Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

MARCO BLANCO
RALF BORNDÖRFER
NAM DŨNG HOÀNG
ANTON KAIER
PEDRO MARISTANY DE LAS CASAS
THOMAS SCHLECHTE
SWEN SCHLOBACH

Cost Projection Methods for the Shortest Path Problem with Crossing Costs

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

ZIB Report 17-48 (August 2017)

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

Cost Projection Methods for the Shortest Path Problem with Crossing Costs*

Marco Blanco[1,2], Ralf Borndörfer[1], Nam Dũng Hoàng[1,4], Anton Kaier[2], Pedro M. Casas[1], Thomas Schlechte[1,3], and Swen Schlobach[2]

1 **Zuse Institute Berlin, Berlin, Germany**
{blanco, borndoerfer, hoang, maristany, schlechte}@zib.de
2 **Lufthansa Systems GmbH & Co. KG, Kelsterbach, Germany**
{marco.blanco-sandoval, anton.kaier, swen.schlobach}@lhsystems.com
3 **LBW Optimization GmbH**
schlechte@lbw-optimization.de
4 **Faculty of Mathematics, Mechanics and Informatics, Vietnam National University, Hanoi, Vietnam**
hoangnamdung@hus.edu.vn

Abstract

Real world routing problems, e.g., in the airline industry or in public and rail transit, can feature complex non-linear cost functions. An important case are costs for crossing regions, such as countries or fare zones. We introduce the *shortest path problem with crossing costs* (SPPCC) to address such situations; it generalizes the classical shortest path problem and variants such as the resource constrained shortest path problem and the minimum label path problem.

Motivated by an application in flight trajectory optimization with *overflight costs*, we focus on the case in which the crossing costs of a region depend only on the nodes used to enter or exit it. We propose an exact *Two-Layer-Dijkstra Algorithm* as well as a novel *cost-projection* linearization technique that approximates crossing costs by shadow costs on individual arcs, thus reducing the SPPCC to a standard shortest path problem. We evaluate all algorithms' performance on real-world flight trajectory optimization instances, obtaining very good à posteriori error bounds.

1998 ACM Subject Classification "F.2.2 Nonnumerical Algorithms and Problems", "G.1.6 Optimization", "G.2.2 Graph Theory"

Keywords and phrases shortest path problem, resource constrained shortest path, crossing costs, flight trajectory optimization, overflight fees, cost projection

Digital Object Identifier 10.4230/OASIcs.CVIT.2016.23

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 a path intersects a region, the corresponding crossing costs are determined using the region's cost

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

function, which depends on the arcs in the intersection. Crossing costs come up as overflight costs in aircraft routing, where the regions correspond to countries [15], in passenger routing in public and rail transport, where fare zones and air distance dependent tariffs are common [**?**], and in intermodal routing, when modes are changed [**?**].

The SPPCC constitutes a generalization of the classical SPP, see [18] for an excellent survey, and of several NP-hard variants such as the resource constrained shortest path problem and the minimum label path problem [11]. While the first can be approximated in polynomial time [16], the second cannot within a polylogarithmic factor unless P=NP [13]. Exact algorithms for the resource constrained shortest path problem have been suggested by [14], and for 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, by [7].

We focus on a variant of the SPPCC that arises from assuming linear crossing costs that depend only on the first and last nodes used in each region. To the best of our knowledge, this problem has not yet been studied in the optimization community. The paper makes the following contributions:

1. The *Two-Layer-Dijkstra Algorithm*, which solves the problem to optimality in polynomial time.
2. Two *cost-projection* methods (one combinatorial, one LP-based) that transform crossing costs into regular arc weights, these are used to transform the problem into a related standard shortest path problem.
3. A computational evaluation of all algorithms presented.

We will show that cost projection allows to deal with complex crossing costs in a way that is highly accurate, computationally efficient, and allows to deal with changes of arc weights acoording to, e.g., weather.

Our motivation for studying the SPPCC stems from an application in flight trajectory optimization, as described in [15]. This problem can be described as follows. The input is an airway network, origin and destination airports, an aircraft of a given weight, a weather forecast, and countries' overflight fees functions, among other factors. The output is a flight route. 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 onto the airway segments (i.e., the arcs of a 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 last two 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 linear, constant, or piecewise-constant. 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 list some representative airspaces which use the aforementioned 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 documentations of overflight cost models can be found for example at [1], [2], or [10].

Table 1 Examples of airspaces and 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 reported in [8], an airline could save 884 USD in overflight charges on a single flight from San Francisco to Frankfurt by deviating from the standard, most direct route; thus avoiding Canada's expensive charges. However, despite their obvious importance, the literature on overflight costs is scarce. The problem is presented in [15], but no solution approaches are discussed. The authors in [6] consider the linear/GCD model, but their optimal control algorithm approximates overflight costs by using FD instead of GCD. Similarly, in [19], several variants of overflight charges are introduced (including linear/GCD and flat-rate), but the optimization algorithm deals only with linear/FD costs. [17] introduces the linear/GCD model as well as the piecewise-constant/FD model, but their heuristic solution method allows no *à posteriori* quality assessment.

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 framework for classification of existing overflight cost models. Furthermore, we introduce efficient algorithms to handle a problem variant that is one of the most relevant in practice. The paper is structured as follows. In Section 2.1, we formally introduce the SPPCC, which models the problem of flight trajectory optimization with overflight costs. In Sections 3 and 4, we focus on one of the most important variants of the SPPCC. We present

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

both the Two Layer Dijkstra Algorithm, a polynomial, exact algorithm; as well as two novel cost-projection techniques that reduce the problem (heuristically) to a standard shortest path problem. In Section 5 we present computational results on real-world data.

2 Problem Description

This section gives a formal definition of the shortest path problem with crossing costs. For consistency with the literature, we will first introduce a general, NP-hard version and then proceed for the remainder of the paper with a polynomially solvable version, which is relevant for the applicatons in aircraft (and intermodal) routing that we have in mind.

2.1 The General SPPCC

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

Let $\mathcal{R} = \{R_1, \ldots, R_k\} \subseteq 2^A$ be a family of sets of arcs, not neccessarily disjoint, 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 $\mathcal{R}$. $\mathcal{R}^F$ represents the regions where every arc belonging to a solution path is important for computing the costs. For regions in $\mathcal{R}^G$, only the nodes used to enter and exit the region are relevant. For every $R \in \mathcal{R}$, let $f_R : [0, \infty) \to [0, \infty)$ be a non-decreasing function. 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:

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

▶ **Definition 1.** The *shortest path problem with crossing costs* (SPPCC) is defined as follows:
Input: Directed graph $D = (V, A)$, arc weights $\varphi : A \geq \mathbb{R}_+$, regions $\mathcal{R} \subseteq 2^A$, crossing costs γ, nodes $s, t \in V$.
Objective: 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 2.** 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 weights 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

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

cost of the example path is thus:

$$\begin{aligned} c(p) =& \sum_{a \in p} \varphi_a + \gamma_{R^1}(p) + \gamma_{R^2}(p) + \gamma_{R^3}(p) + \gamma_{R^4}(p) + \gamma_{R^5}(p) + \gamma_{R^6}(p) \\ =& 88 + f_{R^1}(9) + f_{R^2}(4) + f_{R^3}(7) + f_{R^4}(10) + f_{R^5}(10) + 0 = 158. \end{aligned}$$

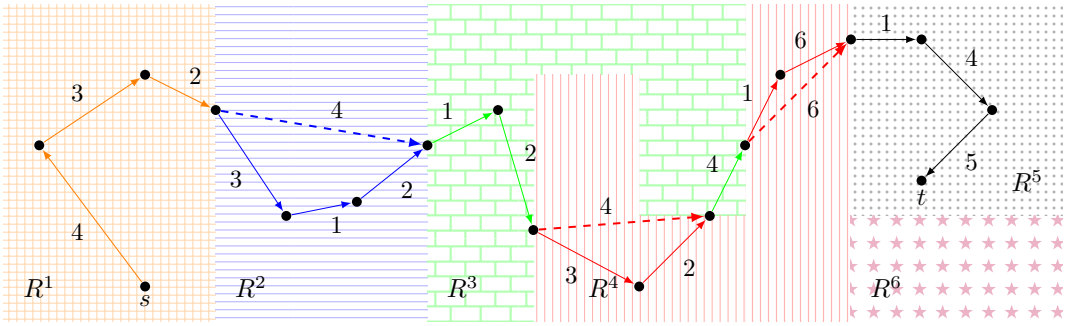

Figure 1 Example of an SPPCC instance with six regions. Arcs are labeled with the distances d_a between their endpoints, and the (s,t)-path p is represented by the continuous arrows. Dashed arrows represent distances between nodes that are not connected by an arc in the path.

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, multiplied by a wind distortion that will be explained in the next subsection. 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.

The five important airspace cost models outlined in Section 1 induce several different variants of the SPPCC. An extensive analysis of their complexity can be found in [5]. In this paper, we will focus on a variant which is of particular interest in our application.

2.2 The SPPCC With Wind Influence

In the remainder of the paper, we will focus on a case that best matches our application and turns out to be polynomial. We assume that $\mathcal{R}^G = \mathcal{R}$, $\mathcal{R}^F = \emptyset$, and that f_R is linear for all $R \in \mathcal{R}$. This allows to model the overflight fees used, for example, in Canada, USA, and most European states. We also assume that the regions in R are pairwise disjoint. This is natural, since usually there is a one-to-one correspondence between cost airspaces and countries, and the latter do not overlap. What about the weather? Of course, overflight fees do not depend on the weather, but the time needed to fly between two nodes (and

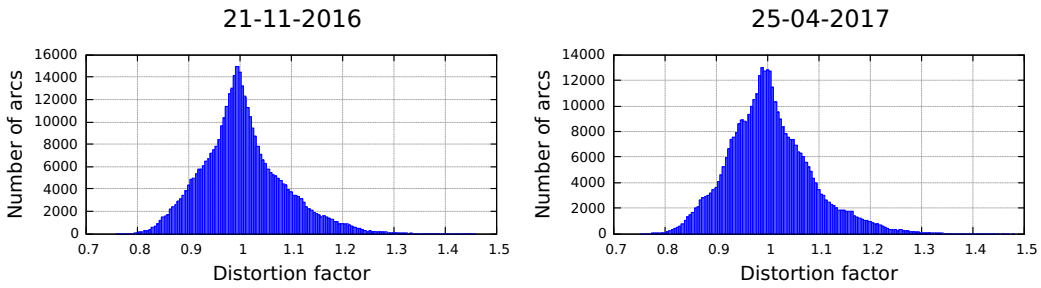

Figure 2 Typical distributions of the weight distortion factors ω_a over all arcs, for two different weather prognoses (November 2016 on the left, April 2017 on the right).

thus the corresponding fuel consumption) varies greatly as the wind speed and direction change with time, see [4] for a more detailed discussion of this phenomenon. Luckily, these tow cost factors can be separated in such a way that we can deal with crossing costs in *in a preprocessing step independently of the weather*, and with the wind *at query time* by multiplying each arc weight φ_a with an appropriate *distortion factor* ω_a. Figure 2 tries to provide some intuition on the distribution of these factors in our application. Motivated by this data, we will assume $\omega_a \in [0.7, 1.4]\ \forall a \in A$ throughout the paper.[1] We refer to the resulting variant of the SPPCC as SPPCC-W:

▶ **Definition 3.** The *shortest path problem with crossing costs and wind influence* (SPPCC-W) is defined as follows:
Input at preprocessing time: Directed graph $D = (V, A)$, arc weights $\varphi : A \geq \mathbb{R}_+$, regions $\mathcal{R} \subseteq 2^A$, crossing costs γ.
Input at query time: Nodes $s, t \in V$, distortion factors $\omega : A \to [0.7, 1.4]$.
Objective: Compute an (s, t)-path p in D that minimizes the cost function

$$c(p) = \sum_{a \in p} \omega_a \varphi_a + \sum_{R \in \mathcal{R}} \gamma_R(p). \tag{2}$$

3 The Two-Layer-Dijkstra Algorithm

We introduce in this section the *Two-Layer-Dijkstra Algorithm* (2LDA), which solves the SPPCC-W to optimality in polynomial time. Recall that $\mathcal{R}^G = \mathcal{R}$, that is, for each region, crossing costs are 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). The main difficulty is that two different paths crossing a region and using the same nodes to enter and leave it incur the same crossing costs. It can be overcome by considering a *coarse* graph on a subset of the nodes of the original (*fine*) graph, whose arcs correspond to shortest paths traversing a region in the fine graph. We run a Dijkstra algorithm on the coarse graph, determining arc costs on-the-fly by repeatedly using Dijkstra's algorithm on the fine graph. Finally, from the optimal path in the coarse graph, the minimum-cost path in the fine graph is reconstructed. We recall that, in Section 2.2, we assumed that the arc weights are multiplied with a distortion factor after preprocessing.

[1] We do not use these bounds explicitly, but the knowledge that ω_a is always "close to 1" is an important motivation for our algorithms in Section 4.

Since the 2LDA does not have a preprocessing phase, we will ignore distortion in this section, without loss of generality.

Let p be a path in D, whose intersections with a region $R \in \mathcal{R}$ are the subpaths $p_1^R, \ldots, p_r^R$. By the assumption of linearity of all crossing cost functions, there exists $\alpha_R > 0$ such that the crossing costs of p in R can be written as $\gamma_R(p) = \alpha_R \cdot \big(d(t(p_1^R), h(p_1^R)) + \cdots + d(t(p_r^R), h(p_r^R))\big)$. Recall that $t(\cdot)$ and $h(\cdot)$ denote a path's first and last nodes, or an arc's tail- and head nodes.

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 exit R as follows:

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

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

where $\delta^-(v)$ is the set of arcs incident to v and $\delta^+(v)$ is the set of arcs going out from v. $V^{\text{entry}}(R)$ 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 region $R_{\bar{a}}$ 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. Let $D|_{R_{\bar{a}}}$ denote the subgraph of D induced by $R_{\bar{a}}$. 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 weights φ. 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:

Phase 1 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 Phase 2, 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 weights φ. Finally, all such subpaths are concatenated to return an (s, t)-path in D. See Appendix A.1 for a formal description.

▶ **Theorem 4.** *The Two-Layer-Dijkstra Algorithm returns the optimal solution to SPPCC-W.*

Proof. 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 does $O(|V|^2)$ 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:

▶ **Corollary 5.** *SPPCC-W can be solved in polynomial time.*

4 The Cost Projection Method

While polynomial, the 2LDA is too slow for practical purposes. This motivates the development of fast heuristics. For this purpose, we introduce what we call the *cost projection problem*. It is based on the idea of replacing crossing costs in a preprocessing phase with shadow arc weights x_a and adding these to the original arc weights φ_a, thus resulting in a related SPP instance. Intuitively, the goal is to define the arc shadow weights in such a fashion that the path hierarchy (with respect to total costs) is preserved after the cost transformation. Formally:

▶ **Definition 6.** The *cost projection problem (CPP)* is the following:
Input: SPPCC-W instance (see Definition 3).
Objective: In a preprocessing phase (i.e., without knowledge of s, t and ω), construct *projected costs* $x : A \to \mathbb{R}_+$ so that, for any two paths p, q in D, it holds:

$$\sum_{a\in p} \omega_a\varphi_a + \sum_{R\in\mathcal{R}} \gamma_R(p) \leq \sum_{a\in q} \omega_a\varphi_a + \sum_{R\in\mathcal{R}} \gamma_R(q) \Leftrightarrow \sum_{a\in p} (\omega_a\varphi_a + x_a) \leq \sum_{a\in q} (\omega_a\varphi_a + x_a).$$

It is clear that, if we could successfully solve the CPP, then the SPPCC-W could be reduced to a simple SPP, and then solved very efficiently. Unfortunately, due to the enormous number of potential paths, the non-linear nature of the crossing costs, and the non-deterministic properties of the distortion factors ω, the CPP does not have a feasible solution in general. Some progress can be made by observing that, for our purposes, it would suffice if the optimal path in the original SPPCC-W instance were mapped to the optimal path in the transformed SPP instance. However, this is an equally unrealistic goal. We will therefore try to construct projected costs in such a way that the path found by solving the transformed SPP instance to optimality has a small optimality gap w.r.t. the original SPPCC-W instance. We suggest to do this by focussing on the subpaths in each region that are likely to be contained in the optimal SPPCC path.

▶ **Definition 7.** For each region, we define a set of *good paths*: Given $R \in \mathcal{R}, r \geq 1, u \in V^{\text{entry}}(R)$ and $v \in V^{\text{exit}}(R)$, we say that a (u, v)-path p is *r-good* if

$$\sum_{a\in p} d_a \leq r \cdot d(u, v). \tag{3}$$

We denote the set of r-good paths in R by $\mathcal{G}_r^R$. For any subset $\mathcal{G} \subseteq \mathcal{G}_r^R$, the subset of arcs in R used by at least one path in $\mathcal{G}$ is denoted by $A(\mathcal{G})$.

Intuitively, since arc weights φ are defined as a constant multiple of arc lengths, a path is good if it is not much longer than the theoretically shortest possible path. Recalling the assumption we made in Definition 3 on the possible ranges for the distortion factors ω_a, it is clear that the shortest paths within an airspace at query time are likely to be good paths, after a suitable choice of the parameter r (see Section 5). Finally, note that the size of $\mathcal{G}_r^R$ is exponential in the number of arcs.

We propose two approaches for solving the CPP.

4.1 Averaging Unit Costs: A Combinatorial Cost Projection Method

The first method we discuss is purely combinatorial and very simple, we call it *Averaging Unit Costs* (`AUC`). The idea behind the approach has the following motivation:

Suppose all paths in $\mathcal{G}_r^R$ in a region R are pairwise disjoint. A path $p \in \mathcal{G}_r^R$ has length $d(p) = \sum_{a \in p} d_a$ and costs $\gamma_R(p)$. A natural way to define projected costs on an arc $a \in p$ is to do so proportionally to its length:

$$x_a^{R,p} := d_a \cdot \frac{\gamma_R(p)}{d(p)}. \tag{4}$$

As a consequence of regions being disjoint, an arc $a \in A$ belongs to exactly one region $R \in \mathcal{R}$. Hence, to ease notation, we omit the index R in $x_a^{R,p}$. Since, in addition, all paths in $\mathcal{G}_r^R$ are disjoint, projected costs (4) for each arc in a good path are unique. Particularly, for each path $p \in \mathcal{G}_r^R$, there holds

$$x(p) = \sum_{a \in p} x_a^p = \gamma_R(p). \tag{5}$$

If (5) were to hold for all paths in all regions, then the CPP would be solved. However, this assumption is not realistic. In practice, an arc $a \in \mathcal{C}^R$ is usually contained in more than one good path. Furthermore, as noticed above, the size of $\mathcal{G}_r^R$ is exponential, making the iteration over all good paths impractical. For that reason, we restrict ourselves to a polynomially-sized subset $\bar{\mathcal{G}}_r^R \subset \mathcal{G}_r^R$ (a possible way of constructing $\bar{\mathcal{G}}_r^R$ is presented in Section 5), and define the final projected costs x_a of an arc a as the average over the projected costs (4) of a derived from all paths $p \in \bar{\mathcal{G}}_r^R$ covering it. Formally:

Let $a \in A(\bar{\mathcal{G}}_r^R)$, and let $\mathcal{G}_a^R \subseteq \bar{\mathcal{G}}_r^R$ be the subset of good paths in $\bar{\mathcal{G}}_r^R$ containing a. Note that $\mathcal{G}_a^R \neq \emptyset$, since $a \in A(\bar{\mathcal{G}}_r^R)$. Then, the final projected cost x_a of a is defined as

$$x_a := \frac{\sum_{p \in \mathcal{G}_a^R} x_a^p}{|\mathcal{G}_a^R|}, \tag{6}$$

Note that at this point, projected costs x_a have been defined only for arcs $a \in A \cap A(\bar{\mathcal{G}}_r^R)$, i.e., all arcs covered by at least one good path in our set. By construction, the optimal path of the SPPCC-W instance is likely to be fully or mostly contained in this set. For that reason, we assign projected costs that are higher than average to the other arcs. We define $m_R := \max_{a \in A(\bar{\mathcal{G}}_r^R)} \frac{x_a}{d_a}$. That is, m_R is the maximum cost-distance ratio in R. Then, for $a \in A \backslash A(\bar{\mathcal{G}}_r^R)$, we define $x_a := m_R \cdot d_a$.

A computational evaluation of the `AUC` approach is presented in Section 5.

4.2 Bound Underestimation: An LP-Based Cost Projection Method

In this section, we present linear programming (LP) based method for computing projected costs that we call `BOUND`. We use the following formulation:

$$\text{(LP-B)} \qquad \begin{aligned}
\min \quad & \sum_{a \in R} x_a && && && \text{(7a)} \\
\text{s.t.} \quad & x(p) && \geq \eta \cdot \gamma_R(p) && \forall p \in \mathcal{G}_r^R && \text{(7b)} \\
& x_a/d_a && \geq \alpha && \forall a \in R && \text{(7c)} \\
& x_a/d_a && \leq \beta && \forall a \in R && \text{(7d)} \\
& \beta && \leq \tau \cdot \alpha && && \text{(7e)} \\
& \alpha\,,\, \beta && \geq 0 && && \text{(7f)}
\end{aligned}$$

where $1 > \eta > 0$, $\tau > 0$ are parameters. The heart of (LP-B) is given by constraints (7b),

which can be seen as a relaxation of equality (4). For a good path it ensures that the total projected costs are not allowed to underestimate the corresponding crossing costs by much. Combined with the objective function (7a), it forces crossing costs and corresponding projected costs to be close to each other. Constraints (7c)-(7f) are meant to keep unit costs within reasonable bounds, thus avoiding projected costs that are negative or very close to zero. Note that the number of constraints (7b) is exponential. However, we have the following:

▶ **Proposition 8.** (LP-B) *can be solved in pseudo-polynomial time.*

Proof. Given $x \in \mathbb{R}^R_+, \alpha, \beta > 0$, our goal is to show that we can separate over all constraints of (LP-B) in pseudo-polynomial time. This reduces to separating over constraints (7b), since all others are polynomial in number. Given an entry-exit pair $(u, v) \in V^{\text{entry}}(R) \times V^{\text{exit}}(R)$, we know that the right-hand side of (7b) will be the same for all (u, v)-paths in R, say γ^R_{uv}. Thus, we would like to know whether there exists an (u, v)-path in $\mathcal{G}^R_r$ such that $\sum_{a\in p} x_a < \gamma^R_{uv}$. We recall that, by definition, $p \in \mathcal{G}^R_r$ if and only if $\sum_{a\in p} d_a \leq r \cdot d(u, v)$. It is thus clear that (since there is a polynomial number of entry-exit pairs), solving the separation problem is equivalent to solving an instance of the resource constrained shortest path problem (RCSPP) with costs x, resources d and resource limit $r \cdot d(u, v)$ for each entry-exit-pair (u, v) of a region. It is well-known [12] that the RCSPP can be solved in pseudo-polynomial time, thus completing the proof. ◀

5 Computational Results

In this section, we evaluate the performance of the algorithms on real-world instances.

All algorithms were implemented in C++ and compiled with GCC 5.4.0. All computations were performed on machines with 132 GB of RAM and an Intel(R) Xeon(R) CPU E5-2660 v3 processor with 2.60GHz and 25.6 MB cache. The computation of projected costs was carried out in parallel using 28 threads. All other computations were carried out in single-thread mode. We used Gurobi 7.0.2 as our LP-solver.

5.1 Instances

All instances used in our computations correspond to real-world data, provided by Lufthansa Systems GmbH & Co. KG. This data comprises the airway network graph, the weather prognoses, and the set of overflight charges.

For our queries, we use a list of 4917 Origin-Destination (OD) pairs for which a commercial aircraft route exists and such that the optimal route does not leave the European airspace for any of the given weather prognoses. The reason for selecting these particular instances is that all airspaces involved define overflight fees using linear functions and the great-circle-distance (see Table 1), which is the variant that we considered in this paper.

The directed graph D corresponds to an horizontal layer of the airway network at $30,000$ feet altitude, which is a flight level common for cruising. It consists of $97,534$ nodes and $136,903$ arcs. Moreover, we consider three weather prognoses corresponding to January 6th 2015, November 21st 2016, and April 25th 2017. We identify them by the names *Jan*, *Nov*, and *Apr*, respectively. These prognoses contain the wind information defining the distortion factor ω_a for each arc $a \in D$ (see Figure 2). We use overflight charges and average fuel costs corresponding to an Airbus 320 aircraft. The fuel cost used is 1.73USD/km.

To gain some perspective on the size of the regions used for our computations, in Table 3 we describe a few properties of some of the instances used.

Table 3 The first three columns show the number of arcs, entry- and exit nodes corresponding to a few European airspaces. The fourth column represents the cost per km used for the airspaces' overflight charges, i.e., the factor α_R defined in Section 3.

	entry nodes (#)	exit nodes (#)	arcs (#)	cost per km (USD)
Germany	1938	1906	6361	1.58
Italy	1078	1258	7361	1.82
Norway	1705	1636	6863	1.08
Sweden	4922	4905	9561	1.39
USA	5689	5341	88782	0.22

5.2 Computational Results

The two cost-projection methods presented rely on sets of good paths $\bar{\mathcal{G}}_r^R$ and $\mathcal{G}_r^R$, respectively. For comparability and computational efficiency, in this section we restrict both algorithms to use a single set $\bar{\mathcal{G}}_r^R$, which we define as follows for every region $R \in \mathcal{R}$.

We start with an empty set $\bar{\mathcal{G}}_r^R$. For every entry-exit pair (u, v) in R and for every $a \in R$, we compute the φ-shortest (u, v)-path in R that contains the arc a (easily obtainable by concatenating two φ-shortest paths and a), and insert it to $\bar{\mathcal{G}}_r^R$ only if it is cycle-free and it satisfies (3). Furthermore, for every (u, v) we store only the 10 shortest paths obtained in this way. For our experiments, we use $r = 1.2$.

We also set the parameters in (LP-B) to $\eta = 0.98$ and $\tau = 1.1$.[2]

While the LP models for all European airspaces could be solved in a reasonable time by Gurobi, and we only consider OD-pairs crossing these regions, the long-term goal is to use these methods worldwide. For that reason, we also compute projected costs for the USA airspace, which is particularly challenging, due to its size (see Table 3). In particular, even after restricting the set of good paths used as described above, the number of constraints (7b) exceeds 58 million. In conjunction with approximately $88,000$ variables, this leads to Gurobi quickly running out of memory. Inspired by the framework presented in Section 4.2, we implemented a rudimentary constraint generation algorithm which stores all constraints in memory, then iteratively adds subsets of those that are violated to the model and solves it, until no constraints are violated. This simple approach makes it possible to solve the LP model for USA. For completeness, we ran this constraint generation approach on all instances.

Figure 3 shows the preprocessing times needed to compute projected costs in each European region. As could be expected, the LP-based approach is always slower than the combinatorial `AUC` method. In our testing environment, we can handle the size of the generated models using the `BOUND` method. In nearly all cases, the latter is easier to solve in a single run than using constraint generation. However, for a couple of large regions (Norway and USA), the reverse is true.

Using constraint generation for the USA region enables us to solve the corresponding model. To do so, Gurobi requires 63.6GB of memory and it takes $8,696$s to find an optimal solution. Only 27.9 million constraints have to be added to the model, since all others never get violated.

[2] This seemingly deliberate selection of parameters yielded the best results among some tested variants. However, there is definitely plenty of room for further parameter optimization.

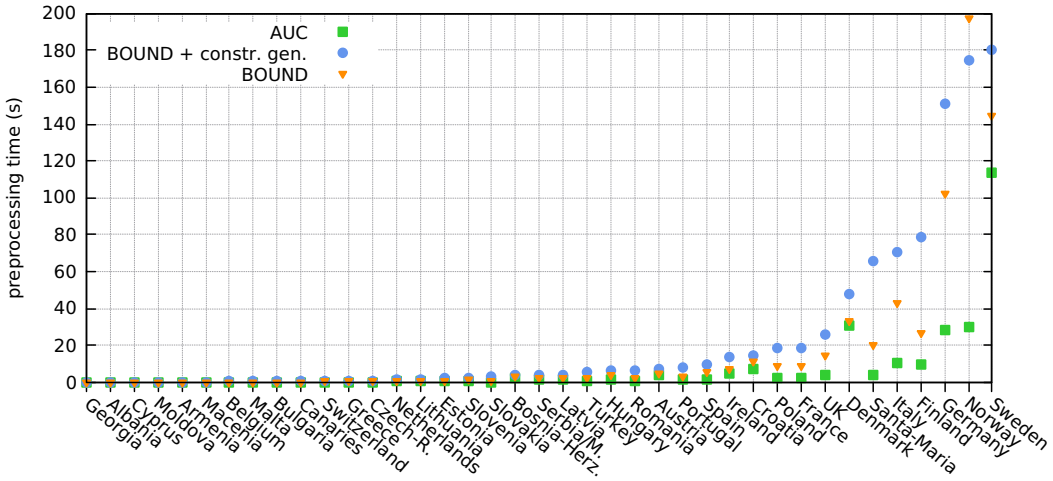

Figure 3 Time needed to compute projected costs using different methods on all airspaces

Finally, in Table 4 we measure the quality and efficiency of our algorithms. Both cost-projection heuristics find solutions with a very small optimality gap. In particular, there is a large percentage of instances for which they find the optimal solution. Furthermore, the speedup of more than a factor of 350 with respect to 2LDA is remarkable.

Table 4 For each weather prognosis, we present the time needed by 2LDA. For both heuristics, the average error, relative error, and speedup with respect to 2LDA is shown. Column *exact* lists the percentage of perfect hits, i.e., instances for which the error is zero.

	AUC				BOUND			
	abs err (USD)	rel err (%)	exact (%)	speedup (×)	abs err (USD)	rel err (%)	exact (%)	speedup (×)
Jan	34.84	1.03	35.27	382.63	4.83	0.16	36.89	380.00
Nov	34.60	1.03	34.29	362.15	4.73	0.16	36.81	363.48
Apr	35.44	1.06	35.57	361.63	4.70	0.16	37.60	364.44

6 Conclusions

In this paper, we introduced the SPPCC and the SPPCC-W, which models the flight trajectory optimization problem under influence of overflight fees and wind. We presented the Two-Layer-Dijkstra Algorithm for solving the SPPCC-W to optimality in polynomial time. We also developed a novel cost-projection method and suggested two computational procedures. In the corresponding computations, we showed that they achieve approximation errors well below 1%, while yielding a speedup of over 350 with respect to the Two-Layer-Dijkstra Algorithm.

7 Acknowledgements

We thank Lufthansa Systems GmbH & Co. KG for providing us with the data used in this paper, as well as for the many fruitful discussions.

References

1 Federal Aviation Administration. Overflight fees, 2016. [Online; accessed 8-April-2016]. URL: https://www.faa.gov/air_traffic/international_aviation/overflight_fees/.

2 ASECNA. Air navigation services charges, 2016. [Online; accessed 8-April-2016]. URL: http://www.ais-asecna.org/pdf/gen/gen-4-2/00gen4-2-01.pdf.

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, January 2014.

4 Marco Blanco, Ralf Borndörfer, Nam Dung Hoang, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. Solving time dependent shortest path problems on airway networks using super-optimal wind. In *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54, 2016. epub ahead of print. doi:10.4230/OASIcs.ATMOS.2016.12.

5 Marco Blanco, Ralf Borndörfer, Nam Dung Hoang, Anton Kaier, Thomas Schlechte, and Swen Schlobach. The shortest path problem with crossing costs. Technical Report 16-70, ZIB, Takustr.7, 14195 Berlin, 2016.

6 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(5):1267–1277, July 2013.

7 Hajo Broersma, Xueliang Li, Gerhard Woeginger, and Shenggui Zhang. Paths and cycles in colored graphs. *Australasian journal of combinatorics*, 31:299–311, 2005.

8 Susan Carey. Calculating costs in the clouds. *The Wall Street Journal*, March 2007.

9 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

10 EUROCONTROL. Establishing route charges, 2016. [Online; accessed 8-April-2016]. URL: http://www.eurocontrol.int/articles/establishing-route-charges.

11 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.

12 Refael Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, 1992.

13 Refael Hassin, Jérôme Monnot, and Danny Segev. Approximation algorithms and hardness results for labeled connectivity problems. *Journal of Combinatorial Optimization*, 14(4):437–453, 2007.

14 Stefan Irnich and Guy Desaulniers. *Column Generation*, chapter Shortest Path Problems with Resource Constraints, pages 33–65. Springer US, Boston, MA, 2005.

15 Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. Operations. In Cynthia Barnhart and Barry Smith, editors, *Quantitative Problem Solving Methods in the Airline Industry*, volume 169 of *International Series in Operations Research & Management Science*, pages 283–383. Springer US, 2012.

16 Dean H. Lorenz and Danny Raz. A simple efficient approximation scheme for the restricted shortest path problem. *Oper. Res. Lett.*, 28(5):213–219, June 2001.

17 Robert L. Schultz, Stephen G. Pratt, and Donald A. Shaner. Four-dimensional route planner, March 27 2003. WO Patent App. PCT/US2002/029,474.

18 Christian Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4):45:1–45:31, March 2014.

19 Banavar Sridhar, Hok Kwan Ng, Florian Linke, and Neil Y. Chen. Impact of airspace charges on transatlantic aircraft trajectories. In *15th AIAA Aviation Technology, Integration, and Operations Conference*. American Institute of Aeronautics and Astronautics, jun 2015. URL: https://doi.org/10.2514/6.2015-2596, doi:10.2514/6.2015-2596.

A Appendix

Here, we give a detailed description of the Two-Layer-Dijkstra Algorithm presented in Section 3.

A.1 The Two-Layer-Dijkstra Algorithm

The complete version of the Two-Layer-Dijkstra Algorithm can be found in Algorithm 1. We use the same notation as in Section 3. 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.

Algorithm 1 Two-Layer-Dijkstra Algorithm for SPPCC

Input: D, s, t, φ, disjoint regions $\mathcal{R}$ and linear 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*
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**
    $\ell_{\bar{u},\bar{v}} \leftarrow$ 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**
  $p \leftarrow$ ShortestPath$(pred[u], u, D, \varphi) \cup p$ ▷ *reconstruct path in D recursively*
return p