Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

MARCO BLANCO
RALF BORNDÖRFER
NAM DŨNG HOÀNG
ANTON KAIER
THOMAS SCHLECHTE
SWEN SCHLOBACH

The Shortest Path Problem with Crossing Costs

Zuse Institute Berlin (ZIB), Takustr. 7, D-14195 Berlin, Germany {blanco,borndoerfer, hoang, schlechte}@zib.de

ZIB Report 16-70 (November 2016)

Zuse Institute Berlin
Takustr. 7
D-14195 Berlin

Telefon: +49 30-84185-0
Telefax: +49 30-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Zuse Institute Berlin
Takustr. 7
D-14195 Berlin

Telefon: +49 30-84185-0
Telefax: +49 30-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

The Shortest Path Problem with Crossing Costs*

Marco Blanco[1], Ralf Borndörfer[1], Nam Dũng Hoàng[1], Anton Kaier[2], Thomas Schlechte[1], and Swen Schlobach[2]

[1]Zuse Institute Berlin, Berlin, Germany, {blanco, borndoerfer, hoang, schlechte}@zib.de
[2]Lufthansa Systems GmbH & Co. KG, Kelsterbach, Germany, {kaier, schlobach}@lhsystems.com

November 30, 2016

Abstract

We introduce the *shortest path problem with crossing costs* (SPPCC), a shortest path problem in a directed graph, in which the objective function is the sum of arc weights and *crossing costs*. The former are independently paid for each arc used by the path, the latter need to be paid every time the path intersects certain sets of arcs, which we call regions. The SPPCC generalizes not only the classical shortest path problem but also variants such as the resource constrained shortest path problem and the minimum label path problem. We use the SPPCC to model the flight trajectory optimization problem with overflight costs.

In this paper, we provide a comprehensive analysis of the problem. In particular, we identify efficient exact and approximation algorithms for the cases that are most relevant in practice.

1 Introduction

In this work, we introduce the shortest path problem with crossing costs (SPPCC), which seeks to find a minimum-cost path between two given nodes in a directed graph. As opposed to the classical shortest path problem (SPP), where the objective function is the sum of weights corresponding to the arcs of the path, we also consider *crossing costs*. These are associated with regions in the digraph, which are defined as sets of arcs. Every time the solution path intersects a region, the corresponding crossing costs are determined using the region's cost function, which takes the arcs in the intersection as input.

*This work was partially supported by the BMBF program "Mathematics for Innovations in Industry and Services", under the project "E-Motion" (grant 05M12ZAB).

The SPPCC constitutes a generalization of the SPP, for which a large number of solution methods have been developed, ranging from the famous Dijkstra algorithm [7] to very sophisticated techniques, based on preprocessing, which have been largely motivated by applications in road network routing, see [3]. We refer to [16] for an excellent survey on the theoretical and practical advances of algorithms for the shortest path problem. The SPPCC also generalizes variants of the SPP such as the resource constrained shortest path problem and the minimum label path problem. The first was proven to be NP-hard in [10], and multiple exact [12] and approximate [14] approaches have been developed for its solution. The second is also NP-hard in undirected graphs and, furthermore, cannot be approximated within a polylogarithmic factor unless P=NP [11]. A simplified version of the minimum label path problem, the problem of finding a path with the smallest number of different colors in an arc-colored graph, was studied in [5], where two exact approaches were suggested. Furthermore, the SPPCC encompasses variants where the crossing costs can be determined by only the first and the last nodes in each of the above mentioned intersections. To the best of our knowledge, this last problem has not yet been studied in the discrete optimization community.

In this paper we present a comprehensive analysis of the SPPCC. In particular, we make the following contributions:

- For the variant of the problem defined by assuming linear crossing costs that depend only on the first and last nodes used in each region:
  1. The *Two-Layer-Dijkstra Algorithm*, which solves the special case of pairwise disjoint regions in polynomial time.
  2. A proof that, under certain conditions, the problem obtained by relaxing the pairwise disjoint property is still solvable in polynomial time.
  3. A proof that allowing arbitrary region intersections makes the problem NP-hard.
- For the variant of the SPPCC given by constant-cost (*flat-rate*) regions:
  1. A proof of NP-hardness.
  2. The *ABC-Search Algorithm*, an efficient Branch & Bound framework that solves the problem to optimality.
- For the variant of the SPPCC defined by a single non-trivial region with piecewise-constant crossing costs:
  1. A proof of NP-hardness.
  2. An algorithmic framework that uses existing methods for solving the resource constrained shortest path problem and provides both an FPTAS and an exact approach.

Our motivation for studying the SPPCC comes from the real-world problem of flight trajectory optimization, as described in [13]. Here we have as input an airway network, origin and destination airports, an aircraft of a given weight, and countries' overflight fees functions, among other factors. The objective is to minimize the trajectory-dependent costs (as opposed to constant costs such as airport fees), which are usually the sum of fuel costs, overflight costs, and time costs. Fuel costs and time costs are in general directly proportional to the length of the trajectory, and can thus, after some simplifications, be projected down into the airway segments (i.e., the arcs of our graph representation), essentially reducing the problem to a (time-dependent) shortest-path problem, which can be solved to optimality by using a variant of Dijkstra's algorithm. Overflight fees, however, are determined by very diverse cost models, many of which make Dijkstra's algorithm deliver suboptimal solutions.

Overflight costs, also known as *ATC charges*, are the means by which Air Traffic Control authorities finance themselves. For each *cost airspace* (in general corresponding to a country) used during a flight, airlines are required to pay a fee, determined by factors such as the type of aircraft and the way in which the airspace is overflown. To the best of our knowledge, all currently used models define the overflight cost on an airspace as a function that takes three parameters: The distance defined by the flight trajectory in the airspace, the aircraft's maximum take-off weight, and the origin-destination pair. Since the latter two are given as input and are independent of the trajectory, we will assume from here on that the overflight cost function is solely dependent on the distance.

There exist two widely used definitions of the distance determined by the intersection of a flight trajectory and an airspace. The first and most natural variant is to consider the *flown distance* (FD), which is the total ground distance traversed by the aircraft in the airspace. The second variant uses the *great-circle-distance* (GCD), defined in this context as the length of the great circle connecting the coordinates where the aircraft enters and exits the airspace. If a trajectory intersects an airspace multiple times, the distance considered is the sum of the distances defined by each intersection.

The cost function itself is always non-decreasing and usually defined as a linear, constant, or piecewise-constant function. This results in six different combinations, which encompass nearly all cost models currently in use, and which lead to problems with varying degrees of difficulty. The only outliers, as of April 2016, are India, Argentina, Philippines, Democratic Republic of Congo, and Kenya; where the function used is piecewise-linear or even non-linear. In the remainder of this paper, we will ignore these cases. In Table 1 we see some representative airspaces which use the different models. Notice that when the cost function is constant for an airspace, it is irrelevant whether the distance type used is GCD or FD. Official overflight cost model publications can be found for example at [1], [2], or [8].

Table 1: Examples of airspaces using different cost models, as of April 2016

Function \ Distance	**GCD**	**FD**
Linear	European countries, USA, Thailand	Brazil, Ghana, Saudi Arabia, Iran
Constant	Egypt, Sudan, Myanmar, Japan	
Piecewise-constant	Ethiopia	ASECNA[1], Angola, Vietnam
Other	India, D.R. Congo	Argentina, Kenya

The problem of flight trajectory optimization under consideration of overflight costs is essential for minimizing airlines' operational expenses. In a case mentioned in [6], in 2007 an airline could save 884 USD in overflight charges on a flight from San Francisco to Frankfurt by deviating from the standard, most direct route; thus reducing Canada's expensive charges. However, despite its clear importance, the literature on the problem is scarce. The problem is presented in [13], but no concrete insight on solution approaches is offered. The authors in [4] also define the problem and consider the linear/GCD model, but their optimal control algorithm approximates overflight costs by using FD instead of GCD. Similarly, [15] introduces the linear/GCD model as well as the piecewise-constant/FD model, but the solution method used is heuristic and provides no guarantee of optimality.

In this paper, we study the flight trajectory optimization problem with overflight costs through the more abstract *Shortest Path Problem with Crossing Costs* (SPPCC). We present the first formal classification of existing overflight cost models and an extensive complexity analysis. Furthermore, we introduce efficient algorithms to handle the problem variants that are most common in practice.

In Section 2, we formally introduce the SPPCC, which models the problem of flight trajectory optimization with overflight costs. In Section 3 we present three efficient algorithms that solve the most important variants of the SPPCC to optimality or near-optimality. Finally, in Section 4 we give results on the computational complexity of other variants of the SPPCC which are interesting either from a practical or a theoretical point of view.

[1]In this context, ASECNA is a cost airspace encompassing 18 African countries, with an area of approximately 1.5 times that of Europe.

2 Combinatorial Model

Let $D = (V, A)$ be a directed graph, with source and target nodes s and t. For each $a \in A$, we have a constant cost $\varphi_a > 0$, which we refer to as the *weight* of arc a. Let $d : V \times V \to [0, \infty)$ be a function that we call *distance*. For the special case where $(u, v) = a \in A$, we use the notation $d_a := d(u, v)$.

Let $\mathcal{R} = \{R_1, \ldots, R_k\} \subseteq 2^A$ be a family of sets of arcs, that we shall call *regions*, such that $\bigcup_{R \in \mathcal{R}} R = A$. Furthermore, let $\mathcal{R}^G, \mathcal{R}^F \subseteq \mathcal{R}$ be a partition of $\mathcal{R}$. R^F represents the regions where every arc belonging to the solution path is important for computing the costs. For regions in R^G, only the nodes used to enter and exit the region are relevant. Let $f_R : [0, \infty) \to [0, \infty)$ be a non-decreasing function for every $R \in \mathcal{R}$. For a path p in D and a region $R \in \mathcal{R}$, let $\mathcal{P}_R^p$ be the set of all maximal subpaths of p contained in R. The nodes $t(p)$ and $h(p)$ denote the first and last nodes in a path p, respectively. Similarly, for an arc $a = (u, v)$, we use the notation $t(a) = u, h(a) = v$.

Finally, we can define the *crossing cost* corresponding to $R \in \mathcal{R}$ and an (s, t)-path p:

$$\gamma_R(p) = \begin{cases} 0 & \text{if } R \cap p = \emptyset \\ f_R\left(\sum_{q \in \mathcal{P}_R^p} d\left(t(q), h(q)\right)\right) & \text{if } R \in \mathcal{R}^G \text{ and } R \cap p \neq \emptyset \\ f_R\left(\sum_{a \in R \cap p} d_a\right) & \text{if } R \in \mathcal{R}^F \text{ and } R \cap p \neq \emptyset \end{cases}$$

Definition 1 *The* shortest path problem with crossing costs *(SPPCC) is to compute an (s, t)-path p in D that minimizes the cost function*

$$c(p) = \sum_{a \in p} \varphi_a + \sum_{R \in \mathcal{R}} \gamma_R(p). \tag{1}$$

Example 1 *We illustrate the definition of the SPPCC with the example in Figure 1. There, we have regions $\mathcal{R} = \{R^1, R^2, R^3, R^4, R^5, R^6\}$, with the corresponding crossing cost functions given in Table 2. In this example, we define the arc costs as $\varphi_a := 2d_a$ for each $a \in A$. We also suppose $R^2, R^4 \in \mathcal{R}^G$; $R^1, R^5 \in \mathcal{R}^F$. Since f_{R^3} and f_{R^6} are constant, it is irrelevant whether they belong to $\mathcal{R}^G$ or $\mathcal{R}^F$, but for the sake of completeness let us say $R^3, R^6 \in \mathcal{R}^F$. Given that R^2 and R^4 belong to $\mathcal{R}^G$, we use the distances between the first and last nodes of the intersecting subpaths to evaluate the crossing cost functions. The total cost of the example path is thus:*

$$\begin{aligned} c(p) =& \sum_{a \in p} \varphi_a + \gamma_{R^1}(p) + \gamma_{R^2}(p) + \gamma_{R^3}(p) + \gamma_{R^4}(p) + \gamma_{R^5}(p) + \gamma_{R^6}(p) \\ =& 88 + f_{R^1}(4 + 3 + 2) + f_{R^2}(4) + f_{R^3}(1 + 2 + 4) + f_{R^4}(4 + 6) + f_{R^5}(1 + 4 + 5) + 0 \\ =& 158. \end{aligned}$$

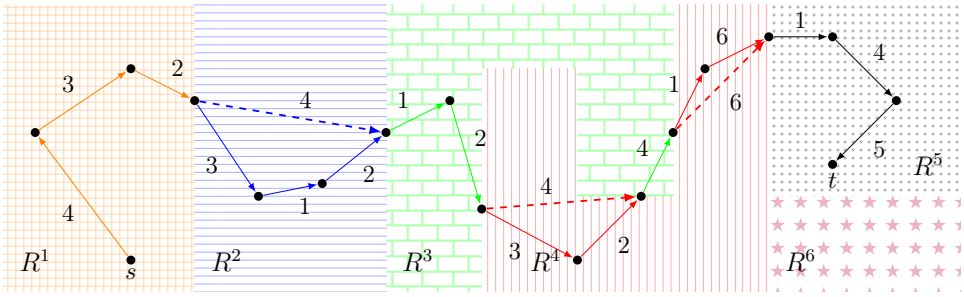

Figure 1: Example of an SPPCC instance with six regions. Arcs are labeled with the distances d_a between their endpoints, and the (s,t)-path p is represented by the continuous arrows. Dashed arrows represent distances between nodes that are not connected by an arc in the path.

Table 2: Crossing cost functions for Example 1

i	1	2	3	4	5	6
$f_{R^i}(x)$	$3x$	$2x$	10	$\begin{cases} 8 & \text{if } x \leq 6 \\ 20 & \text{if } x > 6 \end{cases}$	$\begin{cases} 5 & \text{if } x \leq 10 \\ 10 & \text{if } x > 10 \end{cases}$	15

The problem of flight trajectory optimization with overflight costs, as described in Section 1, can be modeled using the SPPCC as follows: We use V to represent waypoints and A to represent airway segments. The nodes s and t represent the departure and destination airports. An airspace, which is defined as a geographical region, can be modeled by a set $R \in \mathcal{R}$. For each $a \in A$, the cost φ_a can be seen as the fuel cost corresponding to traversing airway segment a. Furthermore, the great-circle-distance between two points u and v is represented by the function $d(u,v)$. $\mathcal{R}^G$ represents the set of airspaces that use the GCD-model for measuring distance, $\mathcal{R}^F$ those that use FD.

Each of the five important airspace cost models outlined in Section 1 induces a variant of the SPPCC. In each variant, we shall assume that all airspaces use the same cost model. That is, depending on the problem variant we require either $\mathcal{R}^F = \mathcal{R}$ and $\mathcal{R}^G = \emptyset$; or $\mathcal{R}^G = \mathcal{R}$ and $\mathcal{R}^F = \emptyset$. Likewise, we suppose that the function f_R has the same form (linear, constant or piecewise-constant) for all $R \in \mathcal{R}$. As before, if f_R is constant then the distinction between $\mathcal{R}^F$ and $\mathcal{R}^G$ is irrelevant, which leads to a total of five cases. We will denote them as listed in Table 3.

3 Algorithms

In this section we present exact and $(1+\varepsilon)$-approximation algorithms to handle three of the cases presented before, which are of particular importance

Table 3: Problem codes corresponding to types of regions in $\mathcal{R}$ and crossing cost functions f_R

$\mathcal{R} =$ / $\forall R$, f_R is...	$\mathcal{R}^G$	$\mathcal{R}^F$
Linear	**SPPCC/L/G**	**SPPCC/L/F**
Constant	**SPPCC/C/·**	
Piecewise-constant	**SPPCC/PC/G**	**SPPCC/PC/F**

in our flight trajectory optimization application.

3.1 Two-Layer-Dijkstra Algorithm for SPPCC/L/G with pairwise disjoint regions

In the first subsection, we consider the variant of the SPPCC where regions are pairwise disjoint and, for each region, the crossing cost is given by a linear function evaluated at the distance between the first and the last nodes used in that region (or the corresponding sum of distances, in case a region is intersected multiple times). This corresponds to the overflight cost model used in all European countries, USA, Canada, and many others. The assumption that the regions in $\mathcal{R}$ are pairwise disjoint is reasonable since usually there is a one-to-one correspondence between cost airspaces and countries, and the latter do not overlap.

The idea of the *Two-Layer-Dijkstra Algorithm* is to define a *coarse* graph on a subset of the nodes of the original (*fine*) graph, where shortest paths that traverse a region in the fine graph correspond to arcs in the coarse graph. We run a Dijkstra algorithm on the coarse graph, repeatedly using calls to Dijkstra on the fine graph to determine arc costs on-the-fly. Finally, from the optimal path in the coarse graph, the minimum-cost path in the fine graph is reconstructed.

We remark that, while there exist more general versions of this problem that do not assume pairwise disjointness of the regions and are also solvable in polynomial time (see Theorem 3 and Proposition 2), the algorithm can easily be generalized to handle the more realistic setting in our application where fuel costs are not constant and need to be computed during the execution of the algorithm.

Formally, we assume that for all $R \in \mathcal{R}$ there exists $\alpha_R > 0$ such that for a path p, whose intersections with R are subpaths $p_1^R, \ldots, p_r^R$, the corresponding crossing cost is $\gamma_R(p) = \alpha_R \cdot \big(d(t(p_1^R), h(p_1^R)) + \cdots + d(t(p_r^R), h(p_r^R))\big)$. Recall that $t(\cdot)$ and $h(\cdot)$ denote the first and last nodes of a path, or the tail- and head nodes of an arc.

Some more definitions are necessary before introducing the algorithm. For $R \in \mathcal{R}$, define the sets of nodes that can be used to enter or to exit R as

follows:

$$V^{\text{entry}}(R) := \begin{cases} \{v \in V | \delta^-(v) \backslash R \neq \emptyset, \delta^+(v) \cap R \neq \emptyset\} \cup \{s\} & \text{if } \delta^+(s) \cap R \neq \emptyset \\ \{v \in V | \delta^-(v) \backslash R \neq \emptyset, \delta^+(v) \cap R \neq \emptyset\} & \text{if } \delta^+(s) \cap R = \emptyset \end{cases}$$

$$V^{\text{exit}}(R) := \begin{cases} \{v \in V | \delta^-(v) \cap R \neq \emptyset, \delta^+(v) \backslash R \neq \emptyset\} \cup \{t\} & \text{if } \delta^-(t) \cap R \neq \emptyset \\ \{v \in V | \delta^-(v) \cap R \neq \emptyset, \delta^+(v) \backslash R \neq \emptyset\} & \text{if } \delta^-(t) \cap R = \emptyset \end{cases}$$

$V^{\text{entry}}(R)$ and $V^{\text{exit}}(R)$ will be referred to as the sets of *entry* and *exit* nodes of R, respectively. We assume that for $R \in \mathcal{R}$ we have $V^{\text{entry}}(R) \cap V^{\text{exit}}(R) = \emptyset$. If this is not the case, we can achieve it through a simple transformation that duplicates nodes and assigns incident arcs in an appropriate way, which does not affect the hardness of the problem.

Consider a directed graph $\bar{D} = (\bar{V}, \bar{A})$, where

$$\bar{V} := \bigcup_{R \in \mathcal{R}} (V^{\text{entry}}(R) \cup V^{\text{exit}}(R)) \quad \text{and} \quad \bar{A} := \bigcup_{R \in \mathcal{R}} (V^{\text{entry}}(R) \times V^{\text{exit}}(R)).$$

Given that a node cannot simultaneously be an entry- and an exit node of the same region, and since we assume that $\mathcal{R}$ partitions A, there exists a unique $R_{\bar{a}}$ for each $\bar{a} \in \bar{A}$ such that the tail node of $\bar{a}$ is an entry point of $R_{\bar{a}}$ and its head node an exit point. We define the function ComputeCost($\bar{a}$) as follows, for every $\bar{a} \in \bar{A}$: First we compute the shortest $(t(\bar{a}), h(\bar{a}))$-path in $D|_{R_{\bar{a}}}$ using arc costs φ. Here, we use $D|_{R_{\bar{a}}}$ to denote the subgraph of D induced by $R_{\bar{a}}$. Let $z_{\bar{a}}$ be the corresponding optimal value (or $+\infty$ if no such path could be found). ComputeCost($\bar{a}$) then returns $z_{\bar{a}} + f_{R_{\bar{a}}}(d(u, v))$.

Given these definitions, the *Two-Layer-Dijkstra Algorithm* is very simple. It runs in two phases, and works as follows:

The first phase is identical to the classical Dijkstra algorithm searching a shortest (s, t)-path on $\bar{D}$ except for one detail. Whenever an arc $\bar{a}$ is examined by the algorithm, we do not know its costs a priori. Instead, we compute them on-the-fly by calling ComputeCost($\bar{a}$).

In the second phase, each arc $\bar{a}$ in the optimal solution is expanded to a subpath in D by recomputing the shortest $(t(\bar{a}), h(\bar{a}))$-path in $D|_{R_{\bar{a}}}$ using arc costs φ. Finally, all such subpaths are concatenated to return an (s, t)-path in D. See Appendix A.1 for a formal description.

Theorem 1 *The Two-Layer-Dijkstra Algorithm returns the optimal solution to **SPPCC/L/G** if the sets in $\mathcal{R}$ are pairwise disjoint.*

Proof 1 *Analogously to the classical Dijkstra algorithm, a simple inductive process on the number of visited nodes in $\bar{D}$ shows that the Two-Layer-Dijkstra Algorithm finds the optimal solution from s to every $\bar{v} \in \bar{V}$; and in particular to t.*

Since the algorithm consists of $O(|V|^2)$ iterations of the Dijkstra algorithm, it has a total running time of $O(n^2 m + n^3 \log(n))$, where $n = |V|$ and $m = |A|$. Thus, we conclude the following result:

Corollary 1 *If the sets in $\mathcal{R}$ are pairwise disjoint, then **SPPCC/L/G** can be solved in polynomial time.*

3.2 ABC-Search Algorithm for SPPCC/C/·

In this subsection, we work with the assumption that all regions have constant crossing cost functions. One can imagine this as each region having an *opening cost* or *flat rate*, which once paid allows an unlimited use of its arcs. This cost model is currently used in approximately 70 countries (mostly in Africa and Asia).

We propose a Branch & Bound algorithm which we call *ABC-Search Algorithm* that runs as follows: At all times, we update a universal upper bound U on the cost of an optimal path and a corresponding feasible solution p. Each node of the tree is characterized by three sets $\mathcal{A}, \mathcal{B}, \mathcal{C} \subseteq \mathcal{R}$, which form a partition of $\mathcal{R}$ and represent *available*, *bought*, and *closed* regions (hence the algorithm's name[1]). Furthermore, at each node we compute lower and upper bounds on the node's optimal value, $L' > 0$ and $U' > 0$, which we compare to the global upper bound to decide whether to prune the tree at this point or to update the global upper bound. At each node, we obtain the local bounds by computing a shortest (s,t)-path using only arcs in the available and bought regions. After the path is found, we add to its length the corresponding cost of the bought regions (which at this point we have committed to purchasing) to obtain the node's lower bound. Then, the node's upper bound (the actual cost of the path) is computed by adding the cost of the crossed available regions to the previous value. For branching, we choose one of the available regions used by the path and create two children by buying or closing this region, respectively.

Formally, we assume that f_R is constant and non-negative for all $R \in \mathcal{R}$. As mentioned before, this means that in this case the distinction between $\mathcal{R}^G$ and $\mathcal{R}^F$ is irrelevant. We denote these constant crossing costs by $\gamma_R \geq 0$ for $R \in \mathcal{R}$. As we show in Proposition 3, the problem **SPPCC/C/·** is NP-hard and cannot be approximated within a polylogarithmic factor unless P=NP. The formal description can be found in Algorithm 1.

Since the algorithm is based on enumeration, it is clear that it finds the optimal solution. Our experience in the flight trajectory optimization problem shows that when minimizing the sum of fuel costs and overflight fees, the latter usually represent between 10% and 50% of the total cost of the optimal solution. This is important because the lower bounds in the algorithm sketched above are strongly dependent on the arc weights φ_a, which represent fuel consumption. In other words, in our application it will often be the case that the cost saved by flying around a large, expensive airspace will be offset by the extra fuel costs, thus bringing the search to a swift end.

[1] No relation to the Artificial Bee Colony algorithm.

Algorithm 1 ABC-Search Algorithm for **SPPCC/C/·**

Input: D, s, t, φ, $\mathcal{R}$ and $\{\gamma_R\}_{R\in\mathcal{R}}$.
Output: (s,t)-path p in D

Initialize $U \leftarrow \infty$, $p \leftarrow \emptyset$, $Q \leftarrow \{(\mathcal{R}, \emptyset, \emptyset)\}$.
while $Q \neq \emptyset$ **do**
    Extract an element $(\mathcal{A}, \mathcal{B}, \mathcal{C})$ from Q.
    Compute the shortest path on $D|_{\mathcal{A}\cup\mathcal{B}}$ using arc costs φ.
    if No path could be found **then continue**. ▷ *prune tree if node is infeasible*
    else Let p' be the optimal solution.
    Define $L' \leftarrow \sum_{a\in p'} \varphi_a + \sum_{R\in\mathcal{B}} \gamma_R$ and $U' \leftarrow L' + \sum_{R\in\mathcal{A}: R\cap p'\neq\emptyset} \gamma_R$ ▷ *local bounds*
    if $U' < U$ **then** update $U \leftarrow U'$ and $p \leftarrow p'$. ▷ *update global upper bound*
    if $L' \geq U$ **or** $R \cap p = \emptyset$ for every $R \in \mathcal{A}$ **then continue**.▷ *prune tree*
    Select some $R \in \mathcal{A}$ with $R \cap p \neq \emptyset$. ▷ *buy/close a used available region*

$$\begin{array}{ll} \mathcal{A}_1 \leftarrow \mathcal{A}\backslash\{R\} & \mathcal{A}_2 \leftarrow \mathcal{A}\backslash\{R\} \\ \mathcal{B}_1 \leftarrow \mathcal{B}\cup\{R\} & \mathcal{B}_2 \leftarrow \mathcal{B} \\ \mathcal{C}_1 \leftarrow \mathcal{C} & \mathcal{C}_2 \leftarrow \mathcal{C}\cup\{R\} \end{array}$$

    $Q \leftarrow Q \cup \{(\mathcal{A}_1, \mathcal{B}_1, \mathcal{C}_1), (\mathcal{A}_2, \mathcal{B}_2, \mathcal{C}_2)\}$ ▷ *add both new children to the tree*
return p

3.3 Exact Algorithm and FPTAS for SPPCC/PC/F and a Single Non-Trivial Region

In this subsection, let us consider a particular case of **SPPCC/PC/F**, where there exists exactly one region with non-zero crossing costs, and its cost function is piecewise constant.

While this seems like a very restricted setting, it is inspired by an important real-world case. The above-mentioned airspace *ASECNA* is formed by the union of 18 African countries, has a piecewise-constant cost function with four steps as well as a very large area, and is highly non-convex. Furthermore, it is routinely crossed by routes from Europe to South America and vice versa. Due to the non-convexity, it is common for aircraft to enter and exit the airspace more than once. Combined with the piecewise-constant cost model, this often leads to Dijkstra-like algorithms delivering solutions of a very bad quality, since subpaths of optimal paths are not necessarily optimal. Given the importance of this airspace, it is justified to consider approaches that assume the existence of a single region with these properties. We give this restricted problem the code name **SPPCC/A** (A for ASECNA).

The algorithm we propose is based on the observation that each step in the cost function can be used to naturally define an instance of the resource-constrained shortest path problem (RCSPP), which is NP-hard but for which there exists a Fully Polynomial Time Approximation Scheme (FPTAS), see [11] or [14]. Given the importance of the RCSPP in column-generation frameworks, exact algorithms that are efficient in practice have also been extensively studied in the literature, see [12] for an overview. Our algorithm solves at most as many RCSPP instances as steps in the cost function, either exactly or approximately, thus returning an optimal solution or an ε-approximate solution, respectively.

Formally, we assume that there exist two regions, $\mathcal{R} = \{R^0, R\}$, where $\mathcal{R} = \mathcal{R}^F$, $f_{R^0} \equiv 0$ and f_R is a piecewise-constant function with T steps, of the form

$$f_R(x) = \begin{cases} k_1 & \text{if } x \in (t_0, t_1] \\ k_2 & \text{if } x \in (t_1, t_2] \\ \vdots & \\ k_T & \text{if } x \in (t_{T-1}, t_T), \end{cases}$$

where $0 \leq k_1 < k_2 < \cdots < k_T$ and $0 = t_0 < t_1 < t_2 < \cdots < t_{T-1} < t_T = \infty$.

We also assume that for any digraph $D = (V, A)$, $s, t \in V$, $\varphi : A \to [0, \infty)$, $d : A \to [0, \infty)$, $B > 0$ and $\varepsilon \geq 0$ we have access to a black-box algorithm RCSPPSolver$(D, s, t, \varphi, d, B, \varepsilon)$ which returns an (s, t)-path p in D such that $\sum_{a \in p} d_a \leq B$ and $\sum_{a \in p} \varphi_a \leq (1 + \varepsilon) z^*$. Here, z^* is the optimal value of the RCSPP instance. Using the results mentioned above, we can assume that RCSPPSolver runs in exponential time when $\varepsilon = 0$, but in time polynomial in $\frac{1}{\varepsilon}$ and in the problem size when $\varepsilon > 0$.

Our contribution is Algorithm 2. In the following, we use $d(p, R)$ to denote $\sum_{a \in p \cap R} d_a$.

Algorithm 2 Exact/approximate approach for solving **SPPCC/A**

Input: D, s, t, φ, d, R and f_R as described above, $\varepsilon \geq 0$.
Output: (s, t)-path p in D

1: Define $S \leftarrow \{1, 2, \ldots, T\}$, $B_1 \leftarrow \infty$, $\mathcal{P} \leftarrow \emptyset$, $k \leftarrow 1$ ▷ *initialize*
2: Define $d^R(a) \leftarrow \begin{cases} d_a & \text{if } a \in R \\ 0 & \text{else} \end{cases}$ ▷ *define new distance function with support in R*
3: **while** $S \neq \emptyset$ **do**
4: $p_k \leftarrow$RCSPPSolver$(D, s, t, \varphi, d^R, B_k, \varepsilon)$ ▷ *solve RCSPP instance with bound B_k*
5: $\mathcal{P} \leftarrow \mathcal{P} \cup \{p_k\}$, $k \leftarrow k + 1$ ▷ *add p_k to the list of candidate paths*
6: Let i s.t. $t_i < d(p, R) \leq t_{i+1}$, set $B_k \leftarrow t_i$ ▷ *identify corresp. step function interval*
7: **for all** $j \geq i$ **do**
8: $S \leftarrow S \backslash \{j\}$ ▷ *discard all steps located on the right of identified step*
9: **return** $p \in \mathcal{P}$ that minimizes $c(p)$.

Since RCSPPSolver is called a linear number of times, it is clear that the running time is polynomial in $\frac{1}{\varepsilon}$ if $\varepsilon > 0$ but exponential if $\varepsilon = 0$. Next, we prove correctness.

Theorem 2 *Algorithm 2 returns a $(1 + \varepsilon)$-approximate solution to **SPPCC/A** for $\varepsilon \geq 0$.*

Proof 2 *Suppose the optimal solution p^* satisfies $t_{i'} < d(p^*, R) \leq t_{i'+1}$ for some i'. Let k such that $B_{k+1} \leq t_{i'}$ and $t_{i'+1} \leq B_k$. We claim that $c(p_k) \leq (1 + \varepsilon)c(p^*)$. By Line 6 in Algorithm 2, we have that there is some i with $B_{k+1} = t_i < d(p_k, R) \leq t_{i+1}$. Thus, we have $d(p_k, R) \leq t_{i'+1}$.*

Let $OPT = OPT^{\varphi} + OPT^{\gamma}$ be the optimal value corresponding to p^*, decomposed into arc costs and crossing costs. We can see that p^* is also the optimal solution to the RCSPP instance defined by $B = t_{i'+1}$, and that its corresponding value is precisely OPT^{φ}. Indeed, if this were not the case, there would exist a path p with $d(p, R) \leq t_{i'+1}$ and $\sum_{a \in p} \varphi_a < \sum_{a \in p^*} \varphi_a$. Such a path would have crossing costs at most as high as p^*, thus contradicting optimality of the latter. Therefore, we have

$$c(p_k) \leq (1 + \varepsilon)OPT^{\varphi} + OPT^{\gamma} \leq (1 + \varepsilon)(OPT^{\varphi} + OPT^{\gamma}) \leq (1 + \varepsilon)OPT.$$

4 Complexity Analysis

In this section, we give complexity results for different variants of the problem.

Proposition 1 ***SPPCC/L/F*** *can be solved in polynomial time.*

Proof 3 *Let $R \in \mathcal{R}$ and p an (s,t)-path. By linearity of f_R, for some constant $\alpha_R \geq 0$ we have*

$$\gamma_R(p) = f_R\left(\sum_{(u,v)\in R\cap p} d(u,v)\right) = \alpha_R \cdot \left(\sum_{(u,v)\in R\cap p} d(u,v)\right) = \sum_{(u,v)\in R\cap p} \alpha_R \cdot d(u,v)$$

Thus, we seek to compute an (s,t)-path p that minimizes

$$c(p) = \sum_{a\in p}\left(\varphi_a + d_a \sum_{R\in\mathcal{R}:a\in R} \alpha_R\right),$$

In other words, all overflight costs can be projected to the arcs, and thus ***SPPCC/L/F*** *reduces to the classical shortest path problem.*

While the algorithms described in Section 3 tackle problems that are very important in our application, Theorem 3 and Proposition 2 are concerned with cases that are interesting rather from a theoretical point of view, and could have applications in other fields.

Theorem 3 *If $\mathcal{R}$ is a laminar family, i.e., any two sets in $\mathcal{R}$ are either disjoint or one a subset of the other, then* ***SPPCC/L/G*** *can be solved in polynomial time.*

Proof 4 *For a region R, let us define its* level *$\ell(R) \in \mathbb{N}$ as the number of different regions $R' \in \mathcal{R}$ such that R is a proper subset of R'. That is, $\ell(R) := |\{R' \in \mathcal{R} | R \subsetneq R'\}|$. Let $N := \max\{\ell(R) | R \in \mathcal{R}\}$. We will prove the result by induction on N.*

By Corollary 1, we know that for $N = 0$ the claim holds. Let us now assume that it is true for $N-1$. Let $\mathcal{R}_N := \{R \in \mathcal{R} | \ell(R) = N\}$.

For each $R \in \mathcal{R}_N$, we consider $V^{entry}(R)$ and $V^{exit}(R)$ as defined in Subsection 3.1. For each $u \in V^{entry}(R)$, $v \in V^{exit}(R)$, let $z(u,v)$ be the value of the shortest (u,v)-path in $D|_R$. If no such path exists, then $z(u,v)$ takes the value $+\infty$.

We then define a new SPPCC instance as follows:

- $D'(V', A')$, $V' := V$, $s' := s$, $t' := t$, $\mathcal{R}' := \mathcal{R}\backslash\mathcal{R}_N$
- $A' := (A\backslash\{\bigcup_{R\in\mathcal{R}_N} R\}) \cup \{(u,v) \in V^{entry}(R) \times V^{exit}(R)$ *for some* $R \in \mathcal{R}_N\}$
- $\varphi'_a := \begin{cases} \varphi_a & \text{if } a \in A \\ z(u,v) + f_R(d(u,v)) & \text{else} \end{cases}$ *for* $a \in A'$.

It can now be seen that for each optimal solution of the original instance there exists a unique optimal solution in the new instance which has the same objective value, since maximal subpaths contained in some $R \in R^N$ must be shortest paths with respect to φ. Similarly, for each optimal solution in the new instance, there exists at least one optimal solution in the original instance which has the same objective value. Thus, using our induction hypothesis, we can compute an optimal solution to the new instance in polynomial time, and from there derive the optimal solution to the original instance.

The following proposition has a technical proof that we do not include here due to space constraints, see Appendix A.2.

Proposition 2 *If the intersection of any three sets in $\mathcal{R}$ is empty, then **SPPCC/L/G** can be solved in polynomial time.*

Theorem 4 ***SPPCC/L/G** is NP-hard, even if D is acyclic.*

Proof 5 *We do a reduction from the Path Avoiding Forbidden Pairs Problem (PAFP), which is defined as follows. Given are a directed, acyclic graph $D = (V, A)$, two nodes $s, t \in V$ and a set of pairs of nodes $\mathcal{F} \subseteq V \times V$. The objective is to decide whether there exists a path from s to t that uses at most one node of each of the pairs in $\mathcal{F}$. The PAFP problem is known to be NP-complete* [9].

Given an instance of the PAFP problem, we label the nodes $V = \{v_1, v_2, \ldots, v_n\}$ according to the topological ordering of D, so that $(v_i, v_j) \in A$ implies $i < j$. We define a distance function d as follows for each $u, v \in V$:

$$d(u,v) := \begin{cases} 0 & \text{if } \{u,v\} \notin \mathcal{F} \\ 1 & \text{if } \{u,v\} \in \mathcal{F} \end{cases} .$$

For each $\{v_i, v_j\} \in \mathcal{F}$, with $i < j$, we define $R^{i,j} := \{(v_k, v_\ell) \in A | i \leq k < \ell \leq j\}$. Let $\mathcal{R} := \bigcup_{i,j:(v_i,v_j)\in\mathcal{F}} R^{i,j}$. For each such set $R^{i,j}$, we define $f_{R^{i,j}}$ as the identity. We define arc weights $\varphi_a = 0$ for each $a \in A$, and consider the resulting SPPCC instance. See Figure 2 for an example of the construction.

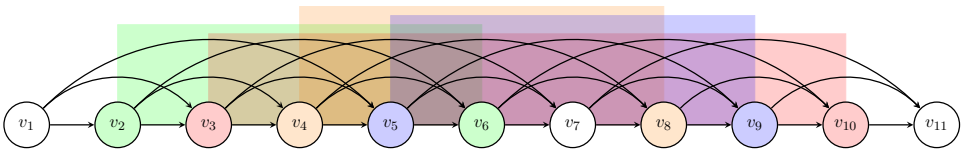

Figure 2: Example of a PAFP instance, where a feasible (v_1, v_{11})-path is sought. Here, $\mathcal{F} = \{(v_2, v_6), (v_3, v_{10}), (v_4, v_8), (v_5, v_9)\}$. For each forbidden pair, we define a region. In the figure, an arc a belongs to a region R if and only if a is completely contained in R.

*Given an (s,t)-path p and a forbidden pair $\{v_i, v_j\} \in \mathcal{F}$, the overflight cost $\gamma_{R^{i,j}}(p)$ in the **SPPCC/L/G** instance is positive if and only if both v_i and v_j are visited by p. Thus, the optimal value of is 0 if and only if the PAFP instance has a feasible solution. This means that a polynomial reduction of PAFP to **SPPCC/L/G** can be made, completing the proof.*

Proposition 3 ***SPPCC/C/·** is NP-hard.*

Proof 6 *The problem of finding an (s,t)-path using the smallest number of different colors in a directed graph where each arc has a color was proven to be NP-hard in* [5]*. It is easy to see that this corresponds to a restricted version of **SPPCC/C/·** in which all arc weights φ_a are set to 0 and all crossing costs are set to 1, thus implying NP-hardness of **SPPCC/C/·**. Furthermore, in* [11] *the authors show that for undirected graphs, if the colors have non-unit costs and one seeks to minimize the costs, the problem cannot be approximated within any polylogarithmic factor unless P=NP. Since this problem is also a particular case of **SPPCC/C/·**, the hardness result also applies for the latter.*

The following corollary is a direct consequence of the previous result, since **SPPCC/PC/G** and **SPPCC/PC/F** are both generalizations of **SPPCC/C/·**.

Corollary 2 ***SPPCC/PC/G** and **SPPCC/PC/F** are NP-hard.*

Furthermore, we can prove that even if $\mathcal{R}$ consists of a single element whose cost function is piecewise-constant and has two steps, then the problem remains NP-hard.

Proposition 4 ***SPPCC/PC/F** is NP-hard, even if $\mathcal{R} = \{R\}$ and f_R has two steps.*

Proof 7 *We will show that in this case, **SPPCC/PC/F** generalizes the resource-constrained shortest path problem (RCSPP), which is well known to be NP-hard* [10]*. Indeed, given $B > 0$, consider the following cost function:*

$$f_R(d) = \begin{cases} 0 & \text{if } d \leq B \\ M & \text{if } d > B, \end{cases}$$

where M is a large positive number. If M is large enough, then the algorithm will try to find a path such that its total distance in R (as defined by d) is at most B. This corresponds precisely to the RCSPP.

5 Conclusions

In this paper, we analyzed the complexity of the most relevant variants of the SPPCC, and introduced three efficient algorithms for the solution of cases that are most important in our application to the flight trajectory optimization problem with overflight costs. An overview of the results and algorithms presented can be found in Table 4.

Table 4: Overview of results

SPPCC/L/G	• Section 3.1: Poly-time *Two-Layer-Dijkstra* algorithm for the case in which regions don't intersect • Theorem 3 and Proposition 2: Polynomial if regions form a laminar family or if any arc belongs to at most two regions • Theorem 4: NP-hard for arbitrary region intersections
SPPCC/L/F	• Proposition 1: Polynomial
SPPCC/C/·	• Proposition 3: NP-hard • Section 3.2: Efficient *ABC-Search* Branch & Bound algorithm
SPPCC/PC/G	• Corollary 2: NP-hard
SPPCC/PC/F	• Proposition 4: NP-hard even if there exists only one non-trivial region and its cost function has two steps • Section 3.3: Efficient exact algorithm and FPTAS if there exists a single non-trivial region

6 Acknowledgements

We thank Lufthansa Systems GmbH & Co. KG for providing us with the data that made the overflight cost model classification possible, as well as for the many fruitful discussions.

References

[1] Federal Aviation Administration, *Overflight fees*, 2016. [Online; accessed 8-April-2016].

[2] ASECNA, *Air navigation services charges*, 2016. [Online; accessed 8-April-2016].

[3] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato Werneck, *Route planning in transportation networks*, Technical Report MSR-TR-2014-4, 2014.

[4] Pierre Bonami, Alberto Olivares, Manuel Soler, and Ernesto Staffetti, *Multiphase mixed-integer optimal control approach to aircraft trajectory optimization*, Journal of Guidance, Control, and Dynamics **36** (July 2013), no. 5, 1267–1277.

[5] Hajo Broersma, Xueliang Li, Gerhard Woeginger, and Shenggui Zhang, *Paths and cycles in colored graphs*, Australasian journal of combinatorics **31** (2005), 299–311.

[6] Susan Carey, *Calculating costs in the clouds*, The Wall Street Journal (2007March).

[7] Edsger W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik **1** (1959), no. 1, 269–271.

[8] EUROCONTROL, *Establishing route charges*, 2016. [Online; accessed 8-April-2016].

[9] Harold N. Gabow, Shachindra N. Maheshwari, and Leon J. Osterweil, *On two problems in the generation of program test paths*, IEEE Transactions on Software Engineering **SE-2** (1976), no. 3, 227–231.

[10] Michael R. Garey and David S. Johnson, *Computers and intractability: A guide to the theory of np-completeness*, W. H. Freeman & Co., New York, NY, USA, 1979.

[11] Refael Hassin, Jérôme Monnot, and Danny Segev, *Approximation algorithms and hardness results for labeled connectivity problems*, Journal of Combinatorial Optimization **14** (2007), no. 4, 437–453.

[12] Stefan Irnich and Guy Desaulniers, *Shortest path problems with resource constraints*, Column generation, 2005, pp. 33–65.

[13] Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković, *Operations*, Quantitative problem solving methods in the airline industry, 2012, pp. 283–383 (English).

[14] Dean H. Lorenz and Danny Raz, *A simple efficient approximation scheme for the restricted shortest path problem*, Oper. Res. Lett. **28** (June 2001), no. 5, 213–219.

[15] Robert L. Schultz, Stephen G. Pratt, and Donald A. Shaner, *Four-dimensional route planner*, Google Patents, 2003. WO Patent App. PCT/US2002/029,474.

[16] Christian Sommer, *Shortest-path queries in static networks*, ACM Comput. Surv. **46** (March 2014), no. 4, 45:1–45:31.

A Appendix

Here, we give a detailed description of the Two-Layer-Dijkstra Algorithm presented in Subsection 3.1, as well as the proof of Proposition 2.

A.1 The Two-Layer-Dijkstra Algorithm

The complete version of the Two-Layer-Dijkstra Algorithm can be found in Algorithm 3. We use the same notation as in Section 3.1. In particular, we make use of the subroutine ComputeCost($\bar{a}$) defined before. We also define the subroutine ShortestPath(s, t, D, φ), which takes a directed graph, a pair of nodes in that digraph, and a non-negative real function on the arcs; and uses Dijkstra's algorithm to return the shortest path connecting the two nodes in the digraph.

A.2 Proof of Proposition 2

Proof 8 *We will prove that the special case of* ***SPPCC/L/G*** *where each arc belongs to at most two regions is solvable in polynomial time. In fact, it is easy to see how our algorithm can be extended to handle more general*

Algorithm 3 Two-Layer-Dijkstra Algorithm for **SPPCC/L/G**

Input: D, s, t, φ, $\mathcal{R}$ and f_R for $R \in \mathcal{R}$.
Output: (s,t)-path p in D

Construct $\bar{D}$, insert dummy node v^{null} to $\bar{V}$ ▷ *$\bar{D}$ as in Section 3.1*
for all $\bar{v} \in \bar{V}$ **do**
    $dist[\bar{v}] \leftarrow \infty$ ▷ *initialize distances*
    $pred[\bar{v}] \leftarrow v^{\text{null}}$ ▷ *initialize predecessors*
$dist[s] \leftarrow 0$
$Q \leftarrow \bar{V}$
while $Q \neq \emptyset$ **do**
    $\bar{u} \leftarrow \text{argmin}_{\bar{v} \in Q}(dist[\bar{v}])$ ▷ *choose node with minimum distance*
    if $\bar{u} = t$ **then**
        break
    $Q \leftarrow Q \backslash \{\bar{v}\}$
    for all $\bar{v}$ s.t. $(\bar{u}, \bar{v}) \in \bar{A}$ **do**
        $\ell_{\bar{u},\bar{v}} \leftarrow \textsc{ComputeCost}((\bar{u}, \bar{v}))$ ▷ *compute cost corresponding to arc $(\bar{u}, \bar{v})$*
        if $dist[\bar{v}] > dist[\bar{u}] + \ell_{\bar{u},\bar{v}}$ **then** ▷ *if new cost is better*
            $dist[\bar{v}] \leftarrow dist[\bar{u}] + \ell_{\bar{u},\bar{v}}$ ▷ *update distance*
            $pred[\bar{v}] \leftarrow \bar{u}$ ▷ *update predecessor*
$p \leftarrow \emptyset, u \leftarrow t$
while $pred[u] \neq v^{\text{null}}$ **do**
    $p \leftarrow \textsc{ShortestPath}(pred[u], u, D, \varphi) \cup p$ ▷ *reconstruct path in D recursively*
return p

Algorithm 4 Algorithm for **SPPCC/L/G** with pairwise region intersections

Input: D, s, t, φ, $\mathcal{R}$ and f_R for $R \in \mathcal{R}$.
Output: (s,t)-path p in D

$Q \leftarrow \{\{s\}\}$
while $Q \neq \emptyset$ **do**
  $p \leftarrow \text{argmin}_{q \in Q}(c'(q))$ ▷ *extract minimum-cost path from queue*
  $u \leftarrow h(p)$ ▷ *define u as the last node of the path*
  if $u = t$ **then**
    return p
  $Q \leftarrow Q \backslash \{p\}$
  for all v s.t. $(u,v) \in A$ **do**
    $p' \leftarrow p \cup (u,v)$ ▷ *extend p by appending (u,v) at the end*
    if $Q(p) = \emptyset$ **then** ▷ *if no comparable paths exist*
      $Q \leftarrow Q \cup \{p'\}$ ▷ *insert path to queue*
    else if p' dominates q for each $q \in Q(p)$ **then** ▷ *see Definition 2*
      $Q \leftarrow Q \cup \{p'\}$
      $Q \leftarrow Q \backslash Q(p)$
return $\emptyset$ ▷ *if no path could be found*

cases, where each arc belongs to at most N regions, for $N \in \mathbb{N}$. However, the running time increases exponentially in N, and the notation becomes very difficult. For that reason, we present a polynomial-time algorithm for $N = 2$.

In this paper, we have defined a path as a set of arcs. For this proof, it is more convenient to think of a path as a set of both nodes and arcs.

Our algorithm follows the same idea as the standard dynamic programming algorithms commonly used for the resource constrained shortest path problem, see [12]*. We will keep a set Q of paths that start at s. Initially, this set contains the single element $\{s\}$. In every iteration, we extract from Q the path p with smallest cost (using the cost function defined below in* (2)*), and insert to Q all paths obtained by appending an arc to p at its end node. Then, we carry out a* dominance *step, where iteratively we compare paths in Q that end at the same node, and eliminate a (possibly empty) subset of them. As opposed to Dijkstra's algorithm, where after the domination step there exists at most one path ending at v for every $v \in V$, we will keep $\mathcal{O}(|V|^2)$-many such paths for each v.*

Given a path $p = (v_1, v_2, \ldots, v_k)$, let $\mathcal{R}^h(p) := \{R \in \mathcal{R} | (v_{k-1}, v_k) \in R, v_k \notin V^{exit}(R)\}$. That is, $\mathcal{R}^h(p)$ represents the set of regions in $\mathcal{R}$ that p crossed with its last edge but has not yet exited. By definition of the problem, we have $\mathcal{R}^h(p) \leq 2$ for every p. For each $R \in \mathcal{R}^h(p)$, let $g(p,R)$ represent the node $v_i \in p$ such that the (v_i, v_k)-subpath of p is maximally contained in R. Furthermore, we define a cost function similar to that defined in (1)*, and*

using the same notation:

$$c'(p) = \sum_{a \in p} \varphi_a + \sum_{R \in \mathcal{R}} f_R \left(\sum_{q \in \mathcal{Q}_R^p} d(t(q), h(q)) \right). \tag{2}$$

Here, $\mathcal{Q}_R^p := \{q \in \mathcal{P}_R^p | h(q) \in V^{exit}(R)\}$. *That is,* $\mathcal{Q}_R^p$ *is the set of subpaths of* p *that are maximally contained in* R *and such that their last node is an exit node of* R. *Thus, the cost function* c' *returns the sum of arc weights of a path plus the crossing costs corresponding to all maximal intersections except for those containing the last arc in* p, *if such an intersection could be extended by adding an arc to* p.

Definition 2 *Given two paths* p^1 *and* p^2 *starting at* s, *we say* p^1 *and* p^2 *are* comparable *if the following two conditions are satisfied:*

1. $\mathcal{R}^h(p^1) = \mathcal{R}^h(p^2)$
2. $g(p^1, R) = g(p^2, R)$ *for every* $R \in \mathcal{R}^h(p^1)$

For a set of paths Q *and a path* p, *let* $Q(p)$ *be the set of paths in* Q *that are comparable to* p. *Furthermore, we say* p^1 *is* dominated *by* $p^2 \in Q(p^1)$ *if* $c'(p^1) \geq c'(p^2)$.

In the dominance step, we delete a path p^1 *from* Q *only if there exists another* $p^2 \in Q$ *that dominates it. That is, if* p^2 *is a better candidate that used the same nodes to enter the regions* p^1 *is currently visiting.*

A formal description can be found in Algorithm 4. It should be noted that the approach sketched could be made more efficient, for example by not storing the complete paths in the priority queue. However, in this proof we sacrifice efficiency for the sake of clarity.

It is clear that the algorithm runs in polynomial time, since for every $v \in V$, *there are at most* $\mathcal{O}(|V|^2)$ *insertions to* Q *of paths ending at* v. *We also know that if a candidate subpath is eliminated, the domination criterion ensures that we are keeping an alternative that will lead to a solution at least as good as any containing the deleted subpath. Furthermore, by the way* c' *is defined, and since* t *is an exit node of any region containing arcs incident to it, we know that for any* (s,t)*-path considered during the algorithm, we have* $c(p) = c'(p)$, *thus ensuring optimality of the final solution.*