Polynomial-time Construction of Contraction Hierarchies for Multi-criteria Objectives

Stefan Funke * Sabine Storandt†

Abstract

We consider multicriteria shortest path problems and show that contraction hierarchies – a very powerful speed-up technique originally developed for standard shortest path queries in [7] – can be constructed efficiently for the case of arbitrary conic combinations of the edge costs. This extends previous results in [5] which considered only the bicriteria case and *discrete* weights for the objective functions. On the theory side we prove a polynomial time bound for determining whether a path π is part of the lower envelope of all pareto-optimal paths via some polyhedral arguments. Experiments complement these results by showing the practicability of our approach.

1 Introduction

In many routing applications the objective cannot be described sufficiently by only a single weight on the edges. For example, a driver is certainly keen on reaching his destination as quickly as possible, but he might be also interested in keeping the fuel costs low (see Figure 1 for an illustration). Similarly, when planning a bicycle trip one is not only interested in the length of the trip but also wants to avoid steep climbs. Most of the time we have conflicting objectives, that is, minimizing both values (e.g. travel time and fuel costs, or distance and positive height difference) at the same time is impossible. Therefore one either aims for a fair trade-off of both values or searches for the path which minimizes one of the values but does not exceed a given bound on the other.

In the first case we typically ask for the minimum cost path for a *conic combination* of the edge costs, i.e. given a (di)graph $G(V,E)$ and edge weights $c_1, c_2 : E \rightarrow \mathbb{R}^+$ the optimal path $p = s, \cdots, t$ for given $\alpha, \beta \in \mathbb{R}^+$ is the one minimizing $\sum_{e \in p}(\alpha c_1(e) + \beta c_2(e))$. We call this problem the *conic combination shortest path (CCSP)* problem. If the coefficients α, β are known beforehand, the problem reduces to the single edge weight case and all available algorithms and speed-up techniques for this scenario apply as well. If the conic combination is revealed at query time only, still Dijkstra's algorithm can be applied in a straightforward manner. In larger (street) graphs plain Dijkstra is too slow for most applications, though. Hence – like for conventional shortest path computations – developing preprocessing techniques to speed up query times seems worthwhile. For conic combinations of edge costs this was considered first in [5]. They present a variant of the speed-up technique Contraction Hierarchies (CH)[7] to accelerate query processing. Their approach has some limitations, though: They can only compute paths optimal for $\sum_{e \in p}(c_1(e) + \gamma c_2(e))$ with integral γ in a *prespecified* interval $[L, U]$. Of course, choosing the interval large enough the restriction to discrete values appears neglectable. The interval size influences the runtime of the preprocessing, though, hence especially for large street networks U has to be small to allow for reasonable preprocessing times. Moreover the authors combine CH with other approaches like landmarks [9]

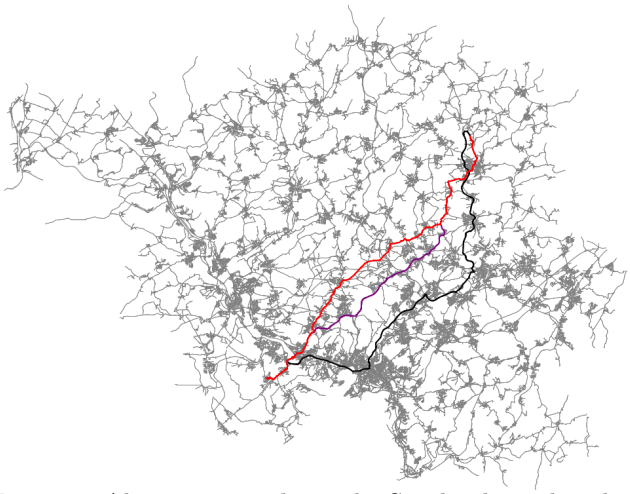

Figure 1: Alternative paths in the Saarland graph: The purple path is optimal in terms of travel time (37min), but has the highest fuel costs (€4.61). The black path has minimal fuel costs (€3.81), but needs time 44min. The red path is a fair trade-off with costs €3.87 and a travel time of 40min.

*FMI, Universität Stuttgart, 70569 Stuttgart, Germany, stefan.funke@fmi.uni-stuttgart.de

†Institut für Informatik, Albert-Ludwigs-Universität Freiburg, 79110 Freiburg, Germany, storandt@informatik.uni-freiburg.de

to achieve the reported overall speed-up. It is unclear how the choice of L and U influences the performance of their combined algorithm. So far, they only consider the combination of *two* edge weights.

In the second case – minimizing one metric while putting a limit on the other – we are faced with an instance of the constrained shortest path (CSP) problem, which is NP-hard in general. An adaption of CH for this scenario was presented in [12]. It was shown that both query time and space consumption can be significantly reduced when employing a variant of CH. While it turns out that even for street graphs with millions of nodes and edges, the optimal solution can be computed within reasonable time (a few seconds), the query times are still far from the millisecond range that can be achieved for ordinary shortest path queries. Hence it is worthwhile to aim for an approximate solution, e.g. by applying binary search [11]. There, a sequence of CCSP queries with varying conic combinations quickly lead to good approximate solutions for the CSP problem. So speeding up CCSP queries provides us automatically with faster algorithms for obtaining approximate CSP solutions.

1.1 Related Work All of the above query variants can be solved by constructing the set of pareto-optimal paths (a path is pareto-optimal, if all other paths are worse according to at least one of the objectives). One approach to compute this set was presented in [4]. The authors call their method Pareto-SHARC, as it is a combination of shortcut insertion (one of the crucial ingredients for CH construction) and arc-flags [10]. Unfortunately, their approach is only practically feasible for a moderate number of pareto-optimal paths between source and target. In general, exploring the complete set of pareto-optimal paths seems not really practical.

Other notions of flexibility were considered e.g. in the context of computing alternative routes [1] or taking into account edge restrictions (like avoiding interstates), [6]. They seem somewhat orthogonal to the view taken in this paper. There exist various speed-up techniques besides CH for the conventional shortest path problem – like transit nodes [3] or reach [8] – that might be useful to accelerate query processing. As shown in [5] and [12], even when considering more than one metric street networks maintain a certain hierarchical structure. Hence there is good reason to hope for improved query times when adapting one of these speed-up approaches also in the multicriteria case.

1.2 Contribution The contributions of this paper are the following: We refine and enhance the adaptation of the CH preprocessing techniques such that not only bicriteria objectives, but also *multicriteria objectives* with arbitrary weight coefficients in $\mathbb{R}^+$ can be considered. More on the theoretical side we prove via some polyhedral considerations that deciding whether a shortcut is necessary can be decided in polynomial time; this is not obvious as the number of paths on the boundary of the convex hull of the pareto-optimal solutions might be exponential in general. Our results are complemented with experimental results which show that our approaches work well also on real-world data.

2 Preliminaries

The Constrained Shortest Path problem (CSP) for two dimensions/metrics is characterized as follows: We are given a (di)graph $G(V, E)$ ($|V| = n, |E| = m$), a cost function $c : E \to \mathbb{R}^+$ and a resource consumption function $r : E \to \mathbb{R}^+$ on the edges. A query is specified by source and target nodes $s, t \in V$ as well as a resource bound $R \in \mathbb{R}^+$. The goal is to determine the minimum cost path from s to t whose resource consumption does not exceed R. Such a path is always pareto-optimal, i.e., no other path p' exists with lower costs *and* lower resource consumption. The set of all pareto-optimal paths between a source and a target node equals the set of all optimal CSP solutions (for different resource bounds).

Exact algorithms like label setting [2] explore the set of pareto optimal paths in search for the best path satisfying the resource constraint – a very time- and space-intensive procedure. In an application context, one is often satisfied with an approximate solution which can be found more quickly. A common idea here is to consider only a subset of all pareto-optimal paths, namely the ones forming the so called *lower envelope*. The lower envelope can be illustrated as follows: For any path p let $c(p)$ be the respective costs and $r(p)$ the resource value. Each tuple $(c(p), r(p))$ can be represented as a line segment

$$\lambda c(p) + (1 - \lambda) r(p),\ \lambda \in [0, 1].$$

An s-t-path p lies on the lower envelope, if there exists a $\lambda \in [0, 1]$ for which $\lambda c(p) + (1-\lambda) r(p)$ is minimal among all s-t-paths (see Figure 2). Paths on the lower envelope have the advantage of being easily computable for fixed λ with a simple Dijkstra run in $G(V, E)$ with edge costs $\lambda c(e) + (1 - \lambda) r(e)$ (we refer to the respective graph also as G^λ). Note that for any conic combination with parameters $\alpha, \beta \in \mathbb{R}$ there exists a value for λ that leads to the same minimum cost path, namely $\lambda = {}^{\alpha}/_{(\alpha+\beta)}$. So the paths on the lower envelope are exactly the ones we are interested in when answering CCSP queries. Another common view of the same problem is in the r-c

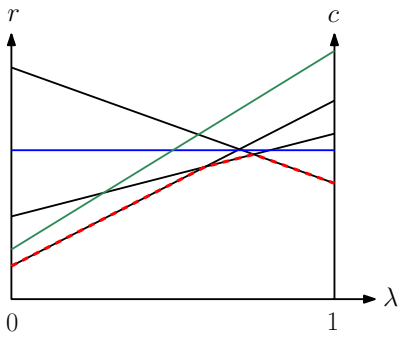

Figure 2: The lower envelope (dashed red) is formed by the three black line segments. The blue line also represents a pareto-optimal path but is not part of the hull. The path corresponding to the green line is not pareto-optimal.

plane where each path is represented by a point in $\mathbb{R}^2$ (according to its resource and cost values) and the paths on the lower envelope are exactly the ones that lie on the lower left boundary of the convex hull of all paths. The binary search algorithm provides approximate CSP solutions by iterating over the slopes on paths on the lower envelope. Here, all edge cost values must be positive, discrete and bounded by some constant M. As shown in [11] this provides a lower bound of $1/(n^2M^2)$ on the difference of any two distinct slopes. Hence the binary search algorithm (see Alg. 1) terminates after $\mathcal{O}(\log(nM))$ rounds which is polynomial in the input.

Algorithm 1 Binary Search

```
s1 ← 0
s2 ← nM
while s2 − s1 > 1/(n²M²) do
  s ← (s2 − s1)/2
  compute optimal path p for edge costs w1 + s · w2
  if r(p) = R then
    break
  end if
  if r(p) < R then
    s2 ← s
  else
    s1 ← s
  end if
end while
```

The CSP problem can be generalized to higher dimensions by assigning multiple resource values to the edges and revealing bounds for each of these on query time. Details for this scenario will be given along with the respective CH construction in Section 4.

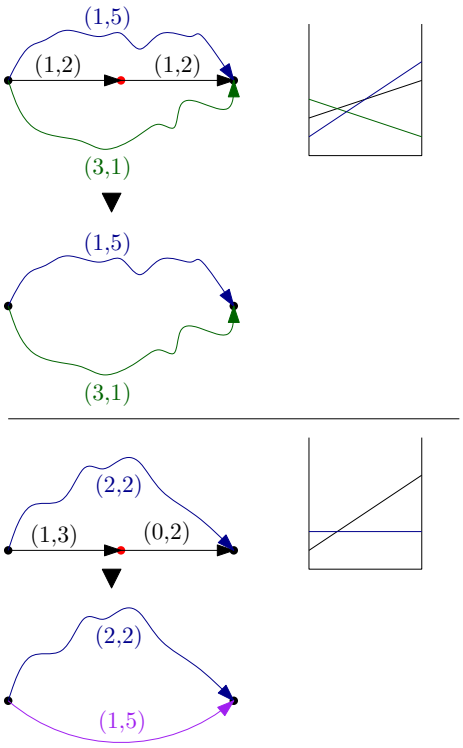

Figure 3: Node contraction step: When removing the red node, all paths on the LE between the remaining nodes have to be preserved. In the upper example the reference path (black) is pareto-optimal, but not on the LE as the blue and the green path form together a witness. Hence the respective shortcut can be omitted. In the lower image the reference path must be shortcutted (realized by the purple edge).

Contraction Hierarchies (CH) This speed-up technique for shortest path computations was introduced in [7]. It is based on the augmentation of the graph with shortcuts that allow to disregard a lot of edges at query time.

To that end, in a preprocessing phase nodes are sorted according to some notion of importance. Afterwards the nodes are removed/contracted one by one in that order, while preserving all shortest path distances in the remaining graph by inserting additional edges (so called shortcuts). More precisely, when contracting a node v for every path uvw the distance from u to w must remain the same even after contraction of v. Hence the edge $e = (u, w)$ has to be added if the only shortest path from u to w is uvw. The cost of e equals the chained costs of the edges (u, v) and (v, w). If there exists a path with cost less than the cost of uvw – a so called *witness path* – the shortcut can be omitted. Such a witness path is typically found via a Dijkstra run from u to w. After all nodes have been removed, a new graph G' is created consisting of all nodes and edges of the original graph and all shortcuts. An edge $e = (v, w)$ (original or shortcut) is called upwards if the importance of v is smaller than that of w and downwards otherwise. In G' for every pair of nodes s and t there exists a shortest path from s to t that can be subdivided into an upward and a downward path (in that order). Therefore s-t-queries can be answered bidirectionally, with the forward run (starting at s) considering only upward outgoing edges and the backward run (starting at t) considering exclusively downward incoming edges.

Note that the contraction process does not have to be completed to the end to ensure correctness. Instead we can abort the process at any point in time and assign an importance value of ∞ to all uncontracted nodes. Declaring all edges between such uncontracted nodes both up- and downwards maintains the upwards-downwards characteristic of optimal paths and allows for the application of bidirectional Dijkstra in the CH-graph. Incomplete contraction is often used to limit the total number of inserted shortcuts and will come in handy for our application on real-world data later on.

For our scenario with conic combinations of the metrics, we have to make sure that during the contraction process *all* paths on the lower envelope are preserved (see Figure 3 for two small examples). In the following we will provide efficient witness search procedures for $d \geq 2$ with d denoting the number of edge weights to consider.

3 The 2-dimensional Case

In this section we present a CH construction scheme that allows for answering CCSP queries with two metrics on each edge. In particular this means that a reference path $\pi = u, v, w$ with costs $c(p)$ and resource value $r(p)$ has to be preserved (by a shortcut), if there exists a λ-value in $[0, 1]$ for which π has minimal costs in G^λ. Therefore the only way to omit the shortcut is identifying a set of alternative $u - w$-paths $p_1, \cdots, p_t$ with their lower envelope being below π. A naive way to search for such a family of witnesses is selecting a set of λ-values (e.g. randomly or equally distributed over $[0, 1]$) $\lambda_1, \cdots, \lambda_t$, compute the respective optimal paths p_λ in G^λ via a single Dijkstra run each and check if the resulting collection of paths expells π from the lower envelope. While being easy to implement, this approach suffers from several disadvantages: On one hand many λ support values might lead to the same path, hence spending time on redundant computations. On the other hand – and much more severe – if the lower envelope of $p_1, \cdots, p_t$ is *not* below π, we don't know whether the shortcut is necessary or can be omitted. To guarantee exact query results no essential shortcut can be left out, hence in that case the shortcut has to be inserted or additional λ-values have to be tested. Adding too many (superfluous) shortcuts slows down both preprocessing time and query processing later on, hence should be avoided if possible. Requesting new λ-paths as long as the result is inconclusive might become very time intensive. We now describe an approach, that overcomes both of these disadvantages. In fact our witness search procedure also relies on choosing some suitable λ values, but we will describe a way to select these values carefully such that

- every new λ-value leads to the discovery of a *new path* on the lower envelope,
- after termination we can *certify* whether the shortcut is needed or not,
- the overall runtime is *polynomial* in the input size.

3.1 Witness Search

The basic idea of our approach is somewhat similar to the binary search algorithm, as in every step we will at least halve the interval for the relevant parameter. However in contrast to conventional binary search, we don't want to compute a certain path on the lower envelope (LE) but want to check if our reference path $\pi = u, v, w$ is part of the LE or not. To that end we start with $\lambda = 0.5$ and compute the respective optimal path $p_{0.5}$. If $\pi = p_{0.5}$ we know that π is contained in the LE and therefore the shortcut is necessary. If instead $p_{0.5}$ dominates π, we found a witness consisting of a single path. Hence the shortcut can be omitted for sure. In both of these cases the witness search is already conclusive and we are done. If neither of the cases applies, there must exist an

intersection point of the line segments representing π and $p_{0.5}$ for some $\lambda \in [0,1]$. Accordingly we can split the interval $I = [0,1]$ in two continuous subintervals I_1, I_2 with I_1 representing the λ-values for which the segment of $p_{0.5}$ lies below the one of π and vice versa in I_2. For any $\lambda \in I_1$ we already know that π will not be the minimum cost path, as $p_{0.5}$ dominates π in this interval. Hence it is sufficient to continue the witness search in I_2 only. Observe that I_2 cannot cover more than the half of I, because at $\lambda = 0.5$ the path $p_{0.5}$ is below π for sure. Now we can repeat this procedure by selecting the new λ support point as the center of the interval I_2, see the pseudocode in Algorithm 2. So the procedure always maintains an interval in which the reference path is still 'alive', i.e. is below all previously identified paths. If this interval runs empty the shortcut is not necessary because π is not part of the LE. If $\pi \in$LE, the search will go on until we find a λ value that reveals this (see Figure 4 for an example). Therefore our search procedure always provides a conclusive result.

Algorithm 2 Witness Search

input: $\pi = u, v, w$ reference path
output: TRUE or FALSE
(insert shortcut (u, w) or not)

$low \leftarrow 0$
$upp \leftarrow 1$
while TRUE **do**
  $\lambda \leftarrow (upp + low)/2$

  get optimal $\lambda - path$ $p_\lambda = u, \cdots, w$
  via a Dijkstra run in G^λ

  if $\pi = p_\lambda$ **then**
    return TRUE
  end if
  if π is dominated by p_λ **then**
    return FALSE
  end if
  $\lambda' \leftarrow \dfrac{r(p_\lambda) - r(\pi)}{c(\pi) - r(\pi) - c(p_\lambda) + r(p_\lambda)}$
  if $\lambda' \notin [low, upp]$ **then**
    return FALSE
  end if
  if $r(\pi) > r(p_\lambda)$ **then**
    $low \leftarrow \lambda'$
  else
    $upp \leftarrow \lambda'$
  end if
end while

It remains to show that our witness search algorithm terminates in a polynomial number of iterations. For that purpose we prove that two distinct corner points of the LE (i.e. intersection points of paths on

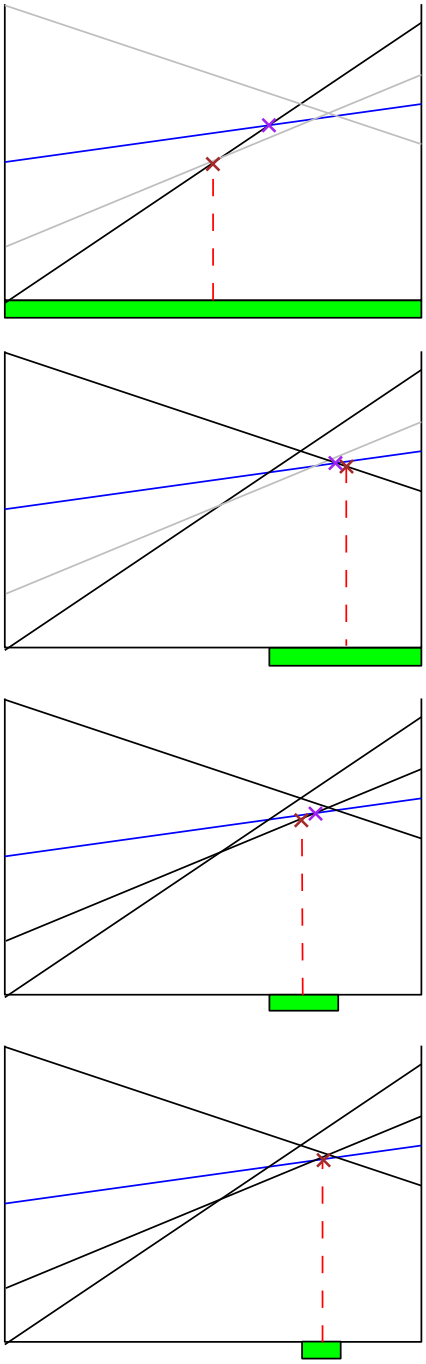

Figure 4: Example for our witness search procedure. The reference path π is given in blue. The green box indicates the remaining interval in which the blue segment might still be part of the lower envelope (LE). The brown cross denotes which path on the LE is optimal for the actual λ-value (red dashed line). The purple cross represents the intersection point with π and therefore the new interval boundary. In the last image the blue path is identified as optimal and therefore proven to be on the LE.

the LE) have a certain minimal distance. Therefore we can abort the process (and insert the shortcut) as soon as the interval covers a range smaller than twice this distance. Like for the binary search algorithm, we assume that all cost/ resource values are positive integers and bounded by $M = \max_{e \in E} \max(c(e), r(e))$. Let p be a path on the LE and p', p'' two other paths on the LE which both form corner points with p. The line (segment) representing p (and analogue for p', p'') can be described by

$$y(\lambda) = \lambda(c(p)) - r(p)) + r(p).$$

Accordingly the λ-values for the intersection points are given by

$$\lambda = \frac{r(p') - r(p)}{c(p) - r(p) - c(p') + r(p')}$$

(similar for p''). As for p'' the cost and the resource value must differ by at least $\delta, \varepsilon \geq 1$ from the values corresponding to p' (as they are integers), the difference of the two λ values can be expressed as

$$\frac{\pm\varepsilon\omega + (\pm\delta \pm \varepsilon)(r(p') - r(p))}{\omega^2 + (\pm\delta \pm \varepsilon) \cdot \omega}$$

with

$$\omega = c(p) - r(p) - c(p') + r(p').$$

Clearly the numerator must be ≥ 1, while the denominator is bounded in $\mathcal{O}(M^2 n^2)$ as Mn represents the maximal possible cost /resource consumption of a path. Hence the difference between the λ-values of any two corners is bounded in $\Omega\left(\frac{1}{n^2 M^2}\right)$. As the interval in which the reference path π is 'alive' gets at least halved in every iteration, we will reach this lower bound in a logarithmic number of rounds. As each round requires the computational effort of a Dijkstra run, we end up with a total runtime of $\mathcal{O}(\log(nM)(n \log n + m))$.

4 From Bicriteria to Multicriteria Shortest Paths

In the general, multicriteria constrained shortest path (CSP) problem each edge e bears a *cost* c_e and resource consumptions $r_e^1, r_e^2, \ldots, r_e^{d-1}$; the CSP problem is to find for given source and target s, t a path $\pi = v_0 v_1 \ldots v_k$ with $v_0 = s$ to $v_k = t$ and edges $e_i = (v_i, v_{i+1})$ which minimizes the cost $\sum_{i=0}^{k-1} c_{e_i}$ of the path while satisfying $d-1$ resource constraints:

$$\sum_{i=0}^{k-1} r_{e_i}^j \leq R_j$$

that is, the added resource consumptions of the jth resource along the path should not exceed R_j.

The CSP problem is known to be NP-hard in general, and known exact solutions (like the standard dynamic programming approach) essentially optimize over all *pareto-optimal* paths. Not surprisingly, only pseudo-polynomial bounds are known for the number of pareto-optimal paths that need to be examined.

For that reason we drop the hard resource constraints and instead optimize a conic combination of cost and resource consumptions. This boils down to optimizing over a subset of all pareto-optimal solutions – namely the solutions on the boundary of the convex hull of all solutions (in $\mathbb{R}^d$ where each path is represented by its cost and resource consumptions). A path $\pi = e_1, e_2, \ldots e_k$ is on the boundary of the convex hull of all pareto-optimal solutions if there exist $\lambda_1, \ldots \lambda_{d-1}$ with $\sum \lambda_i \leq 1$ such that

$$\sum_{e \in \pi} \text{cost}(e) \leq \sum_{e \in \pi'} \text{cost}(e)$$

for all other paths π'. Here

$$\text{cost}(e) = (1 - \sum_i \lambda_i) c_e + \sum_i \lambda_i r_e^i$$

represents the *aggregated weight* of the edge costs and resource consumptions for a conic combination given by the λ_i's. Intuitively speaking, a path π is on the boundary of the convex hull of all pareto-optimal paths if there exist λ_i's such that it is optimal under this aggregated weighting of edge cost and resource consumption.

In the respective dual view, each path π with cost c_π and respective resource consumptions r_π^i, $i = 1, \ldots, (d-1)$ corresponds to a hyperplane

$$h_\pi : y = (1 - \sum \lambda_i) c_\pi + \sum (\lambda_i r_\pi^i)$$

in $\mathbb{R}^d$. For example for $d = 3$ a path π with cost c_π and resource consumptions r_π^1, r_π^2 determines a plane

$$h_\pi : y = (1 - \lambda_1 - \lambda_2) c_\pi + \lambda_1 r_\pi^1 + \lambda r_\pi^2$$

in $\mathbb{R}^3$. When does a path π lie on the boundary of the convex hull of the pareto-optimal solutions? Consider all possible paths from s to t and their respective hyperplanes. A path π lies on the boundary of the convex hull of the pareto-optimal solutions iff its respective plane bounds the *lower envelope* of the hyperplanes (in fact the two view considered here are dual to each other).

In the following we will device a method to decide in *polynomial time* whether the hyperplane corresponding

to a given path π bounds the lower envelope of the hyperplanes of all paths. Note that this seems non-trivial as in general superpolynomially many paths might appear on the boundary of the lower envelope.

The high-level idea of our proof is as follows: we first show that vertices in this arrangement of hyperplanes have a certain minimum pairwise distance. This can be used to derive a lower bound on the hypervolume of facets of the lower envelope. As our algorithm decreases the hypervolume of the facet corresponding to the path π on the lower convex hull by a constant factor in each round, polynomially many rounds suffice to decide whether π bounds the lower envelope.

4.1 Bounding facet hypervolumes in the arrangement of hyperplanes Let us assume that both cost c_e as well as resource consumptions r_e^j of an edge are integers in the range $[0, \dots, M]$.

We first want to show that *vertices* (i.e. points of intersection of $\geq d$ hyperplanes) in this arrangement of hyperplanes have a certain minimum distance. A vertex is the solution of a system $Ax = b$ of d linear equations corresponding to the respective hyperplanes. The i-th coordinate of such a vertex can be computed according to Cramer's rule as $x_i = \frac{\det(A_i)}{\det(A)}$ where A_i is formed by replacing the i-th column vector of A by b. Under the assumption that costs and resource consumptions are bounded by M for each edge, the respective costs/consumptions for a given path are bounded by nM. Hence, if two vertices differ in the i-th coordinate, they differ by at least $\Omega\left(\frac{1}{(nM)^d}\right)$.

A non-degenerate, $d-1$ dimensional facet of the lower envelope hence has a hypervolume of $\Omega\left(\frac{1}{(nM)^{(d-1)^2}\cdot(d-1)!}\right)$. Our goal is to check whether a $d-1$-dimensional hyperplane corresponding to a path π supports a facet of the lower envelope. The facet h_π corresponding to π can have hypervolume at most $O(nM)$ (ignoring all other paths/hyperplanes) as the λ_i are restricted to $[0, 1]$. So if we can guarantee that the hypervolume of the facet h_π decreases by a constant factor $1/\alpha$ in each round of our algorithm, $O(\log_\alpha(nM)^{d^2}(d-1)!) = O(d^2 \log_\alpha(nMd))$ rounds suffice. We will exhibit how to decrease the hypervolume of the respective facet by this constant factor in the following.

4.2 Cutting facets into not too large pieces Given the facet f_π corresponding to path π in the lower envelope of a subset of all paths, we want to certify that either (a) f_π appears on the lower envelope even when all other paths are considered or (b) find a path π' whose corresponding hyperplane cuts off at least a constant fraction of f_π's hypervolume. Certifying (a) is naturally achieved by exhibiting λ_i's which make π an optimal path under the respective aggregated weight. Certifying (b) could for example achieved by choosing appropriate λ_i's such that the hyperplane of the path corresponding to that choice of the λ_i's cuts f_π into two pieces of comparable size.

In the following we will work towards this goal in several steps; first we show that any $d-1$ dimensional facet contains a simplex of comparable hypervolume. Then we choose λ_i''s such that the hyperplane of the optimum path for these λ_i' divides the inscribed simplex into two pieces, none of which is too small.

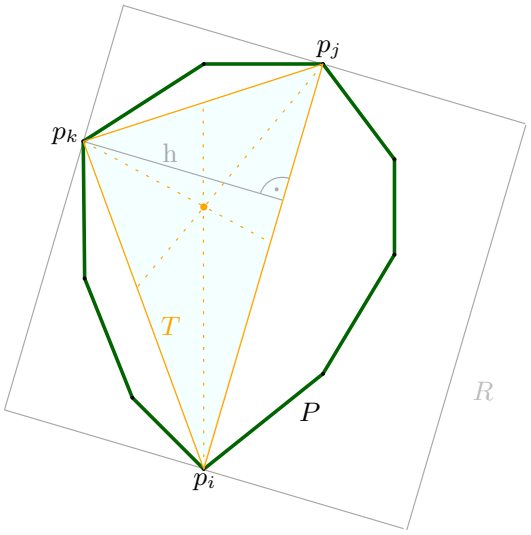

Figure 5: Illustration of Theorem 4.1 for $d = 3$. In this image p_i, p_j are the corner points of the convex polygon P with maximum pairwise distance. The point p_k is the corner point of P_1 which has a maximum orthogonal distance towards $\overline{p_i p_j}$. The gray rectangle R contains P, the triangle T has area exactly $1/4$ of R and hence also at least $1/4$ of the area of P. Combined with Theorem 4.2 we can guarantee that any half-plane with the centroid of the triangle (orange dot) on the boundary contains at least a third of the triangle's area and $1/12$ of the area of P.

4.2.1 Inscribing a Simplex in a Polytope Given a $d-1$-dimensional polytope $P = p_1, \cdots, p_k$, we aim for inscribing a simplex S with a hypervolume of at least a constant fraction of the hypervolume of P. To that end we construct not only a simplex $S \subseteq P$ but also a $(d-1)$-orthotope (a higher dimensional rectangular box) R such that $P \subseteq R$ and $HV(S) = \Omega(HV(R))$ which implies that $HV(S) = \Omega(HV(P))$ by the following

greedy approach:

Let q_1, q_2 be the two vertices of P of maximum pairwise distance. They define a 2-dimensional simplex $S^{(2)}$ and two walls of R namely the hyperplanes orthogonal to $\overrightarrow{q_1q_2}$. Clearly, P is contained between these two hyperplanes. In the next step we construct $S^{(3)}$ by selecting q_3 as the corner of P with maximum distance to the affine hull of $S^{(2)}$. Let $m^{(3)}$ be the point in the affine hull of $S^{(2)}$ such that $\overrightarrow{mq_3}$ is orthogonal to the affine hull of $S^{(2)}$. The next two walls of R are on one side the hyperplane through q_3 orthogonal to $\overrightarrow{mq_3}$ and on the other side the same hyperplane mirrored at the affine hull of $S^{(2)}$. Again, P is clearly contained between these two walls . We continue this process and finally construct $S = S^{(d-1)}$ and having collected all $2(d-1)$ walls of the $d-1$-orthotope R (see Figure 5 for an example in three dimensions).
Obviously, it holds that

$$S^{(d-1)} \subseteq P \subseteq R$$

and furthermore

$$HV(S) = \frac{HV(R')}{(d-1!)}$$

where R' is the $d-1$-orthotope R with each of its dimensions halved. We also have

$$HV(R') = \frac{1}{2^{d-1}} HV(R).$$

Theorem 4.1 follows immediately:

THEOREM 4.1. *Let P be a (d-1)-dimensional polytope and $S = q_1, \cdots, q_{d-1}$ the inscribed $d-1$-simplex derived by the algorithm described above; then we have* $HV(S) \geq \frac{HV(P)}{(d-1)!2^{d-1}}$.

4.2.2 Cutting a Simplex In $\mathbb{R}^2$, a median of a triangle is a segment connecting a corner to the midpoint of its opposing side. The three medians of a triangle intersect in a common point, its *centroid*. In $\mathbb{R}^d$ the centroid of a d-simplex is the intersection point of the segments connecting a corner with the the centroid of its opposite, $d-1$-dimensional face. The following simple theorem will prove useful for us later on:

THEOREM 4.2. *Let S be a d-simplex, C its centroid and H an arbitrary $d-1$ dimensional hyperplane containing C. Then H subdivides S into two d-dimensional polytopes with each of them having a hypervolume of at least* $\frac{HV(S)}{d^2}$.

Proof. Let us look at the 2-dimensional case first. Here, the medians Aa, Bb, Cc intersect in the centroid M,

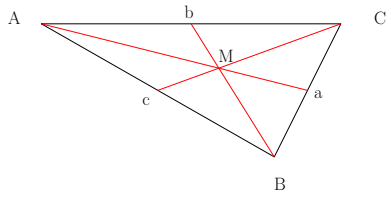

Figure 6: Medians and centroid of a triangle in $\mathbb{R}^2$. a, b, c are the midpoints of the respective edges, the medians Aa, Bb, Cc intersect in the *centroid*. Aa, Bb, Cc are all divided by the centroid into two pieces of ratio $1:2$.

furthermore each of the medians is split by M with ratio $1:2$. Hence, the areas of the triangles ΔABM and ΔABC also behave like $1:2$; so each triangle formed by an edge of the triangle and the centroid has exactly $1/3$ of the area of the whole triangle. See Figure 6 for an illustration. Note that the medians also partition the triangle into 6 triangles of equal size. Now consider any hyperplane l (line) in $\mathbb{R}^2$ passing through M; l must have at least 2 of the 6 triangles on either side, hence l subdivdes the triangle into two pieces, each of which have area at least $1/3$ of the whole triangle.

In d dimensions, the centroid M of a d-simplex Σ divides its medians in a ratio $1:d$. So each d-dimensional simplex which is formed by a face of Σ and M has a hypervolume of $1/d$ of the total hypervolume of the original d-simplex. Consider the opposing $(d-1)$-dimensional face of a corner P of the d-simplex. Its centroid partitions it into d many $(d-1)$-simplices of equal hypervolume. Each of them together with the centroid M of the original d-simplex forms a small d-simplex of hypervolume exactly $\frac{1}{d(d+1)}$ of the original d-simplex. Any hyperplane through M must have at least one corner of Σ on either side. We claim that any hyperplane through M cannot intersect the interior of all the small d-simplices (of hypervolume $\frac{1}{d(d+1)}$) adjacent to a corner P. Assume the opposite, then this hyperplane must properly intersect the interior of segment MP which contradicts that fact that this hyperplane passes through M. The theorem follows. ∎

How could this theorem help in decreasing the hypervolume of a facet f_π corresponding to a path π? We choose the λ_i's such that the hyperplane of the optimal path π' for this choice of the λ_i intersects the centroid of the inscribed simplex in f_π or lies below. The shortest path computation for these weights either certifies that π appears as a facet on the lower envelope or another path π' is exhibited which cuts off at least $1/d^2$ of the simplex S inscribed in f_π and hence at least $1/(dd!2^d)$ of the facet f_pi.

4.3 Algorithm Let us summarize the algorithm for larger, but fixed dimension d in the following. We are given a path π of cost c_π and resource consumptions r_π^j and are interested whether for some choice of the λ_i's the hyperplane corresponding to π is on the boundary of the lower envelope of all paths. We consider the path π in the d-dimensional space spanned by $\lambda_1, \ldots, \lambda_{d-1}$ and the aggregated weight. Since the λ_i are non-negative and restricted to sum up to 1 at most, the relevant facet f_π of the lower envelope can be explicitly computed. We then proceed as follows:

1. construct the inscribed simplex S_{f_π} for f_π
2. construct the centroid of S_{f_π}
3. use the first $d-1$ coordinates of the centroid to determine λ_i values
4. compute the optimal $s-t$-path π' under edge costs weighted according to the λ_i
5. if $\pi' = \pi$ report π to be part of the lower envelope/boundary of the convex hull of all pareto-optimal solutions and exit
6. otherwise intersect the facet f_π with the halfspace below the hyperplane $h_{\pi'}$ corresponding to π' to obtain the new facet f_π
7. if f_π disappears, report π not to be part of the convex hull of all pareto-optimal solutions and exit, otherwise go to step 1.

In Figure 7 the initial steps for $d = 2$ and $d = 3$ are illustrated for comparison.

According to our discussion, the hypervolume of the initial facet is bounded and decreased by a constant factor in each round, hence the total number of iterations is $O(d^2 \log_\alpha(nMd))$ for $\alpha = 1 - (1/(dd!2^d))$ and therefore polynomially bounded (for constant d). The runtime of a single iteration step is determined by a Dijkstra run to compute the optimal plane for a given λ (vector) and the computation of the corners of the new facet in which the reference plane is still 'alive'. With k being the number of hyperplanes identified so far (including the one corresponding to π) a naive way to perform this computation is to construct all $\binom{k}{d}$ corners in the arrangement of these hyperplanes. The hyperplane h_π corresponding to π ist still part of the lower envelope iff a corner involving h_π lies below or on all other hyperplanes. The corners of facet f_π are exactly the corners involving h_π which are below or on all other hyperplanes. Each corner can be computed via Gaussian elimination in $O(d^3)$ on a RealRAM. Hence a rough bound for the overall runtime is given by $\mathcal{O}(d^2 \log(nMd)(n \log(n) + m + d^{5+d-1} \log^{d-1}(nMd))$.

5 Robustness

In a concrete implementation, we cannot assume exact arithmetic on real numbers hence one has to be careful not to compromise correctness in an implementation based on floating point arithmetic. In our case, though, we can easily ensure correctness by employing an interval version of Gauss elimination; the corners then have *coordinate intervals* and the above/below test has to take the interval boundaries into account, that is, we only claim a facet f_π not to be part of the lower envelope, if all corners involving h_π are above all other constraints even in terms of their interval representation. For moderate dimensions, the coordinate intervals are typically still of acceptable size. If after $O(d^2 \log_\alpha(nMd))$ iterations the facet has not disappeared, we simply construct the shortcut and hence never compromise correctness of our procedure.

6 A Generalizable Binary Search Approach

So far, we developed a construction scheme for CH which allows to answer CCSP queries in the resulting graph. But as outlined before, we would also like to answer CSP queries at least in an approximative manner for $d \geq 3$.

Note, that obtaining such a solution via a binary search like algorithm is not as simple as for $d = 2$. If there exists a feasible solution for the bicriteria problem (i.e. a path not exceeding the resource bound), there is always also at least one path on the LE fulfilling the criterion, namely the one with minimal resource consumption. This path can be easily identified via a single Dijkstra run considering resource values only. For $d \geq 3$ this observation is no longer true. A feasible solution might be above the LE of all other $s-t$-paths without any of them being feasible as well. So nothing can be guaranteed when restricting the path search to the ones on the LE. Nevertheless we will propose an algorithm that searches systematically for a feasible path on the LE. For this procedure we adapt the idea of our witness search algorithm. In fact we will use 'imaginary' reference paths and check if they are part of the LE.

We start with the plane π described by $Mn + 1, R_1, R_2, \cdots, R_{d-1}$. Clearly, any feasible solution must dominate this plane. So we run our witness search routine with π as reference. Three possible outcomes must be considered: First, the reference plane is part of the LE. This means that there exists no feasible solution on the LE and we can abort the search. Secondly, the reference plane might be dominated by a *single* other plane. In that case we found an initial feasible solution. Third, the explored part of the LE might dominate π but none of the single planes. So the result is inconclusive up to now. We proceed in the second and third case by reset-

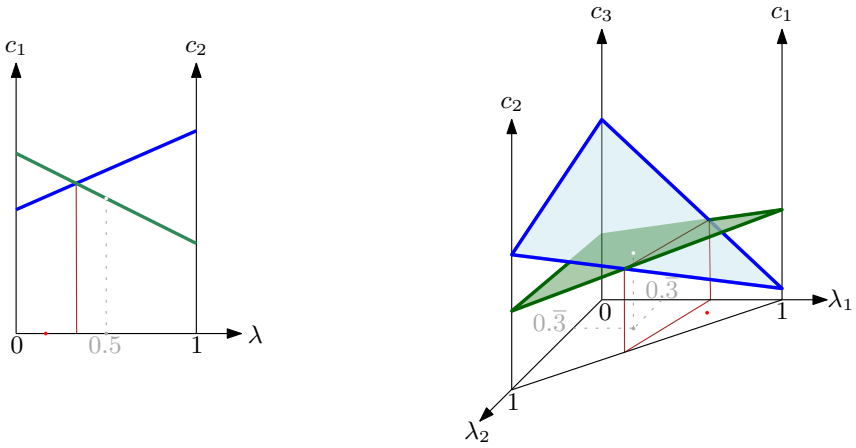

Figure 7: Binary search over the lower convex hull for 2d (left) and 3d (right). The reference path p is represented by the blue line/triangle. The gray dot indicates the initial λ(-vector) which leads to the detection of the new path q (represented by the green line/triangle). The intersection point/line of p and q determines the new facet that has to be considered. Its projection (brown) implies the new reduced search space, where the next λ support point (red) is chosen.

ting the costs of our imaginary plane by halving them. Now we apply again the witness search procedure. If the imaginary plane is matched exactly by a real one, we can abort the search. If we identify the new reference plane as part of the LE, we know that for any smaller cost value the plane will also be on the LE. Hence we only have to consider larger cost values in the remaining search. If on the other hand the plane is dominated by the LE, the same must be true for all larger cost values. Choosing the next cost value always as the mean of the remaining interval boundaries, we can halve this interval in every step. Because the possible cost values are integers, this provides us with a maximal number of $\log(Mn)$ iterations and therefore an overall polynomial runtime (as we proved before that the witness check runs in polynomial time). The output of the algorithm is the feasible plane with minimal costs that was found during the witness search – if such a plane was identified at all. An illustration of this algorithm for two dimensions can be found in Figure 8.

7 Experimental Results

We implemented our CH construction scheme for two and three edge weights (i.e. $d = 2$ and $d = 3$) and evaluated them on real-world data. The used test graphs (SL - Saarland, HE - Hessen, BW - Baden-Württembeg and CAL - California) are based on OpenStreetMap data. The implementation was written in C++, timings were taken on a single core of an Intel Core i5-3360M CPU with 2.80GHz and 16 GB RAM.

7.1 Metrics We considered the metrics euclidean distance (e), travel time (t), fuel costs (f) and 'quietness' (q). The euclidean distance was obtained by considering the latitude and longitude of the nodes, which are connected via a road segment. For estimating travel times we used the street tags provided in the OpenStreetMap data and assigned a typical speed for each road type. The explicit speed profile along with the distribution of the road types in our test graphs can be found in Table 1. Fuel costs were calculated as suggested in [5] using a gas price of €1.60 per liter. For measuring 'quietness' we assigned penalties to streets with a high maximum speed, in particular roads with an estimated speed up to 50 km/h received a penalty of 0, with 70 and 100 km/h a penalty of 1 and above (motorways) a penalty of 2 per metre. In the following we will present results on the features of the created CH-graphs and compare query times with and without applying CH for several metric combinations.

7.2 Bicriteria Paths ($\mathbf{d = 2}$) For the bicriteria case, we first used the metric combination travel time (t) and fuel costs (h) for all test graphs. The respective experimental results for computing the CH under these metrics can be found in Table 2. Note that we only contracted about 99.95% of the nodes during the shortcut creation as amongst the remaining nodes there was a large number of pareto-optimal paths, so a complete contraction would have added a too large number of shortcuts – slowing down both preprocessing and query processing. The first two columns show the number of nodes and edges of the used test graphs before

road type	speed (km/h)	edge ratio SL	HE	BW	CAL
motorway	130	1.29	0.60	0.39	0.48
motorway-link, primary(-link)	100	1.49	5.27	3.64	1.02
secondary(-link), tertiary(-link), trunk(-link)	70	23.47	25.40	22.63	8.45
unclassified, residential, road	50	56.26	53.80	57.63	86.96
living street, service, path	30	17.49	14.93	15.71	3.09

Table 1: Considered road types (extracted from OpenStreetMap) along with the estimated speed and the distribution of the types in the test graphs.

t+f	nodes	edges	Dijkstra polls	Dijkstra time(ms)	CH Preprocessing time(s)	CH Preprocessing edges	CH Query polls	CH Query time(ms)	CH Query speed-up
SL	203731	404521	$1.2 \cdot 10^5$	23.72	4	715571	152	0.05	460
HE	1121082	2269020	$6.1 \cdot 10^5$	176.67	65	4089958	506	0.29	611
BW	2459354	4993582	$1.3 \cdot 10^6$	409.77	121	9186484	599	0.42	965
CAL	11283833	22918849	$7.2 \cdot 10^6$	2097.42	1078	44806866	2434	1.98	1059

Table 2: Characteristics of the used test graphs along with experimental results for speeding up queries with CH. Query times and the poll numbers are averaged over 1000 random queries with the weight parameter λ being chosen u.a.r in $[0, 1]$ for each query.

applying CH. The next two columns show how plain Dijkstra performs in these graphs, exhibiting query times above two seconds for California. For the CH construction we provide the preprocessing time, which naturally increases with the graph size and the resulting number of pareto-optimal paths. Note that the maximal number of iterations in a single search step was only 8 for California. The number of shortcuts added is below the number of original edges, hence the resulting CH-graph has only about twice the size of the original one. In the last columns we present the number of poll operations and query times using bidirectional Dijkstra in the augmented CH-graph. Both of these values decrease dramatically compared to plain Dijkstra, leading to query times of a few milliseconds and a speed-up of two orders of magnitude. The speed-up for bicriteria paths reported in [5] is comparable or even better than this, but there not only pure CH was applied but also an adaption of landmarks on top. Moreover their used weight parameters are integer values only (chosen from the predefined interval) and the preprocessing times are longer.

In Figure 9 the differences between several metric combinations are noticeable. Metrics that are somewhat similar – like euclidean distance and travel time – produce less shortcuts and therefore better query times. In contrast to that, the 'quiet' metric considered on its own would lead to very different paths in comparison to the shortest or quickest one, because here edges with low speed limit are preferred. As a result the number of CH-edges is about 1 million (11 %) higher and query times in the CH-graph increase by a factor of 6.4 from 0.31ms to 1.98ms, while the run times for the conventional Dijkstra algorithm are relatively unaffected by the choice of the metrics. Still this yields a speed-up of factor 230 for the combination of travel time and quiet metric in the CH-graph (over 1000 for e+f).

t+e	bin. search time(ms)	CH bin. search time(ms)	CH bin. search speed-up
SL	286.52	0.88	322
HE	2139.73	3.64	586
BW	7223.38	7.48	965
CAL	20827.41	18.29	1138

Table 3: Experimental results for running binary search on the original and the preprocessed graph for minimizing travel time limiting the distance. The resource bound R was chosen as 1.1 times the minimum possible distance (computed by a conventional Dijkstra run). Query times are averaged over 500 random queries.

Table 3 shows the query times and the respective speed-ups in the CH-graph for the classical binary search algorithm to obtain an approximation to CSP under travel time and distance metrics. Not surprisingly, the speed-ups are similar to the ones reported in table 2, as the number of issued CCSP queries within the binary search procedure remains unaffected by the application of CH. Again, the difference in terms of the query times is quite considerable: Without making use of bicriteria CH, several seconds are required for computing the approximate CSP solution even in the small test graphs, while in the CH-graph this can be

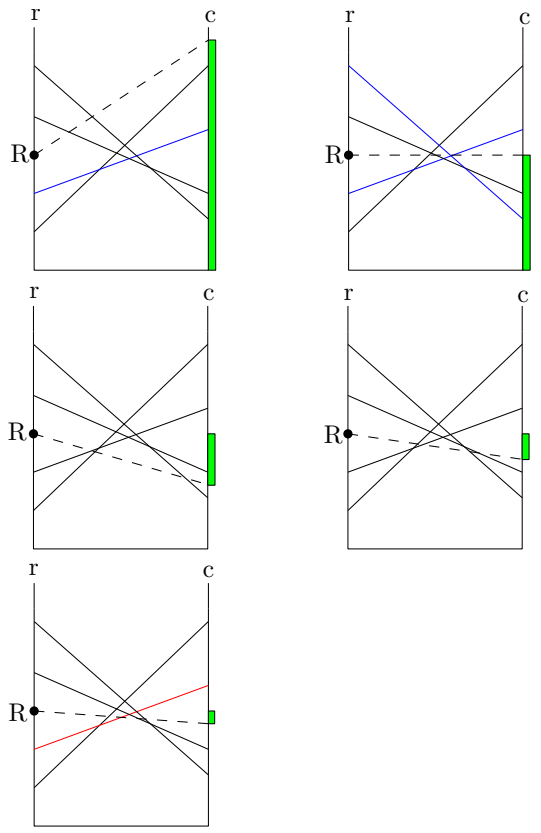

Figure 8: Illustration of the generalizable binary search procedure for $d = 2$. The (temporary) imaginary reference path is indicated by the dashed line. The other line segments correspond to 'real' paths. If the reference is not part of the LE the respective witness path(s) is/are coloured blue. The green box shows the remaining cost interval, which is halved in each step until it covers only one unit (bottom image). The solution returned by the algorithm is indicated by the red line.

achieved in the order of centiseconds even for California.

7.3 Tricriteria Paths (d = 3) For the three-dimensional case we considered the metric combination euclidean distance, travel time and fuel costs (e+t+f). Again we decided for an incomplete contraction in order to keep the total number of edges (original + shortcut) small. We used a contraction bound of 99.75% of the nodes, so slightly below the one for the two dimensional case, because the number of paths on the lower envelope and therefore the number of necessary shortcuts is expected to increase significantly with every additional metric considered. Like for the two-dimensional scenario we present query times for

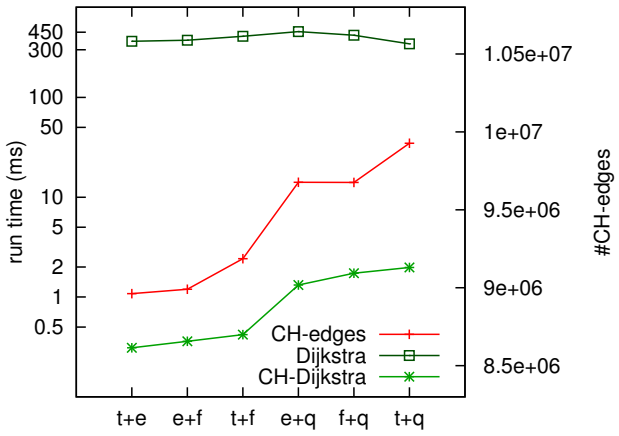

Figure 9: CH features for the BW graph for a variety of metric combinations. Runtime is presented in logscale.

plain Dijkstra and the CH-Dijkstra in the constructed graph, which correspond to answering single CCSP queries. Moreover we will evaluate our proposed binary search variant for retrieving approximate CSP solutions in higher dimensions for $d = 3$, measuring query times and quality of the found solutions.
The experimental results for CCSP queries in three dimensions are collected in Table 4. We observe that the query time for plain Dijkstra does not increase significantly compared to the two dimensional case. Applying CH, we see that the number of edges in the resulting graph is similar to or slightly above the one for the 2d CH-graph, although we stopped the contraction process earlier. The resulting speed-up is not as large as for $d = 2$ because every additional metric considered weakens the hierarchical structure of the graph. Nevertheless query times decrease by two orders of magnitudes and are again in the range of only a few milliseconds and hence allow for the answering of CCSP queries in real-time.

In Table 5 the query times for the generalized binary search procedure can be found, which allows us to compute approximate solutions for the NP-hard CSP problem. The used CSP queries ask for a path with minimal length that takes at most 25% longer than the fastest path and bears at most 25% increased fuel costs compared to the respective minimum (both bounds were computed by Dijkstra runs). Also computing an approximate CSP solution can be performed about two orders of magnitudes faster, reducing the runtime in California from several minutes to about a second. Moreover, we observe that the obtained approximate solutions have to be very close to the optimal ones,

e+t+f	Dijkstra	CH		CH-Dijkstra	
	time(ms)	time(s)	edges	time(ms)	speed-up
SL	29.45	4	714159	0.17	167
HE	226.92	29	4144965	1.18	191
BW	742.022	68	9152973	3.16	234
CAL	3299.07	561	44704479	15.62	211

Table 4: Speeding up flexible queries in three dimensions. Query times are averaged over 1000 random queries, with the three customizable weights being chosen u.a.r. in $[0, 1]$ for each query.

e+t+f	APX-CSP	APX-CSP + CH			
	time(ms)	time(ms)	speed-up	quality	fail
SL	1641.36	5.63	291	1.0079 (1.1197)	0.02
HE	15259.61	62.70	243	1.0014 (1.0291)	0.03
BW	29036.13	117.18	247	1.0018 (1.0266)	0.07
CAL	342795.45	1374.99	254	1.0010 (1.0358)	0.05

Table 5: Computing approximate CSP solutions in three dimensions with and without acceleration by CH. Query times are averaged over 500 random queries. Here, 'quality' denotes the averaged ratio of the cost returned by our algorithm (with the path respecting the resource bounds) and the cost of the shortest path (without considering resource bounds). Hence this provides an upper bound for the approximation quality of our solutions. The value in brackets is the maximum ratio that occurred. 'fail' reports the percentage of queries for which under the given constraints no feasible path was found (either because no such path exists or because our algorithm could not find such a solution).

because on average their distance is close to the respective shortest path distance(which is a lower bound for the optimal solution as resource bounds are neglected here). The number of queries for which no feasible path was found (see the last column) is pretty small, hence we can assume that our procedure finds in most cases a good approximation on the lower envelope (if such a solution exists at all).

f_1/f_2	1.01	1.05	1.25	1.50
1.01	815.98 3.67	1385.91 6.10	1061.01 5.58	1491.42 8.13
1.05	1365.89 6.30	1308.32 7.25	2634.04 11.95	1802.80 8.75
1.25	15590.40 83.71	15827.02 67.78	15259.61 62.70	9361.09 46.57
1.50	19077.87 94.16	14329.43 70.48	33148.12 172.06	20229.78 110.08

Table 6: Run times for approximate CSP solutions under varying constraints in the HE graph: Queries demand the shortest path which has a travel time no longer than f_1 times the fastest respective path and fuel costs not exceeding f_2 times the minimal possible costs. In each cell the lower number denotes the query time (in ms) for the 3d search sped up by CH, the upper number gives the runtime without acceleration (also in ms). All timings are averaged over 500 random queries.

The running times for a variety of different constraint combinations are presented in Table 6. While the speed-up by CH naturally remains almost unchanged (about 200), the query times increase significantly with larger resource bounds. Nevertheless for even larger bounds the queries become 'trivial', because the shortest path already fulfilled these constraints in over 98% of the cases right away. Therefore the query times decrease then significantly as we can find the optimal solution with a plain (accelerated) Dijkstra run.
So all in all our witness search procedure cannot only be used to construct the CH-graph efficiently but also to enable approximate CSP query answering for the three dimensional case.

8 Conclusions

In this paper we considered the multicriteria shortest path problem and presented a polynomial-time procedure to decide whether a given path π appears as a facet on the lower envelope of all pareto-optimal solutions. This procedure can be instrumented to build contraction hierarchies for the multicriteria shortest path problem such that paths optimizing a conic combination of the edge weights can be found orders of magnitudes faster than using ordinary Dijkstra. While optimizing over conic combinations in the bicriteria case is an established and powerful tool to obtain approximate solutions for the bicriteria constrained shortest

path problem, our procedure also proved useful for obtaining good heuristic solutions in the tri- and multi-criteria case. Our polynomial time bound was obtained via polyhedral arguments about the hypervolume of the facet of the lower envelope corresponding to π.

9 Acknowledgement

This work was partially supported by the Google Focused Grant Program on Mathematical Optimization and Combinatorial Optimization in Europe.

References

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Alternative routes in road networks. In *Proceedings of the 9th international conference on Experimental Algorithms*, SEA'10, pages 23–34, Berlin, Heidelberg, 2010. Springer-Verlag.

[2] V. Aggarwal, Y. Aneja, and K. Nair. Minimal spanning tree subject to a side constraint. In *32nd ACM Symposium on Theory of Computing (STOC)*, pages 286–295, 1982.

[3] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast Routing in Road Networks with Transit Nodes. *Science*, 316(5824):566, Apr. 2007.

[4] D. Delling and D. Wagner. Pareto paths with sharc. In J. Vahrenhold, editor, *Experimental Algorithms*, volume 5526 of *Lecture Notes in Computer Science*, pages 125–136. Springer Berlin Heidelberg, 2009.

[5] R. Geisberger, M. Kobitzsch, and P. Sanders. Route planning with flexible objective functions. In *ALENEX'10*, pages 124–137, 2010.

[6] R. Geisberger, M. N. Rice, P. Sanders, and V. J. Tsotras. Route planning with flexible edge restrictions. *J. Exp. Algorithmics*, 17(1):1.2:1.1–1.2:1.20, Mar. 2012.

[7] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, pages 319–333, 2008.

[8] A. Goldberg, H. Kaplan, and R. F. Werneck. Reach for a : Efficient point-to-point shortest path algorithms. In *In Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2006.

[9] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, pages 156–165, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[10] U. Lauther. Slow preprocessing of graphs for extremely fast shortest path calculations. *Lecture at the Workshop on Computational Integer Programming at ZIB.*, 1997.

[11] K. Mehlhorn and M. Ziegelmann. Resource constrained shortest paths. In M. Paterson, editor, *Algorithms - ESA 2000*, volume 1879 of *Lecture Notes in Computer Science*, pages 326–337. Springer Berlin / Heidelberg, 2000.

[12] S. Storandt. Route planning for bicycles - exact constrained shortest paths made practical via contraction hierarchy. In *22nd Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2012.