a slight increase in ILP run times with spikes when computing the solution to the inner approximation as discussed in Section 5.4. Otherwise, there is no performance difference.

# 6 Conclusions

We have exhibited an example for a real-world application where theoretical complexities and hardness do not prevent the computation of optimal results even for non-toy problem instances. The presented method allows the analysis of large trajectory sets as they are, for example, collected within the OpenStreetMap project. Our results suggest that exact solutions are computable in practice, even more so if commercial ILP solvers like CPLEX or GUROBI are employed. From a theoretical point of view, it would be interesting to prove or disprove our guess that exponentially (in the network size) many optimal paths between one source-target pair exist. Another open problem is to find an explanation for the observed average preference polyhedra complexity, which is surprisingly low. On a more abstract level, our approach of approximating the hitting set problem with increasing precision by refining the inner and outer approximations until an optimal solution can be guaranteed could also be viewed as an extension of the framework of *structural filtering* ([10]), where the high-level idea is to certify exactness of possibly error-prone calculations. It could be interesting to apply this not only to the instance generation step but simultaneously also to the hitting set solution process.

**Table 4** Run times and optimal solution of instances generated with varying amount of preferences. The instances each consist of 1.000 paths on the car graph.

| Algorithm | $\alpha$ | Polygon Time | Arrangement Time | Set Minimization Time | ILP Solution | ILP Time |
|---|---|---|---|---|---|---|
| Inner-8 | 10 | 7.9 | 4.8 | 1.2 | 23 | 6.5 |
| Exact | 10 | 5.9 | 4.5 | 0.4 | 10 | 0.3 |
| Outer-8 | 10 | 7.9 | 4.4 | 0.4 | 10 | 0.2 |
| Inner-8 | 20 | 7.8 | 7.7 | 2.8 | 28 | 96.6 |
| Exact | 20 | 5.2 | 6.0 | 0.6 | 17 | 0.8 |
| Outer-8 | 20 | 7.8 | 6.1 | 0.6 | 17 | 0.8 |
| Inner-8 | 100 | 8.3 | 5.5 | 1.7 | 57 | 3601.0 |
| Exact | 100 | 6.3 | 5.8 | 0.6 | 49 | 23.1 |
| Outer-8 | 100 | 8.3 | 5.6 | 0.6 | 49 | 21.1 |
| Inner-8 | 1000 | 8.0 | 5.7 | 2.1 | 61 | 1166.5 |
| Exact | 1000 | 5.9 | 5.3 | 0.5 | 53 | 2.1 |
| Outer-8 | 1000 | 8.0 | 5.2 | 0.5 | 53 | 21.9 |

**Table 5** Run times and optimal solution of instances generated with varying amount of preferences. The instances each consist of 1.000 paths on the bicycle graph.

| Algorithm | $\alpha$ | Polygon Time | Arrangement Time | Set Minimization Time | ILP Solution | ILP Time |
|---|---|---|---|---|---|---|
| Inner-8 | 10 | 47.9 | 3.0 | 0.6 | 45 | 0.9 |
| Exact | 10 | 57.9 | 2.9 | 0.4 | 10 | 0.1 |
| Outer-8 | 10 | 47.9 | 3.0 | 0.5 | 10 | 0.1 |
| Inner-8 | 20 | 42.4 | 2.9 | 0.7 | 69 | 3600.0 |
| Exact | 20 | 47.7 | 3.1 | 0.5 | 20 | 0.3 |
| Outer-8 | 20 | 42.4 | 3.1 | 0.6 | 20 | 0.3 |
| Inner-8 | 100 | 43.8 | 2.0 | 0.3 | 125 | 398.3 |
| Exact | 100 | 50.1 | 2.1 | 0.2 | 85 | 0.2 |
| Outer-8 | 100 | 43.8 | 2.1 | 0.3 | 83 | 0.3 |
| Inner-8 | 1000 | 43.9 | 2.0 | 0.4 | 193 | 1.2 |
| Exact | 1000 | 49.0 | 2.2 | 0.3 | 177 | 1.6 |
| Outer-8 | 1000 | 43.9 | 2.2 | 0.3 | 177 | 3.7 |

## References


1 Pankaj K Agarwal, Sariel Har-Peled, Kasturi R Varadarajan, et al. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.

2 Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for $k$-restrictions. *ACM Trans. Algorithms*, 2(2):153–177, 2006.

3 Andrew Makhorin. GLPK – GNU Project – Free Software Foundation (FSF), 2012. URL: `https://www.gnu.org/software/glpk/glpk.html`.

4 Florian Barth, Stefan Funke, Tobias Skovgaard Jepsen, and Claudius Proissl. Scalable unsupervised multi-criteria trajectory segmentation and driving preference mining. In *BigSpatial@SIGSPATIAL*, pages 6:1–6:10. ACM, 2020.

5 Hervé Brönnimann and Michael T Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

6 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1987.

7 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981.

8 Stefan Funke, Sören Laue, and Sabine Storandt. Deducing individual driving preferences for user-aware navigation. In *SIGSPATIAL/GIS*, pages 14:1–14:9. ACM, 2016.

9 Stefan Funke, Sören Laue, and Sabine Storandt. Personal routes with high-dimensional costs and dynamic approximation guarantees. In *SEA*, volume 75 of *LIPIcs*, pages 18:1–18:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

10 Stefan Funke, Kurt Mehlhorn, and Stefan Näher. Structural filtering: a paradigm for efficient and exact geometric programs. *Comput. Geom.*, 31(3):179–194, 2005.

11 Stefan Funke, André Nusser, and Sabine Storandt. On k-path covers and their applications. *VLDB J.*, 25(1):103–123, 2016.

12 Stefan Funke and Sabine Storandt. Personalized route planning in road networks. In *SIGSPATIAL/GIS*, pages 45:1–45:10. ACM, 2015.

13 Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route planning with flexible objective functions. In *2010 Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 124–137. SIAM, 2010.

14 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method. In *Geometric Algorithms and Combinatorial Optimization*, pages 64–101. Springer, 1993.

15 David S Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

16 Nabil H. Mustafa and Saurabh Ray. PTAS for geometric hitting set problems via local search. In *Symposium on Computational Geometry*, pages 17–22. ACM, 2009.

17 Johannes Oehrlein, Benjamin Niedermann, and Jan-Henrik Haunert. Inferring the parametric weight of a bicriteria routing model from trajectories. In *SIGSPATIAL/GIS*, pages 59:1–59:4. ACM, 2017.

18 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.3 edition, 2020. URL: `https://doc.cgal.org/5.0.3/Manual/packages.html`.

19 Ron Wein, Eric Berberich, Efi Fogel, Dan Halperin, Michael Hemmer, Oren Salzman, and Baruch Zukerman. 2D arrangements. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.3 edition, 2020. URL: `https://doc.cgal.org/5.0.3/Manual/packages.html#PkgArrangementOnSurface2`.