Finding Extreme Supported Solutions to the Bi-Objective Shortest Path Problem

Bachelor Thesis[1]

July 2, 2017

Adriaan Molendijk[2]

Supervisor: T. Breugem MSc
Co-Reader: Dr. T.A.B. Dollevoet

[1]Erasmus University Rotterdam, Erasmus School of Economics, Econometrics and Operations Research
[2]Student id: 409865

Abstract

The extreme supported solutions to the one-to-all bi-objective shortest path problem are analyzed. We start with the implementation of a ratio-labeling algorithm which obtains all these solutions in $\mathcal{O}(N(m+n\log n))$ steps, where N is the total number of extreme supported solutions and n and m are the number of nodes and arcs in the graph respectively. Next we consider a setting where the input graph has the property of being acyclic, and propose a $\mathcal{O}(Nm)$ labeling algorithm to solve this problem. We conclude this thesis by showing how this same algorithm can easily be adapted to handle the scenario where either one or both of the criteria are of the bottleneck type. Due to the nature of the extreme supported solutions under this criterion function, a polynomial time algorithm with a complexity of $\mathcal{O}(m^2)$ steps is found in this case.

Keywords: Bi-objective, shortest path, extreme supported, acyclic, bottleneck

Acknowledgements

This thesis was written during the summer of 2017, as part of the Bachelor's degree programme Econometrics and Operations Research at the Erasmus University Rotterdam. I would like to express my gratitude to my supervisor T. Breugem for his guidance and criticism throughout the writing process. His valuable suggestions have greatly helped shape this thesis.

Contents

1 Introduction

In a combinatorial optimization problem a decision maker (DM) aims to find a solution among a finite or countably infinite set of solutions, that minimizes or maximizes some criterion function. A multi-objective combinatorial optimization problem (MOCO) is an optimization problem where, unlike in a regular combinatorial optimization problem, several usually conflicting objectives or criteria are considered. The multi-objective shortest path problem (MSP) is an example of such a problem. It is a variation of the shortest path problem (SP), in which there is not one but multiple types of cost associated to the arcs in a network. The MSP problem where only two types of cost are considered is called the bi-objective shortest path problem (BSP), and this problem has probably received most attention among the MSP problems. The BSP problem can be split up into one-to-one or one-to-all, depending on whether the goal is to find the shortest path from an origin node to a destination node or from an origin node to all other nodes in the network.

In many applications regarding transportation there are several factors that need consideration, such as traveling time, traveling distance and cost. One example of such an application is the transportation of hazardous materials as discussed in Erkut and Alp (2007), for which the paths of interest do not only have to be short, but also have to minimize the inflicted damage in the neighborhood of an accident in case of leakage. Another application mentioned in Guerriero and Musmanno (2001) arises in the allocation of new highways. Highways have to be as short as possible to allow fast traversal from A to B, while maintaining the accessibility to surrounding cities. The trade-off between the different types of cost is the major issue that DMs face when they have to choose some solution to these problems, as the solution will most likely not be optimal for all criteria.

Applications of the BSP problem are not limited to being strictly practical. Take the bi-objective knapsack problem, for instance. It turns out that despite being a completely different optimization problem on its own, the problem has a similar structure to that of the BSP problem. In fact, in Captivo et al. (2003) they show that this problem can be converted to a BSP problem over an acyclic network, and then continue to solve this problem by means of a labeling procedure. Other theoretical applications are given in Hallam et al. (2001), in which they show that BSP algorithms can be used to minimize the detection of mobile objects through a field of sensors, and in Arkin et al. (1991), where BSP problems are even considered in a geometric setting.

In this thesis we analyze the solutions to the one-to-all BSP problem which have the property of being so-called extreme supported. While this thesis only looks at the solutions with that specific property, these solutions might be sufficient from a DM's perspective, as the total number of solutions could be overwhelming (see intractability BSP, Serafini (1987)). Bazaraa et al. (2013) show that if the utility function of the decision maker is quasi-convex, the optimal solution to the BSP problem is among the extreme supported solutions. Also, there exist problems where optimality for one of the extreme supported solutions is guarantied, see for example the minimum cost-reliability ratio path problem evaluated in Ahuja (1988). Finally, the supported solutions can also be used to find the non-supported solutions

(as in two-stage approaches), or can function as a starting point in the identification of some compromise solution for the DM, see for example Current et al. (1990).

This thesis is split up into two main parts. In the first part the BSP problem is considered, and in the second part the acyclic BSP problem (ABSP) is considered. The first part starts with Section 2, which presents a literature review of the BSP problem. In Section 3 a description of the BSP problem is given, in which we elaborate on what it means for a path to be an extreme supported solution. Next, in Section 4 the ratio-labeling algorithm from Sedeño–Noda and Raith (2015) is presented and Section 5 describes the results from implementing that algorithm. The rest of the sections in this thesis cover the second part. In Section 6 the ABSP problem is introduced and in Section 7 a labeling algorithm to obtain all extreme supported points to this problem is proposed. The results from implementing this algorithm are described in Section 8, and the ABSP problem with at least one bottleneck objective function is evaluated in Section 9. Finally, in Section 10 we conclude our findings.

2 Literature Review

Multi-objective optimization, often labeled as vector optimization, has received increasing interest in the past decades due to its potential application to real world decision-making problems. Examples include ones like the multi-objective spanning tree problem (e.g. Knowles and Corne (2001)), and the multi-objective traveling salesman problem (e.g. Lust and Teghem (2010)). For a brief survey on the subject we refer to Ehrgott and Gandibleux (2000), where they give an overview of the literature in the field of multi-objective combinatorial optimization.

Within the single-objective combinatorial optimization problems, the shortest path problem is among the best studied problems in this field. Polynomial time algorithms exist for solving the SP problem, given there are no cycles in the considered graph with negative cost. Among those algorithms developed for solving the SP problem, the two most famous ones are Dijkstra's algorithm (see Dijkstra (1959)) and Bellman-Ford's Algorithm (see Bellman (1958)), which have time complexities of $\mathcal{O}(m + n \log n)$ and $\mathcal{O}(nm)$ respectively. In case the graph has the property of being acyclic, linear time (i.e. $\mathcal{O}(n + m)$) algorithms exist to solve the one-to-all SP problem, see for example Cormen (2009).

The BSP problem, however, is computationally difficult. Even though only two types of objectives are considered, it is shown by Serafini (1987) that the problem is $\mathcal{NP}$-hard and intractable, that is, the number of solutions to the BSP problem could be exponential in the number of nodes in the network. In Müller-Hannemann and Weihe (2006) they show that this result still holds, even when only two types of ratios between the two criteria are allowed.

Despite its potential difficulty, there is a wide range of solution approaches available for solving the BSP problem, with the two main approaches being label correcting (e.g. Skriver and Andersen (2000)), and label setting (e.g. Hansen (1980)). Other solving approaches include ranking methods like the one in Climaco and Martins (1982), and two phase methods, such as the one introduced in Mote et al. (1991). Raith and Ehrgott (2009) do an extensive computational comparison between the different approaches for solving the BSP problem, and show that all these approaches are competitive depending on the type of network that is operated on.

Recently, Sedeño–Noda and Raith (2015) introduced two labeling algorithms for obtaining all extreme supported solutions to the BSP problem. The first one for obtaining the extreme supported solutions to the one-to-one BSP problem, and the other one for obtaining those to the one-to-all BSP problem. Other parametric approaches for obtaining the extreme supported solutions are presented in Mote et al. (1991), in the phase one stage of a two phase method, and in Henig (1986), where also search methods are suggested in case the utility function of the DM is assumed to either be quasi-concave or quasi-convex.

Most of the literature regarding the BSP problem concerns objective functions that are of the additive type, but also other types of objective functions can be used. In Martins (1984) they present two algorithms for solving the BSP problem where one of the objectives

is of the bottleneck type, and the other is of the min-sum, min-ratio or also the bottleneck type. One of these algorithms is later revised in Gandibleux et al. (2006), extending it to multicriterion path problems. In de Lima Pinto et al. (2009) they discuss tricriterion path problems where at least two of the objectives are of the bottleneck type.

3 Problem Description BSP Problem

In this section we present the problem description of the BSP problem. We start with introducing the classical shortest path problem and definitions regarding multi-objective combinatorial optimization problems, since most of the concepts in these problems are also used in the BSP problem.

3.1 Shortest Path Problem

In the shortest path problem we are given a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N} = \{1, \ldots, n\}$ is a set of nodes and $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ is a set of arcs. Also given are a function $c : \mathcal{A} \to \mathbb{R}$ assigning lengths to the arcs in the graph, and an *origin node* and *destination node* denoted by $s \in \mathcal{N}$ and $t \in \mathcal{N}$ respectively. The problem comes to finding a directed path between the nodes s and t for which the sum of the lengths of the arcs in that path is minimized.

We define a directed path[1] p as a sequence of alternating nodes and arcs, that is, $p = (i_1,(i_1,i_2),i_2,(i_2,i_3),\ldots,(i_{k-1},i_k),i_k)$, with $(i_m, i_{m+1}) \in \mathcal{A}$ for all $1 \leqslant m \leqslant k-1$, and $i_m \in \mathcal{N}$ for all $1 \leqslant m \leqslant k$. We call a path simple if it visits every node exactly once, that is, $i_{m_1} \neq i_{m_2}$ if $m_1 \neq m_2$, and write p_{ij} in case $i_1 = i$ and $i_k = j$ (we lose the subscripts if the path is clear from the context or the actual path sequence is not important). We denote $\mathcal{P} = \mathcal{P}_{st}$ for the set of all $s-t$ paths (likewise $\mathcal{P}_{ij}$ for the set of all $i-j$ paths). Finally, we extend the length function by defining the length of a path as the sum of the lengths of the arcs in that path, so $c(p) = \sum_{(i,j)\in p} c_{ij}$ for any path p. Hence we can formulate the SP problem as

$$\min_{p \in \mathcal{P}} \left\{ c(p) = \sum_{(i,j)\,\in\, p} c_{ij} \right\}.$$

3.2 MOCO Problems

A combinatorial optimization problem is a problem of the form

$$\min_{x \in \mathcal{X}} \big\{ f(x) \big\},$$

where $x = (x_1, \ldots, x_m)$ is some decision vector, $\mathcal{X} \subseteq \mathbb{R}^m$ is the feasible decision space, and $f : \mathcal{X} \to \mathbb{R}$ is a criterion function mapping a solution in the feasible decision space to a scalar value. A multi-objective combinatorial optimization problem differs from a regular combinatorial optimization problem, in a sense that the criterion function is now defined as a mapping $f : \mathcal{X} \to \mathbb{R}^k$, assigning $k \geqslant 2$ different criteria to a solution in the feasible decision space. The two problems also differ in terms of their *feasible objective space*

$$\mathcal{Y} = \{ f(x) : x \in \mathcal{X} \}$$

[1]We often consider it simply as a sequence of arcs, writing $(i,j) \in p$, $p = q \cup (i,j)$, etc., slightly disregarding the formal notation here as long as the meaning is clear from the context.

and in terms of their definition of optimality. In case a single-objective criterion function is used, a solution $x^* \in \mathcal{X}$ is optimal if $f(x^*) \leqslant f(x)$ for all $x \in \mathcal{X}$ (in the case of minimization). In case a multi-objective criterion function is used, however, there is the potential absence of an optimal solution (in all criteria), since a solution could be better in one criterion and worse in another with respect to another solution.

Yet we can still evaluate the solutions of MOCO problems using different notions of optimality, one of which is called *Pareto efficiency*. The concept of Pareto efficiency is based on a binary relation, and is defined as follows.

Definition 1. Let $x_1, x_2 \in \mathcal{X}$ be two solutions in the feasible decision space, then x_2 is said to be *dominated* by x_1 if

$$f(x_1) \leqslant f(x_2) \text{ and } f(x_1) \neq f(x_2),$$

that is, there in a strict improvement in at least one of the criteria without worsening the others.

We write $x_1 \prec x_2$ to indicate that x_1 and x_2 satisfy this relationship. A solution $x \in \mathcal{X}$ is called *efficient* if there does not exist another solution $\overline{x} \in \mathcal{X}$ for which $\overline{x} \prec x$.

Another notion of optimality is that of the supported solutions, for which we will only present the bi-objective case.

Definition 2. A solution $x \in \mathcal{X}$ in the feasible decision space is called *supported* if there exists a $\lambda \in [0, 1]$ such that this solution x is a solution to the weighted sum problem

$$g(\lambda) = \min_{x \in \mathcal{X}} \left\{ \lambda f^1(x) + (1 - \lambda) f^2(x) \right\}.$$

This definition essentially says that the solution x minimizes some convex combination of the two considered criteria over the feasible decision space $\mathcal{X}$. Supported solutions are relevant for several different reasons. First of all, the fact that supported solutions are guaranteed to be efficient (border cases are an exception) and the fact that their number is significantly less than the number of efficient solutions (e.g. Mulmuley and Shah (2000)), makes them particularly attractive in decision-making contexts. Secondly, they are useful from a practical standpoint, as they can be used in two phase methods to find all or part of the efficient solutions. Finally, there exist situations where only the supported solutions are required, since optimality for one of these solutions is guarantied, see for example the minimum cost-reliability ratio path problem evaluated in Ahuja (1988).

In order to illustrate the introduced concepts above, consider a routing problem where a truck has to travel from A to B. To each possible routing, there is an associated travel time T and cost C needed to perform the routing. Now suppose there are 5 such routings $A_1, \ldots, A_5$ (or alternatives for that matter) with costs $c(A_1), \ldots, c(A_5)$ for a DM to choose from, also shown in Figure 1.

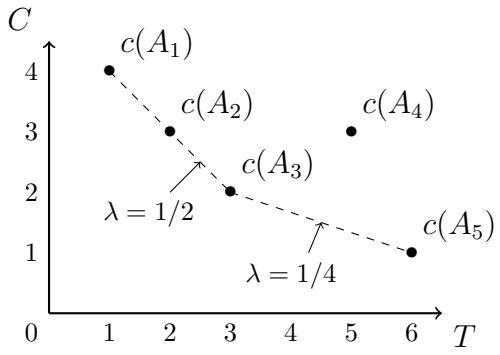

Figure 1: Travel times and costs of different alternatives to a routing problem. The values $\lambda = 1/2$ and $\lambda = 1/4$ indicate the values of λ associated to the slopes of the dotted lines.

If the DM prefers a short routing, he or she would choose alternative A_1, minimizing the travel time to perform the routing, while if he or she prefers a cheap routing he or she would choose alternative A_5, minimizing the cost to perform the routing. In order to illustrate the idea of Pareto efficiency consider alternatives A_3 and A_4. Since alternative A_3 has a travel time of 3 units and a cost of 2 units and alternative A_4 has a travel time of 5 units and a cost of 3 units, $A_3 \prec A_4$, and so if the DM were a rational agent he would never choose alternative A_4 over A_3. Now to illustrate the concept of supported solutions, we imagine a situation where the DM weighs both criteria equally heavy, i.e., $\lambda = 1/2$. In this case his cost preference is described by $1/2T + 1/2C$, for which all three alternatives A_1, A_2 and A_3 are supported. The same argument can be held for $\lambda = 1/4$ for alternatives A_3 and A_5.

3.3 Bi-Objective Shortest Path Problem

In the BSP problem we are given, just like in the SP problem, a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N} = \{1, \ldots, n\}$ is a set of nodes and $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ is a set of arcs. Also given are an origin node and destination node denoted by $s \in \mathcal{N}$ and $t \in \mathcal{N}$ respectively. The problem differs from the regular SP problem in the arc function $c : \mathcal{A} \to \mathbb{R}^2$, which now assigns two different criteria to the arcs in the graph. We keep the rest of the notation identical to that of the SP problem.

We define a directed cycle as any path of the form p_{ii} having at least one arc, and define the concatenation of two paths as $p_{i\ell} = p_{ij} \diamond p_{k\ell}$ given $j = k$. We use c^r_{ij} to indicate the r^{th} cost of arc $(i, j) \in \mathcal{A}$, $r = 1, 2$, and extend the cost functions by defining the cost of a path as the sum of the costs of the arcs in that path, that is, $c^r(p) = \sum_{(i,j)\in p} c^r_{ij}$ for any path p, and cost $r = 1, 2$. We can now formulate the BSP problem as

$$\min_{p \in \mathcal{P}} \left\{ c(p) = \left(\sum_{(i,j)\in p} c^1_{ij}, \sum_{(i,j)\in p} c^2_{ij} \right) \right\}.$$

For the BSP problem we make several assumptions on the graph $\mathcal{G}$, as often done in the literature (e.g. Martins and Dos Santos (1997)). The first assumption we make is that for

all nodes $i \in \mathcal{N} \setminus \{s\}$, $\mathcal{P}_{si} \neq \varnothing$, that is, there is at least one directed path from the origin node to all other nodes in the graph. This assumption is pretty reasonable, since if there is no directed path from s to i for some node $i \in \mathcal{N} \setminus \{s\}$, then this node i does not lie on any $s-t$ path, and thus it can be removed from the graph without changing the solution set $\mathcal{P}$. The second assumption we make is that for every cycle p_{ii} in $\mathcal{G}$ we have $0 \prec p_{ii}$, that is, it is always cheaper to not traverse a cycle in $\mathcal{G}$, since there is a strictly positive cost associated to one criterion and a non-negative cost to the other. This assumption prevents the BSP problem from being unbounded, because it eliminates the potential traversal of cycles in the graph (we will come back to this later). From a practical point of view this assumption also seems pretty reasonable. Let's say we are solving some routing problem, then it is not realistic for a truck to be driving in circles while the primary goal is to go as quickly from A to B.

The concepts first introduced in Section 3.2 apply naturally to the BSP problem. A path $p \in \mathcal{P}$ is said to be *efficient* if there does not exist another path $\overline{p} \in \mathcal{P}$ that dominates this path, $\overline{p} \prec p$. In case a path p is efficient we call its mapping $c(p)$ a *non-dominated point*. A path $p \in \mathcal{P}$ is called *supported* if there exists some $\lambda \in [0,1]$ such that this path p can be obtained by solving the weighted sum problem

$$w(\lambda) = \min_{p \in \mathcal{P}} \left\{ \lambda c^1(p) + (1-\lambda) c^2(p) \right\},$$

much like the definition of supported solutions for general bi-objective combinatorial optimization problems. In case a path p is supported we call its mapping $c(p)$ a *supported point*. We define $\mathcal{C}$ (feasible objective space) as the set of images associated to all the $s-t$ paths, that is, $\mathcal{C} = \{c(p) : p \in \mathcal{P}\}$, and denote $\text{conv}(\mathcal{C})$ for its convex hull. Since $\mathcal{C} \subseteq \text{conv}(\mathcal{C})$, we find that the mappings of all paths are located inside the set $\text{conv}(\mathcal{C})$. Of those points, all the supported points lie on the lower-left boundary of this set and all the non-supported points do not. Finally, recall that for any convex set, the extreme points are those points that do not lie on a line segment joining two points in this set. Hence, an *extreme supported* path differs in definition from a supported path, because its cost must also be an extreme point of the set $\text{conv}(\mathcal{C})$. In terms of the routing problem example from Section 3.2, alternatives A_1 and A_3 are extreme supported but alternative A_2 is not, because its cost is a convex combination of half the cost of alternative A_1 and half the cost of alternative A_3.

The problem addressed in this thesis is the computation of a single path for each extreme supported point to the BSP problem. In case one of these paths is associated to a non-dominated point, it is also simple, due to the following lemma.

Lemma 1. Let $p_{st} \in \mathcal{P}$ be a path associated to an extreme supported non-dominated point in the objective space of the BSP problem. Then p_{st} is simple.

Proof. Clearly. Take any path associated to an extreme supported non-dominated point that is non-simple, say $p_{st} = p_{si} \diamond p_{ii} \diamond p_{it}$. After removal of the cycle p_{ii} from that path, we obtain the path $\overline{p}_{st} = p_{si} \diamond p_{it}$ for which there is a cost reduction in one criterion without worsening the other, due to the second assumption on $\mathcal{G}$. But this contradicts the definition of a non-dominated point (there exists $\overline{p}_{st} \prec p_{st}$), so p_{st} must be simple. ■

4 Ratio-Labeling Algorithm BSP Problem

In this section we present the ratio-labeling algorithm (RLBSP) from Sedeño–Noda and Raith (2015), which we will use to compute all paths associated to extreme supported non-dominated points in the objective space of the one-to-all BSP problem. Sedeño–Noda and Raith (2015) make the observation that instead of finding all the supported $s-t$ paths, we can also compute all the supported $s-i$ paths for all nodes $i \in \mathcal{N} \setminus \{s\}$, obtaining a bi-objective shortest path tree problem (BST). The algorithm they then propose obtains all extreme supported paths by solving a transformation of the parametric program $g(\lambda)$ from Section 3.2 for this BST problem. It does so by first obtaining a tree that is lexicographically optimal, and next making several arc interchanges until the negative ratio of the so-called "reduced cost" of all arcs is infinite. The correctness of the algorithm is proved in the paper.

Before giving the exact algorithm, we will elaborate on some of the concepts used in the RLBSP. These concepts being the shortest path tree, the lexicographic minimum and the sets of neighbouring nodes. The *shortest path tree* $\mathcal{T}$ (rooted at s) of a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ with cost function $c : \mathcal{A} \to \mathbb{R}$ is a subset of the arcs, that is $\mathcal{T} \subseteq \mathcal{A}$, with $(i,j) \in \mathcal{T}$ if and only if $(i,j) \in \mathcal{A}$ is on the shortest $s-j$ path. Next, the *lexicographic minimum* is a minimum for a lexicographical order. The lexicographical order is an order such that given two sequences x and y with n elements, x comes before y if there is a $1 \leqslant k \leqslant n$ such that x_i is equal to y_i for all $1 \leqslant i \leqslant k-1$ and x_k is strictly smaller than y_k. Finally, we define $\delta^-(i) = \{j \in \mathcal{N} : (j,i) \in \mathcal{A}\}$ and $\delta^+(i) = \{j \in \mathcal{N} : (i,j) \in \mathcal{A}\}$ as the sets of nodes associated to the incoming and outgoing arcs for a node $i \in \mathcal{N}$ respectively.

The algorithm stores several labels. For each node $i \in \mathcal{N}$, $d_i = (d_i^1, d_i^2)$ denotes the distance of this node to the origin node in the current shortest path tree and Pred_i denotes the predecessor node of this node in the current shortest path tree. The reduced cost $\overline{c}_{ij} = (\overline{c}_{ij}^1, \overline{c}_{ij}^2)$ of an arc $(i,j) \in \mathcal{A}$ is calculated using $\overline{c}_{ij} = c_{ij} + d_i - d_j$, and represent the change in shortest $s-j$ path cost when replacing arc (Pred_j, j) by (i,j) in the shortest path tree. For all nodes $i \in \mathcal{N}$, the value θ_i denotes the minimum ratio $-\overline{c}_{ji}^1 / \overline{c}_{ji}^2$ of incoming arcs (j,i). The value CPred_i is used to keep track of the predecessor node associated to this ratio and $\hat{c}_i = (\hat{c}_i^1, \hat{c}_i^2)$ is the reduced cost for this node $i \in \mathcal{N}$. Finally, the initial shortest path tree of $\mathcal{G}$ is calculated by applying a variation of Dijkstra's algorithm, where labels are updated if there is an improvement in the first criterion or a tie in the first criterion and an improvement in the second. Note that this tree is a solution to the $\text{lexmin}\{c(p) : p \in \mathcal{P}\}$ problem.

The algorithm makes great use of a heap storing labels of the form $L = (\theta_i, \hat{c}_i^2, i)$, for each node $i \in \mathcal{N}$. A heap is a special case of the tree data structure (e.g. Eck (2006)), where all the nodes in the tree satisfy the heap property, i.e., the key stored in a node is smaller or equal to the keys stored in the node's children (in case of a min heap). The operations that are performed on this heap are createHeap(), insert(L), findMin(), deleteMin() and decreaseKey(L). The operation createHeap() creates the heap, insert(L) inserts a label L into the heap, findMin() returns the minimum label from the heap, deleteMin() returns and removes the minimum label from the heap and decreaseKey(L) decreases the values of θ_i and $\hat{c}_i^2$ of a label L in the heap.

Input : $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, $s \in \mathcal{N}$, $c : \mathcal{A} \to \mathbb{R}^2$.
Output : Paths associated to extreme supported non-dominated points in the objective space of the $s - i$ BSP problem for all nodes $i \in \mathcal{N} \setminus \{s\}$.

Calculate $\mathcal{T}$ being optimal for lexmin$\{c(p) : p \in \mathcal{P}\}$. Set $\mathcal{N}' = \mathcal{N} \setminus \{s\}$.
Store d and pred for tree $\mathcal{T}$. Set $\mathcal{L}_i^1 = (\text{pred}_i, 1, d_i^1, d_i^2)$ and $N_i = 1$ for all $i \in \mathcal{N}'$.
createHeap(). Set $(\text{lastRatio}_i, \theta_i, \text{CPred}_i) = (0, \infty, 0)$ for all $i \in \mathcal{N}'$.
Calculate lex $\min_{j \in \delta^-(i)} \{(-\bar{c}_{ij}^1/\bar{c}_{ij}^2, \bar{c}_{ij}^2, j) : \bar{c}_{ij}^2 < 0\}$ and update θ_i,$\hat{c}_i^2$ and CPred_i for all $i \in \mathcal{N}'$.
for $i \in \mathcal{N}'$ **where** $(\text{CPred}_i \neq 0)$ **do** insert$(\theta_i, \hat{c}_i^2, i)$. Set $\hat{c}_i^1 = \bar{c}_{\text{CPred}_i, i}^1$.
while $(H \neq \varnothing)$ **do**
    $(\theta_i, \hat{c}_i^2, i) =$ findMin(). deleteMin().
    if $(\theta_i > \text{lastRatio}_i)$ **then**
        $N_i = N_i + 1$. $\mathcal{L}_i^{N_i} = (\text{CPred}_i, N_{\text{CPred}_i}, \mathcal{L}_i^{N_i - 1}(d^1) + \hat{c}_i^1, \mathcal{L}_i^{N_i - 1}(d^2) + \hat{c}_i^2)$. $\text{lastRatio}_i = \theta_i$.
    else
        $\mathcal{L}_i^{N_i} = (\text{CPred}_i, N_{\text{CPred}_i}, \mathcal{L}_i^{N_i}(d^1), \mathcal{L}_i^{N_i}(d^2))$. $\mathcal{L}_i^{N_i}(d^1) = \mathcal{L}_i^{N_i}(d^1) + \hat{c}_i^1$. $\mathcal{L}_i^{N_i}(d^2) = \mathcal{L}_i^{N_i}(d^2) + \hat{c}_i^2$.
    end
    Set $(\theta_i, \text{CPred}_i) = (\infty, 0)$. Calculate lex $\min_{j \in \delta^-(i)} \{(-\bar{c}_{ij}^1/\bar{c}_{ij}^2, \bar{c}_{ij}^2, j) : \bar{c}_{ij}^2 < 0\}$.
    Update θ_i,$\hat{c}_i^2$ and CPred_i.
    if $(\text{CPred}_i \neq 0)$ **then** insert$(\theta_i, \hat{c}_i^2, i)$. Set $\hat{c}_i^1 = \bar{c}_{\text{CPred}_i, i}^1$.
    for $j \in \delta^+(i)$ where $(\bar{c}_{ij}^2 < 0)$ **do**
        if $((-\bar{c}_{ij}^1/\bar{c}_{ij}^2 < \theta_j)$ **or** $((-\bar{c}_{ij}^1/\bar{c}_{ij}^2 = \theta_j)$ **and** $(\bar{c}_{ij}^2 < \hat{c}_j^2))$ **then**
            if $(\text{CPred}_j = 0)$ **then** insert$(-\bar{c}_{ij}^1/\bar{c}_{ij}^2, \bar{c}_{ij}^2, j)$. **else** decreaseKey$(-\bar{c}_{ij}^1/\bar{c}_{ij}^2, \bar{c}_{ij}^2, j)$.
            Update θ_j, $\hat{c}_j$, and CPred_j.
        end
    end
end

Figure 2: Ratio-labeling algorithm (RLBSP).

Also, the labels $\mathcal{L}_i = (\mathcal{L}_i^1, \ldots, \mathcal{L}_i^{N_i})$ are stored for all nodes $i \in \mathcal{N}$, where N_i denotes the number of extreme supported solutions to the $s - i$ BSP problem. A label $\mathcal{L}_i^k$ is given by the tuple (j,r,d_i^1,d_i^2), where $j \in \mathcal{N}$ represents the predecessor node in solution $k \in \{1, \ldots, N_i\}$, the number $r \in \{1, \ldots, N_j\}$ is the index of the extreme supported solution associated to this node j needed for restoring the path, and d_i^1 and d_i^2 are the cost of this $s - i$ path in both criteria. Since the values of N_i are not known a priori, we must store counters in order to dynamically generate all the labels for all extreme supported solutions. Lastly, we also keep track of lastRatio_i for all nodes $i \in \mathcal{N}$, which store the slope corresponding to the last extreme supported solution for this node.

Since we do not explicitly store the extreme supported solutions to the one-to-all BSP problem, we need some kind of helper method for restoring their path sequences. This helper method is given by the method getPath(i, k, $\mathcal{L}_i$). It takes as an input a node $i \in \mathcal{N}$, a number $k \in \{1, \ldots, N_i\}$ being the k^{th} extreme supported solution for this node and the set of labels $\mathcal{L}_i = (\mathcal{L}_i^1, \ldots, \mathcal{L}_i^{N_i})$ for this node. The method then makes recursive calls until the complete path has been restored, by returning the node s if the input $i = s$, and getPath($\mathcal{L}_i^k(j)$,$\mathcal{L}_i^k(r)$,$\mathcal{L}_j$) + i otherwise. Since a path associated to an extreme supported non-dominated point is by definition simple, the program will make at most $\mathcal{O}(n)$ recursive calls to construct its path sequence.

5 Computational Results RLBSP

5.1 Dataset

We use some of the large road networks provided in the 9$^{\text{th}}$ DIMACS Implementation Challenge: Shortest Paths. These road networks represent different road networks from the USA. We make use of the instances NY (New York), BAY (San Francisco Bay Area), COL (Colorado), FLA (Florida) and NE (Northeast USA). The instances contain up to around 1.5 million nodes and 4 million arcs each and are rather sparse. Each instance has around two or three outgoing arcs per node on average.

5.2 Results

In order to test the performance of the RLBSP algorithm we execute it on the road network instances described above. We generate 100 random origin nodes s for each instance, for which the computational results are depicted in Table 1. The results are separated based on the total number of extreme supported solutions N. We include the s values for the minimum and maximum number of extreme supported solutions for each instance. The average number of extreme supported solutions for each instance is rounded.

Table 1: Computational results RLBSP for the road network instances. An $*$ indicates that during one of the 100 iterations, the run was interrupted due to an out of memory exception.

	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{A}\rvert$		s	CPU (s)	N
			Avg		3.05	2,917,827
NY	264,346	733,846	Min	47,166	1.98	2,082,588
			Max	176,598	6.09	4,090,583
			Avg		3.08	3,127,221
BAY	321,270	800,172	Min	320,972	1.94	2,144,949
			Max	107,059	6.15	4,575,641
			Avg		6.71	5,464,557
COL	435,666	1057,066	Min	206,043	3.75	3,897,902
			Max	241,410	13.44	8,731,452
			Avg		22.79	14,953,754
FLA	1,070,376	2,712,798	Min	657,833	12.60	9,994,311
			Max	245,832	39.01	28,141,173
			Avg		117.09*	32,377,245*
NE	1,524,453	3,897,636	Min	686,888*	79.68*	28,756,310*
			Max	1,186,909*	187.99*	35,284,750*

The total number of extreme supported solutions we find after executing the RLBSP are within the same range as those found in Sedeño–Noda and Raith (2015). To illustrate this, consider the instance NY (New York). Here we find an average, minimum and maximum number of extreme supported solutions of 2.92 million, 2.08 million and 4.09 million, while they found values of 3.00 million, 2.14 million and 4.03 million. The running times are also similar. While we obtained average, minimum and maximum running times of 3.05, 1.98 and 6.09 seconds for this instance respectively, they obtained running times of 2.30, 1.50 and 3.23 seconds for this instance respectively.

6 Problem Description Acyclic BSP Problem

The acyclic bi-objective shortest path problem (ABSP) concerns a special case of the regular BSP problem, where the underlying graph is assumed to be acyclic. Given are a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N} = \{1, \ldots, n\}$ is a set of nodes and $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ is a set of arcs. Also given are an origin node $s \in \mathcal{N}$, a destination node $t \in \mathcal{N}$ and a function $c : \mathcal{A} \to \mathbb{R}^2$ assigning two types of cost to the arcs in $\mathcal{G}$. We keep the rest of the notation identical to that of the regular BSP problem.

We denote $n = |\mathcal{N}|$ for the number of nodes and $m = |\mathcal{A}|$ for the number of arcs. The first assumption we make on $\mathcal{G}$ is similar to the one from the cyclic case, which is that for all nodes $i \in \mathcal{N} \setminus \{s\}$, $\mathcal{P}_{si} \neq \varnothing$, that is, there is at least one directed path from the origin node to all other nodes in the graph. The second assumption, however, does differ, since now we are considering the ABSP problem. This one simply says that the graph $\mathcal{G}$ does not contain any directed cycles.

The theory of acyclic graphs is well understood. It is well known that graphs that are acyclic can be transformed in order to exploit this property. A transformation that allows us to do so is called the topological ordering of a graph. It is defined as follows.

Definition 3. Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be an acyclic directed graph with node set $\mathcal{N} = \{1, \ldots, n\}$. The *topological ordering* of $\mathcal{G}$ is a bijective mapping of the nodes $\pi = (\pi_1, \ldots, \pi_n)$, such that for all arcs $(\pi_\ell, \pi_k) \in \mathcal{A}$, $\ell < k$.

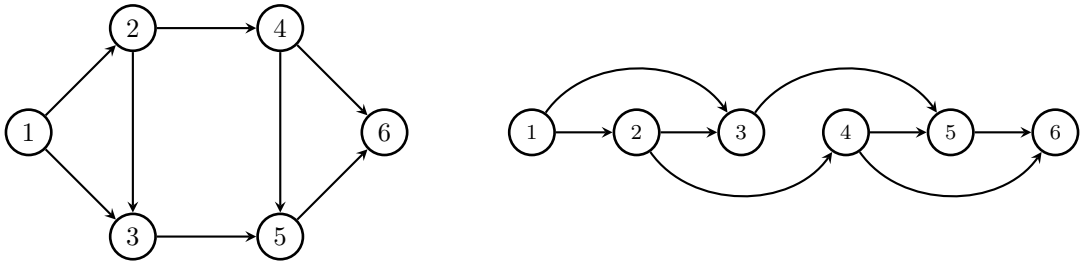

Figure 3: A directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, with node set $\mathcal{N} = \{1, \ldots, 6\}$ and arc set $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$. A topological ordering of the graph $\mathcal{G}$ (right) is given by $\pi = (1, 2, 3, 4, 5, 6)$.

We find that any topological ordering automatically maps the first node to the origin node $\pi_1 = s$, due to the first assumption on $\mathcal{G}$. Note that the topological ordering of a graph is not necessarily unique. Take the example above. Here, $\pi = (1, 2, 4, 3, 5, 6)$ would also satisfy the definition of a topological ordering.

Finding a topological ordering of a graph $\mathcal{G}$, also known as topological sorting, can be done using algorithms like those described in Kahn (1962) and Tarjan (1976). The time complexity of topological sorting algorithms is well known.

Lemma 2. The topological ordering of a graph can be constructed in $\mathcal{O}(n + m)$ steps.

Proof. Omitted. See for example Cormen (2009). ■

In this thesis we consider the one-to-all ABSP problem, which comes down to finding all extreme supported solutions to the $s-i$ ABSP problem for all nodes $i \in \mathcal{N} \setminus \{s\}$. While we could also consider the one-to-one ABSP problem, it is known that a dichotomic approach can be used to solve this problem in $\mathcal{O}(N_t \mathcal{ASP})$ steps (e.g. Steiner and R. (2003)), where N_t denotes the number of extreme supported solutions to the $s-t$ ABSP problem and $\mathcal{ASP}$ is the time complexity of solving the $s-t$ ASP problem. The topological ordering of a graph can be used to solve this last problem in $\mathcal{O}(m)$ steps (Dijkstra's algorithm but we do not need the heap data structure anymore), resulting in an $\mathcal{O}(N_t m)$ algorithm for solving the one-to-one ABSP problem.

We could, in theory, solve the one-to-all ABSP problem by solving separate one-to-one ABSP problems, yielding an algorithm with a time complexity of $\mathcal{O}(Nm)$ steps, where $N = \sum_{i \in \mathcal{N}} N_i$ is the total number of extreme supported solutions. However, there are two main reasons why we must not do so. First of all, we lose a lot of information by solving all the one-to-one problems separately. All the extreme supported solutions of one node can be used to find the extreme supported solutions of another node. Secondly, from a storage point of view this would be extremely inefficient, since now we would have to store all the path sequences explicitly, instead of storing labels allowing us to restore them.

As far as we know, a dichotomic approach for solving the one-to-all ABSP problem would also not be possible. We extend on an argument they give in Sedeño–Noda and Raith (2015), which also holds for the regular BSP problem. When a dichotomic approach is executed on an $s-t$ ABSP problem, the path algorithms applied for solving this problem might not evaluate all the $s-j$ paths (due to the elimination of paths in the labeling procedure). So after the execution of the first dichotomic procedure, some of the extreme supported paths to the $s-j$ ABSP problem might still need to be computed. These remaining paths can be found, however, by applying another dichotomic approach, but now to the $s-j$ ABSP problem. But this means that in the most extreme scenario, we have to solve almost all the $s-i$ ABSP problems separately, and we are back to the argument described above.

7 Labeling Algorithm ABSP Problem

In this section we propose a labeling algorithm to obtain all the extreme supported solutions to the one-to-all ABSP problem.

7.1 Algorithm and Correctness

Before getting into the actual algorithm, we extend some of the definitions regarding the solutions to the BSP problem. We denote $\mathcal{P}_{ij}$ for the set of all $i-j$ paths, and $\mathcal{C}_{ij} = \{c(p) : p \in \mathcal{P}_{ij}\}$ for its set of images, similar as in Section 3.3. Next we define $\mathcal{E}(\mathcal{C})$ as the set of points associated to the extreme supported paths from the set $\mathcal{P}$, that is, the set $\mathcal{E}(\mathcal{C})$ consists of all points $c \in \mathcal{C}$ that are a solution to the weighted sum problem

$$w(\lambda) = \min_{c \in \mathcal{C}} \left\{ \lambda c^1 + (1-\lambda) c^2 \right\}$$

for some $\lambda \in [0,1]$, and are an extreme point of the set $\text{conv}(\mathcal{C})$. Simply put, $\mathcal{E}(\mathcal{C})$ is the set of extreme points on the lower left boundary of the set $\text{conv}(\mathcal{C})$. Note how by definition, $\mathcal{E}(\mathcal{C}) \subseteq \mathcal{C}$ and $N_i = |\,\mathcal{E}(\mathcal{C}_{si})\,|$. We sometimes write $w(\lambda, \mathcal{C})$ to indicate which set $\mathcal{C}$ is being used in the parametric program above. The following lemma plays an important role in the proof of the correctness of the proposed algorithm.

Lemma 3. Let $\mathcal{C} \subseteq \mathbb{R}^2$ and $\mathcal{E}(\mathcal{C}) \subseteq \mathcal{C}_1 \subseteq \mathcal{C}$. Then

$$\mathcal{E}\left(\mathcal{C}_1\right) \subseteq \mathcal{E}(\mathcal{C}).$$

Proof. Since $\mathcal{E}(\mathcal{C}) \subseteq \mathcal{C}_1$, $\mathcal{C}_1$ can be partitioned into $\mathcal{C}_1 = \mathcal{E}(\mathcal{C}) \cup \mathcal{C}_2$, with $\mathcal{C}_2 = \mathcal{C}_1 \setminus \mathcal{E}(\mathcal{C})$. If $\mathcal{C}_2 = \varnothing$ the lemma follows. For the following, assume that $\mathcal{C}_2 \neq \varnothing$. Take any $c \in \mathcal{E}(\mathcal{C}_1)$. If $c \in \mathcal{E}(\mathcal{C})$ we are done. If $c \in \mathcal{C}_2$, then $c \notin \mathcal{E}(\mathcal{C})$, due to the partition of $\mathcal{C}_1$. We evaluate two cases, $|\,\mathcal{E}(\mathcal{C})\,| = 1$ and $|\,\mathcal{E}(\mathcal{C})\,| \geqslant 2$.

If $|\,\mathcal{E}(\mathcal{C})\,| = 1$, then a single point $e \in \mathcal{E}(\mathcal{C})$ is supported for all $\lambda \in [0,1]$, and all the points in $\mathcal{C} \setminus \{e\}$ are not supported at all. But this means that c is not supported for $\mathcal{C}_1$ (because $e \in \mathcal{C}_1$), so this results in the contradiction $c \notin \mathcal{E}(\mathcal{C}_1)$.

Take now the second case, $|\,\mathcal{E}(\mathcal{C})\,| \geqslant 2$. Let's number the points $e_1, \ldots, e_N \in \mathcal{E}(\mathcal{C})$, $2 \leqslant N$, first on non-decreasing value of the first criterion and then on non-increasing value of the second criterion. We distinguish between two scenarios, which are $c \in \text{conv}(\mathcal{E}(\mathcal{C}))$ and $c \notin \text{conv}(\mathcal{E}(\mathcal{C}))$. If $c \in \text{conv}(\mathcal{E}(\mathcal{C}))$ (e.g. c_2 in Figure 4), then the point c can not be an extreme point of this set. Since $\mathcal{E}(\mathcal{C}) \subseteq \mathcal{C}_1$, this means that c can also not be an extreme point of the set $\text{conv}(\mathcal{C}_1)$, and we arrive at the contradiction $c \notin \mathcal{E}(\mathcal{C}_1)$. Now consider the second scenario, $c \notin \text{conv}(\mathcal{E}(\mathcal{C}))$ (e.g. c_3 in Figure 4). This means that c lies above right of the line ℓ between the points e_1 and e_N. But this means that c is not supported for the set $\mathcal{C}_3 = \{e_1, e_N, c\}$ for any $\lambda \in [0,1]$. If c is not supported for the set $\mathcal{C}_3 \subseteq \mathcal{C}_1$, then it is clearly also not supported for the set $\mathcal{C}_1$, because $w(\lambda, \mathcal{C}_1) \leqslant w(\lambda, \mathcal{C}_3)$ for all $\lambda \in [0,1]$. We again arrive at the contradiction $c \notin \mathcal{E}(\mathcal{C}_1)$. For all points $c \in \mathcal{E}(\mathcal{C}_1)$ we have $c \in \mathcal{E}(\mathcal{C})$, thus $\mathcal{E}(\mathcal{C}_1) \subseteq \mathcal{E}(\mathcal{C})$. ■

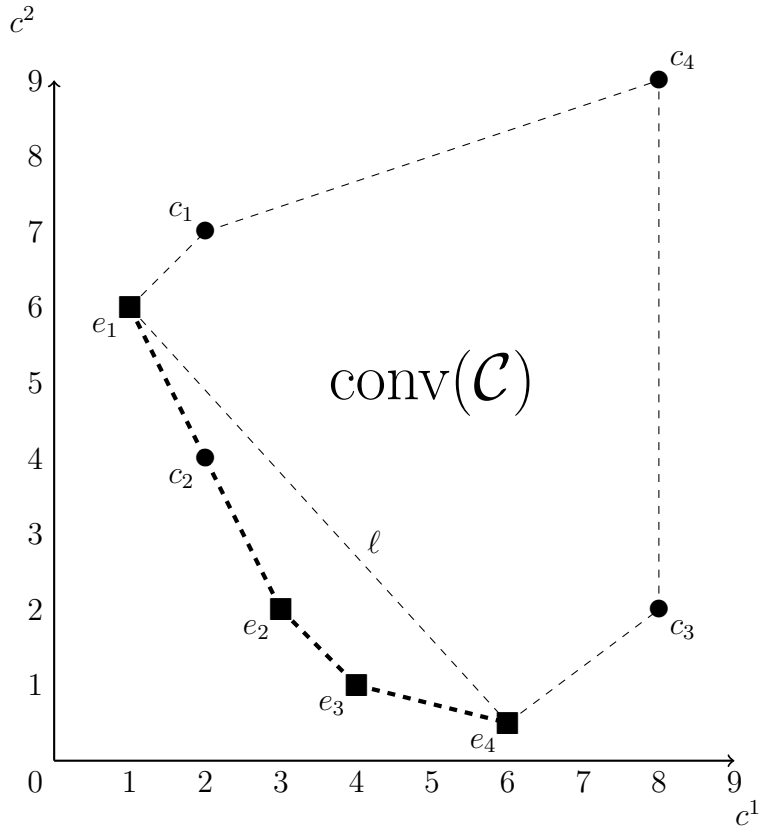

Figure 4: Points in the objective space of a BSP problem. A • indicates a point in the objective space, while a ■ indicates a point in the objective space that is extreme supported for the set conv($\mathcal{C}$). A thick dotted line between two points means that these points are both supported for some $\lambda \in [0, 1]$. Also displayed is the line ℓ between the two points e_1 and e_4 from the set $\mathcal{E}(\mathcal{C})$. An example of a set $\mathcal{C}_1$ would be $\mathcal{C}_1 = \mathcal{C} \setminus \{c_4\}$, for which the partition would be given by $\mathcal{C}_1 = \mathcal{E}(\mathcal{C}) \cup \mathcal{C}_2$, with $\mathcal{E}(\mathcal{C}) = \{e_1, \ldots, e_4\}$ and $\mathcal{C}_2 = \{c_1, c_2, c_3\}$.

Before giving the labeling algorithm, we would like to introduce one more result regarding the supported solutions to the BSP problem.

Lemma 4. All supported paths $p_{st} \in \mathcal{P}$ satisfy *Bellman's Principle of Optimality*. That is, if p_{st} is supported for all $\lambda \in \Lambda \subseteq [0, 1]$, than any sub-path p_{ij} of this path ($p_{st} = p_{si} \diamond p_{ij} \diamond p_{jt}$) is also supported for all $\lambda \in \Lambda$.

Proof. Similar to the proof of Theorem 3.2 from Sniedovich (1986). Assume that a sub-path $p_{ij} \in \mathcal{P}_{ij}$ is not supported for some $\overline{\lambda} \in \Lambda$. Then there exists another path $\overline{p}_{ij}$ which is, so $w(\overline{\lambda}, \overline{p}_{ij}) < w(\overline{\lambda}, p_{ij})$. If we let $\overline{p}_{st} = p_{si} \diamond \overline{p}_{ij} \diamond p_{jt}$, then for this $\overline{\lambda}$,

$$\begin{aligned} w(\overline{\lambda}, \overline{p}_{st}) &= w(\overline{\lambda}, p_{si}) + w(\overline{\lambda}, \overline{p}_{ij}) + w(\overline{\lambda}, p_{jt}) \\ &< w(\overline{\lambda}, p_{si}) + w(\overline{\lambda}, p_{ij}) + w(\overline{\lambda}, p_{jt}) \\ &= w(\overline{\lambda}, p_{st}), \end{aligned}$$

contradicting the fact that p_{st} is supported for $\overline{\lambda} \in \Lambda$. ■

We are now ready to present the algorithm. The pseudocode of the algorithm is given as follows.

Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be a topological ordering of $\mathcal{G}$.
Set $K_s = \{(0,0)\}$ and $K_i = \varnothing$ for all nodes $i \in \mathcal{N} \setminus \{s\}$.
for $i = 2$ to n **do**
  for $(\pi_\ell, \pi_i) \in \mathcal{A}$ **do**

$$K_{\pi_i} = \mathcal{E}\left(K_{\pi_i} \bigcup_{c \in K_{\pi_\ell}} (c + c_{\pi_\ell \pi_i})\right)$$

Return $(K_1, \ldots, K_n)$.

Figure 5: Labeling algorithm for the ABSP problem.

Here, π denotes a topological ordering of the graph $\mathcal{G}$ where $\pi_1 = s$ and K_i is a set of costs associated to the solutions to the $s - i$ ABSP problem, for all nodes $i \in \mathcal{N}$. We claim that after executing the algorithm, all the sets K_i are exactly the sets containing all points associated to extreme supported solutions to the $s - i$ ABSP problems. This claim is captured in the following theorem.

Theorem 1. After executing the algorithm it holds, for all $1 \leqslant i \leqslant n$, that

$$\mathcal{E}(\mathcal{C}_{s\pi_i}) = K_{\pi_i}.$$

Proof. We will prove the theorem by induction on $1 \leqslant i \leqslant n$. For $i = 1$ the proof is trivial, since

$$\mathcal{E}(\mathcal{C}_{s\pi_1}) = \mathcal{E}(\mathcal{C}_{ss}) = \{(0,0)\} = K_s = K_{\pi_1}.$$

Now let $2 \leqslant j \leqslant n$, and suppose that $\mathcal{E}(\mathcal{C}_{s\pi_i}) = K_{\pi_i}$ for all $1 \leqslant i \leqslant j-1$. We will now show that $\mathcal{E}(\mathcal{C}_{s\pi_j}) = K_{\pi_j}$, by showing that both $\mathcal{E}(\mathcal{C}_{s\pi_j}) \subseteq K_{\pi_j}$ and $\mathcal{E}(\mathcal{C}_{s\pi_j}) \supseteq K_{\pi_j}$.

$\subseteq$. Take any $\overline{c} \in \mathcal{E}(\mathcal{C}_{s\pi_j})$ associated to a path $p_{s\pi_j} \in \mathcal{P}_{s\pi_j}$. Than this $\overline{c}$ can be split up in

$$\overline{c} = c + c_{\pi_\ell \pi_j},$$

where $c_{\pi_\ell \pi_j}$ is the cost of some arc $(\pi_\ell, \pi_j) \in \mathcal{A}$, and c is a supported point in $\mathcal{C}_{s\pi_\ell}$ due to Lemma 4. Now suppose c is not extreme, then this would imply that $\overline{c}$ is also not extreme, so c must be an extreme supported point, hence $c \in \mathcal{E}(\mathcal{C}_{s\pi_\ell})$. By the induction hypothesis it then holds that $c \in K_{\pi_\ell}$. Since the arc $(\pi_\ell, \pi_j) \in \mathcal{A}$ and the set K_{π_ℓ} are both visited in the algorithm, the point $\overline{c}$ must at some point be included in the set K_{π_j}, and since this point has the property of being both extreme and supported for the set $\text{conv}(\mathcal{C}_{s\pi_j})$, it cannot be eliminated. Hence $\overline{c} \in K_{\pi_j}$ and $\mathcal{E}(\mathcal{C}_{s\pi_j}) \subseteq K_{\pi_j}$.

$\supseteq$. Let's write $\overline{K}_{\pi_j}$ for the set that is given by K_{π_j} before the execution of the last $\mathcal{E}(\cdot)$ operation for this node π_j. Since $\mathcal{E}(\mathcal{C}_{s\pi_j}) \subseteq K_{\pi_j}$, all points of $\mathcal{E}(\mathcal{C}_{s\pi_j})$ must be included in $\overline{K}_{\pi_j}$ before executing this last $\mathcal{E}(\cdot)$ operation (taking $\mathcal{E}(\mathcal{C}_1)$ of any set $\mathcal{C}_1 \subseteq \mathcal{C}$ does not add any points to the set). We also have $\overline{K}_{\pi_j} \subseteq \mathcal{C}_{s\pi_j}$, because $\mathcal{C}_{s\pi_j}$ is the set of points associated to all the $s - \pi_j$ paths. From Lemma 3 now follows that after executing the last $\mathcal{E}(\cdot)$ iteration, we get $K_{\pi_j} = \mathcal{E}\left(\overline{K}_{\pi_j}\right) \subseteq \mathcal{E}(\mathcal{C}_{s\pi_j})$. ■

7.2 Running Time

In order to implement the algorithm we need some kind of merging procedure, which we will call EXT(A, B), for merging two sets of extreme supported points in $\mathbb{R}^2$. We elaborate on an algorithm used in Henig (1986). The elimination procedure takes as an input two ordered sets $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ containing extreme supported points in $\mathbb{R}^2$, and returns a single ordered set $C = (c_1, \ldots, c_k)$, $1 \leqslant k \leqslant n + m$, containing all extreme supported points of the set $\text{conv}(A \cup B)$. The ordering is first done in non-decreasing order on the first coordinate and then on non-increasing order on the second coordinate.

In order to determine all the extreme supported points, the algorithm needs to compute the slope Δ between two points $x_1, x_2 \in \mathbb{R}^2$, defined as $\Delta(x_1, x_2) = (x_2^2 - x_1^2)/(x_2^1 - x_1^1)$ if $x_1^1 \neq x_2^1$, and $\pm\infty$ or 0 if $x_1^1 = x_2^1$ depending on whether x_1 lies above, below, or is equal to x_2. The pseudocode of the algorithm is given by:

Set curr equal to most left (in case of tie most above) point of (a_1, b_1). Set $C = \{\text{curr}\}$.
while $(A, B) \neq (\varnothing, \varnothing)$ **do**
Find next $a_i \in A$ for which $\text{curr}^1 < a_i^1$ and $\Delta_1 = \Delta(\text{curr}, a_i)$ is minimal. If no such a_i exists set $A = \varnothing$ and $\Delta_1 = \infty$.
Find next $b_j \in B$ for which $\text{curr}^1 < b_j^1$ and $\Delta_2 = \Delta(\text{curr}, b_j)$ is minimal. If no such b_j exists set $B = \varnothing$ and $\Delta_2 = \infty$.
If $(\Delta_1, \Delta_2) \neq (\infty, \infty)$ then choose from (a_i, b_j) according to the minimal value of Δ_i, $i = 1, 2$. If $\Delta_1 = \Delta_2$ choose the furthest of the two points. Update curr and set $C = C \cup \{\text{curr}\}$. If a_i is chosen set $A = A \setminus \{a_i\}$. Else set $B = B \setminus \{b_j\}$.

Figure 6: EXT.

Here, curr represents the last point added to the set C, Δ_1 denotes the minimal slope between the point curr and some point from $(a_i, \ldots, a_n)$, $a_i^1 > \text{curr}^1$, and Δ_2 denotes the minimal slope between the point curr and some point from $(b_j, \ldots, b_m)$, $b_j^1 > \text{curr}^1$. Since A and B both only contain extreme supported points, the values of Δ_1 and Δ_2 will first (possibly) decrease, and then steadily increase. As soon as they increase the minimal slope between curr and the point from that set is found. If $\Delta_1 = \Delta_2$, the furthest of the two points is chosen to prevent the inclusion of supported non-extreme points. While a double for loop would have been sufficient in determining the set C, this somewhat more elaborate algorithm has a nice computational complexity, as stated by the following lemma.

Lemma 5. EXT(A, B) runs in $\mathcal{O}(n + m)$ steps, where $n = |\, A \,|$ and $m = |\, B \,|$.

Proof. All lines in the main while loop run in constant time and consist of either visiting or removing an element from one of the sets. Since an execution of the main while loop therefore permanently visits at least one of the elements from one of the sets, the algorithm runs in linear time in the sizes of the sets A and B. ■

The computational complexity of the proposed algorithm for solving the ABSP problem is given by the following theorem.

Theorem 2. The ABSP algorithm runs in $\mathcal{O}(\max_{i\in\mathcal{N}}\{N_i\}m)$ steps, where N_i is the number of extreme solutions to the $s-i$ ABSP problem, and $m = |\,\mathcal{A}\,|$ is the number of arcs in the graph under consideration.

Proof. Finding a topological ordering of the graph $\mathcal{G}$ does not contribute much to the asymptotic complexity, because this can be done in $\mathcal{O}(n+m)$ steps. Finding the extreme supported points, however, does, since it takes at most $\mathcal{O}(m\mathcal{EXT})$ steps to execute all the merging procedures, with $\mathcal{EXT}$ the maximum running time of the algorithm $\textsc{ext}(A,B)$. The input of the merging procedure $\textsc{ext}(A,B)$ at any point during execution is given by the sets K_{π_ℓ} and K_{π_i}, associated to the evaluation of arc $(\pi_\ell,\pi_i)\in\mathcal{A}$. Now, since the merging procedure runs in linear time, we find that the maximum number of steps $\mathcal{EXT}$ needed for the algorithm to execute is bounded by $2\max_{i\in\mathcal{N}}\{N_i\}$, for any arc $(\pi_\ell,\pi_i)\in\mathcal{A}$, so that the full algorithm is executable in $\mathcal{O}(\max_{i\in\mathcal{N}}\{N_i\}m)$ steps. ■

We will state that the time complexity is $\mathcal{O}(Nm)$ though, with $N=\sum_{i\in\mathcal{N}}N_i$, for simplicity. The exact running time is something we will use later, when dealing with the case where at least one of the criteria is of the bottleneck type in Section 9.

7.3 Example Execution

In this section we present an example execution of the proposed algorithm for solving the ABSP. We consider an instance of the ABSP problem given by the graph in Figure 7 below, with an origin node $s=1$.

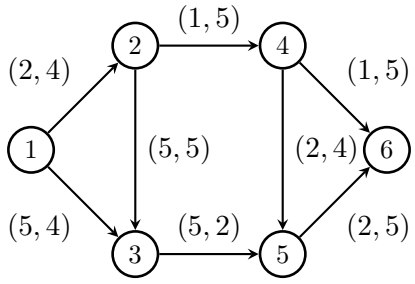

Figure 7: A directed graph $\mathcal{G}=(\mathcal{N},\mathcal{A})$, with node set $\mathcal{N}=\{1,\ldots,6\}$, arc set $\mathcal{A}\subseteq\mathcal{N}\times\mathcal{N}$, and (c^1_{ij},c^2_{ij}) the costs associated to traversing arc $(i,j)\in\mathcal{A}$.

The algorithm starts with determining a topological ordering of the graph $\mathcal{G}$, which is given by $\pi=$ (1,2,4,3,5,6). It then initializes $K_1=\{(0,0)\}$ and $K_i=\varnothing$ for all nodes $i\in\mathcal{N}\setminus\{1\}$. Since $\pi_2=2$, the first arc visited is (1,2), giving $K_2=\mathcal{E}\left(\{(2,4)+(0,0)\}\right)=\{(2,4)\}$. The next node in the topological ordering is given by $\pi_3=4$, visiting arc (2,4) and giving $K_4=\mathcal{E}\left(\{(1,5)+(2,4)\}\right)=\{(3,9)\}$. After that, node $\pi_4=3$ is visited. This node has two incoming arcs (1,3) and (2,3), resulting in $K_3=\mathcal{E}\left(\{(5,4)+(0,0),(5,5)+(2,4)\}\right)=\{(5,4)\}$, because (5,4) < (7,9). Node $\pi_5=5$ is considered next, with incoming arcs (3,5) and (4,5), hence $K_5=\mathcal{E}\left(\{(5,2)+(5,4),(2,4)+(3,9)\}\right)=\{(10,6),(5,13)\}$. Finally, the node $\pi_6=6$ is evaluated with incoming arcs (4,6) and (5,6), for which $K_6=\mathcal{E}\left(\{(1,5)+(3,9),(2,5)+(10,6),(2,5)+(5,13)\}\right)=\{(4,14),(12,11)\}$, because (4,14) < (7,18). The algorithm then terminates by returning $(K_1,\ldots,K_6)$.

7.4 Optimality Intervals

While our proposed algorithm currently only computes the extreme supported points in the objective space and thus does not provide any information regarding the extreme supported solutions to the ABSP problem, the algorithm can easily be adapted to do so. Let, without loss of generality, the extreme supported solutions $p_1, \ldots, p_N \in \mathcal{P}$ with costs $c_1, \ldots, c_N \in \mathcal{C}$ of an $s - i$ ABSP problem be numbered first on non-decreasing value of the first criterion and then on non-increasing value of the second criterion. Right now, the only information that is available for a path $p \in \mathcal{P}$ is its associated cost $c(p) \in \mathcal{C}$. Yet, much more information of these paths can be captured, like the actual path sequence $i_1 \to i_2 \to \ldots \to i_\ell$ or its optimality interval $\Lambda \subseteq [0, 1]$ for the parametric program $w(\lambda)$ described in Section 3.3. This information can then be captured by using labels of the form

$$\mathcal{L}_i^k = (j, r, c, \Lambda),$$

similar as we would for the RLBSP algorithm. When executing the algorithm, the predecessor nodes j, the indices r and the costs c of the extreme supported solutions are then readily available, but Λ is not. However, Λ can easily be calculated using the following lemma.

Lemma 6. Let $\mathcal{C} = (c_1, \ldots, c_k)$ be a set of extreme supported points in $\mathbb{R}^2$. For all $1 \leqslant i \leqslant k-1$, the points c_i and c_{i+1} are both supported for $\lambda_i = \Delta_i/(\Delta_i - 1)$, where $\Delta_i = \Delta(c_i, c_{i+1})$ is the slope between these two points $c_i, c_{i+1} \in \mathcal{C}$.

Proof. Since $\Delta_i = \Delta(c_i, c_{i+1})$ denotes the slope between the points c_i and c_{i+1}, and $\mathcal{C}$ is a set of extreme supported points, c_i and c_{i+1} both minimize the expression $-\Delta_i c^1 + c^2$. This is because the line between the two points can be described as $c^2 = K + \Delta_i c^1$, with K some non-negative constant. The value of Δ_i is either negative or zero, so $1 - \Delta_i > 0$, hence c_i and c_{i+1} are both also minimizers of (similar trick used in the proof of Lemma 5.1 from Hamacher and Ruhe (1994))

$$\frac{1}{1-\Delta_i}\left(-\Delta_i c^1 + c^2\right) = \underbrace{\left(\frac{\Delta_i}{\Delta_i - 1}\right)}_{\lambda_i} c^1 + \underbrace{\left(1 - \frac{\Delta_i}{\Delta_i - 1}\right)}_{1-\lambda_i} c^2,$$

and are therefore a solution to the parametric program $w(\lambda)$ from Section 7.1 for $\lambda_i = \Delta_i/(\Delta_i - 1)$. ∎

The optimality intervals $\Lambda = [\lambda_1, \lambda_2]$ for the extreme supported solutions can then be calculated as follows. For the k^{th} extreme supported solution, the value of λ_1 is given by $\Delta/(\Delta - 1)$, with $\Delta = \Delta(c(p_{k-1}), c(p_k))$ the slope between the cost of the previous solution and the cost of the current solution, and the value of λ_2 is given by that same expression, but now with $\Delta = \Delta(c(p_k), c(p_{k+1}))$ the slope between the cost of the current solution and the cost of the next solution. Note that for the border cases p_1,$p_N \in \mathcal{P}$ we intuitively define $\Delta_0 = \Delta(c(p_0), c(p_1)) = -\infty$ and $\Delta_N = \Delta(c(p_N), c(p_{N+1})) = 0$, so the first solution p_1 and the last solution p_N are optimal for values of λ of 1 and 0 respectively. That a path is supported for all $\lambda \in \Lambda$ requires us to evaluate any $\lambda \in (\lambda_1, \lambda_2)$, but the fact that this path is supported for these λ follows from an argument similar to the one given above.

8 Computational Results ABSP Problem

8.1 Dataset

We use grid networks in order to test the implementation of the labeling algorithm used for solving the ABSP. The grid networks consist of L layers, with every layer containing L nodes each.

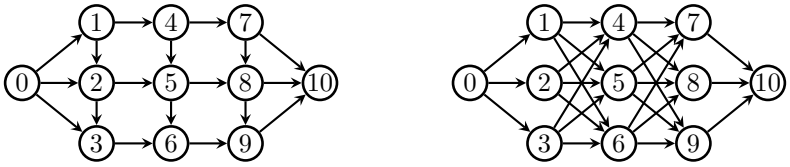

Figure 8: A sparse grid network (left) and a dense grid network (right) of size $L = 3$.

For every node in the sparse grid network, there is an outgoing arc to its immediate right and bottom neighbour, given these neighbours exist. An origin node is added to the front and outgoing arcs from this node are added to every node in the first layer, and a destination node is added to the back and outgoing arcs to this node are added for every node in the last layer. For every node in the dense grid network, there is an outgoing arc to all the nodes in the right neighbouring layer, given this layer exists. The origin node and the destination node are added in the same way as described for the sparse grid network. Hence, for a given grid size L, the number of nodes is equal to $L^2 + 2$ for both types of grid networks and the number of arcs is equal to to $2L^2$ and $L^2(L-1) + 2L \approx L^3$ for the sparse and dense grid networks respectively. We label the nodes as $r + (\ell - 1)L$ for row $1 \leqslant r \leqslant L$ and layer $1 \leqslant \ell \leqslant L$, and 0 and $L^2 + 1$ for the origin and destination node. The costs $c_{ij} = (c^1_{ij}, c^2_{ij})$ are drawn randomly from the set $\{1, \ldots, 100\}^2$, for all arcs (i, j). Finally, note that due to the structure of the grid networks, all instances satisfy the property of being acyclic.

8.2 Results

We tested the algorithm by executing it on the sparse and dense grid networks introduced in Section 8.1, for which the results are depicted in Table 2.

Table 2: Computational results of the proposed algorithm for solving the ABSP problem. Both the sparse and dence grid networks are considered.

Type	L	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{A}\rvert$	Min		Max		Avg	
				CPU (s)	N	CPU (s)	N	CPU (s)	N
Sparse	400	160,002	320,000	1.14	6,378,903	4.55	7,347,313	1.45	6,706,961
	600	360,002	720,000	5.59	21,354,559	14.89	22,940,312	7.43	22,027,827
	800	640,002	1,280,000	15.39	49,664,229	38.07	52,764,789	20.22	51,127,673
	950	902,502	1,805,000	33.68	82,506,930	78.33	87,988,381	52.83	85,008,735
	985	970,227	1,940,450	93.99	91,871,797	322.38	95,363,161	153.90	93,958,098
Dense	100	10,002	990,200	6.10	463,998	8.48	561,798	6.72	503,388
	150	22,502	3,352,800	28.29	1,321,438	39.52	1,623,482	31.13	1,444,929
	200	40,002	7,960,400	75.40	2,622,431	91.34	3,145,959	83.84	2,912,480
	250	62,502	15,563,000	166.24	4,547,961	202.44	5,459,411	186.81	5,036,390
	300	90,002	26,910,600	279.07	6,844,633	378.71	8,530,446	320.65	7,770,658

We ran the algorithm 100 times on both types of grid networks for various values of L, and stored the minimum, maximum and average run statistics. The sizes of the dense grid network instances we considered are much smaller than those of the sparse grid network instances, due to the rapidly increasing number of arcs in the network. We also performed a computational comparison between the RLBSP algorithm from Sedeño–Noda and Raith (2015) and the algorithm we proposed for solving the ABSP problem.

The results from the comparison of the sparse grid networks are shown in Figure 9. We see a superior performance by the algorithm we proposed. The difference in running time can be explained by the structure of the algorithms. The running time of the algorithm we proposed is insensitive to the number of nodes in the network due to the inclusion of a topological sorting phase, while the RLBSP is dependent on the number of nodes by a term $\mathcal{O}(Nn \log n)$ due to the use of a heap data structure. Hence, for rather sparse networks we find that many extra operations are needed to execute the RLBSP, while this is not the case for our proposed algorithm. For the dense grid networks we find a similar performance by both algorithms. For sizes $L = 75$, 100 and 125 (dense grid network instances with $L = 125$ already have 2 million arcs) we find average running times of 2.27 and 1.86, 6.72 and 5.09, and 17.13 and 10.22 seconds for the ABSP algorithm and the RLBSP respectively. This suggests that our algorithm is a little slower, but both algorithms can still be used to quickly solve dense grid network instances. An explanation for the similar performance can be found in the structure of the dense grid networks. For a given number of layers L, the number of arcs in the network is much bigger than the number of nodes in the network. Hence, for dense grid networks where the size L is rather large, the term $n \log n$ in the complexity of the RLBSP becomes negligible, so in this case the algorithm is roughly dependent on an Nm term (since in this case $N(m + n \log n) \approx Nm$, or at least by some factor). Since this dependence is similar for the algorithm we proposed, it explains the similar performance.

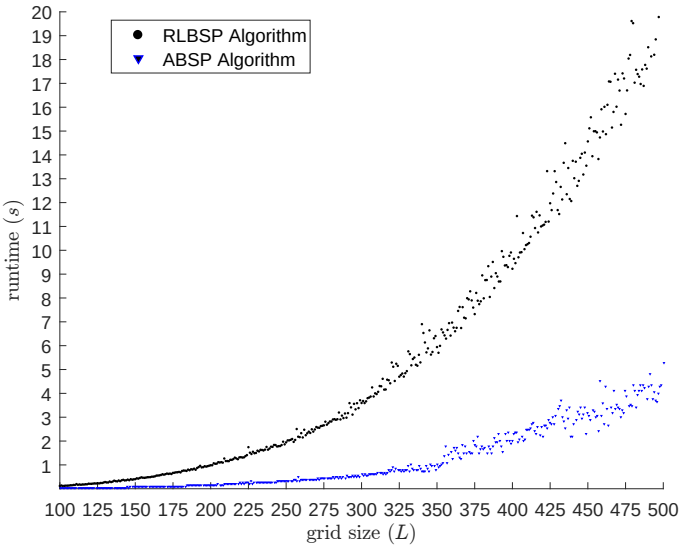

Figure 9: The ABSP algorithm runtimes and the RLBSP runtimes for different sizes of the sparse grid networks.

9 ABSP Problem Bottleneck

In this section we evaluate the ABSP problem, where either one or both of the objectives are of the bottleneck type (or min-max type for that matter). We are given, exactly like in the regular ABSP problem, a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, an origin node $s \in \mathcal{N}$, a destination node $t \in \mathcal{N}$ and a cost function $c : \mathcal{A} \to \mathbb{R}^2$ assigning two types of cost to the arcs in $\mathcal{G}$. We continue to use π to denote a topological ordering of the graph $\mathcal{G}$. We consider a difference that takes place in the way that the cost function is extended for a path $p \in \mathcal{P}$ for at least one of the criteria. Without loss of generality we assume that this difference in extension is only for the first criterion, but all arguments are still valid if the extension is for both criteria.

We will show that only minor changes to the algorithm are needed while still obtaining the extreme supported solutions for the one-to-all ABSP problem. In order to realize these changes, we consider the algorithm from Section 7.1, with the generalization of the main line given by

$$K_{\pi_i} = \mathcal{E}\left(K_{\pi_i} \bigcup K_{\pi_\ell \pi_i}\right),$$

where the set $K_{\pi_\ell \pi_i}$ is defined appropriately for the type of criteria used. This set $K_{\pi_\ell \pi_i}$ should be defined in such a manner that the cost of an $s - \pi_i$ path can be calculated using the cost of an $s - \pi_\ell$ path and the cost of the arc $(\pi_\ell, \pi_i) \in \mathcal{A}$ connecting these two sets. Simply put, the problem comes down to finding how we can determine the cost of a path q given the cost of a path p and the cost of an arc (k, ℓ), given $q = p \cup (k, \ell)$. In case two additive criterion functions are used we can find $c(q)$ simply by $c(q) = c(p) + c_{k\ell}$, since

$$c(q) = \sum_{(i,j) \in q} c_{ij} = \sum_{(i,j) \in p \cup (k,\ell)} c_{ij} = \sum_{(i,j) \in p} c_{ij} + c_{k\ell} = c(p) + c_{k\ell},$$

and this is exactly the relationship used in the calculation of $K_{\pi_\ell \pi_i}$ for the labeling algorithm we proposed in Section 7.1. In case a *bottleneck* criterion function is used instead of the regular additive criterion function for the first criterion, we have the criterion function given by $c^1(p) = \max\{c^1_{ij} : (i, j) \in p\}$, for any path $p \in \mathcal{P}$. We can then easily determine the cost $c(q)$ given the cost $c(p)$, with the help of the following lemma.

Lemma 7. Given two paths p and q satisfying $q = p \cup (k, \ell)$ and a maximum function d, then $d(q)$ can be determined using $d(q) = \max\{d(p), d_{k\ell}\}$.

Proof.

$$\begin{aligned} d(q) &= \max_{(i,j) \in q} \{d_{ij}\} = \max_{(i,j) \in p \cup (k,\ell)} \{d_{ij}\} \\ &= \max\left\{ \max_{(i,j) \in p} \{d_{ij}\}, d_{k\ell} \right\} = \max\{d(p), d_{k\ell}\}. \end{aligned}$$ ■

Hence, we can find all extreme supported points by defining the set $K_{\pi_\ell \pi_i}$ as

$$K_{\pi_\ell \pi_i} = \left\{ (\max\{c^1, c^1_{\pi_\ell \pi_i}\}, c^2 + c^2_{\pi_\ell \pi_i}) : c \in K_{\pi_\ell} \right\},$$

given the set K_{π_ℓ} and the cost of the arc $(\pi_\ell, \pi_i) \in \mathcal{A}$. We now claim that the algorithm we proposed with this definition of the set $K_{\pi_\ell \pi_i}$ obtains all the extreme supported solutions to

the ABSP problem with the bottleneck criterion in $\mathcal{O}(m^2)$ steps, where m is the number of arcs in the graph. Before we justify this claim we introduce one more lemma. This lemma provides an upper bound on the number of extreme supported solutions in the graph.

Lemma 8. Let c^1 be a bottleneck function on the arcs in $\mathcal{G}$. Then, for all nodes $i \in \mathcal{N} \setminus \{s\}$,

$$N_i \leqslant m + 1.$$

Proof. The proof builds on the idea that there are at most m unique values for any path for the first criterion. Consider, without loss of generality, the objective space $\mathcal{C}$ associated to an $s - i$ BSP problem. Next partition the set $\mathcal{E}(\mathcal{C})$ into $\mathcal{E}_1 \cup \mathcal{E}_2$, with $\varnothing \subsetneq \mathcal{E}_1 \subseteq \mathcal{E}(\mathcal{C})$ the set of points supported for $\lambda = 1$ and $\mathcal{E}_2 = \mathcal{E}(\mathcal{C}) \setminus \mathcal{E}_1$ the rest of the points. Since all the points in $\mathcal{E}_1$ have the same value for the first criterion, there can be at most two of them, one with the second criterion minimal and the other with the second criterion maximal. Now take any two points $c, \overline{c} \in \mathcal{E}_2$ for which $c^1 = \overline{c}^1$ and $c^2 < \overline{c}^2$. But now

$$\lambda c^1 + (1 - \lambda)c^2 < \lambda \overline{c}^1 + (1 - \lambda)\overline{c}^2$$

for any $\lambda \in [0, 1)$, so $\overline{c}$ is not supported for any λ in this interval. This implies that $\overline{c} \notin \mathcal{E}_2$ and we end up in a contradiction. Hence, every point in $\mathcal{E}_2$ must have a unique value for the first criterion. Since there are at most $m - 1$ unique values for the first criterion (one of them has points supported for $\lambda = 1$) and $N_i = |\,\mathcal{E}(\mathcal{C})\,|$, we get

$$N_i = |\,\mathcal{E}_1\,| + |\,\mathcal{E}_2\,| \leqslant 2 + (m - 1) = m + 1.$$

■

Notice how Lemma 8 implies that the total number of extreme supported solutions, N, is bounded, since

$$N = \sum_{i \in \mathcal{N}} N_i \leqslant \sum_{i \in \mathcal{N}} (m + 1) = n(m + 1).$$

We could stop here and argue that the ABSP bottleneck algorithm runs in $\mathcal{O}(nm^2)$ steps, by substituting this upper bound on N. However, when using the exact running time of the proposed algorithm, an even better time complexity is found.

Theorem 3. The ABSP algorithm for the bottleneck case runs in $\mathcal{O}(m^2)$ steps, where m is the number of arcs in the graph under consideration.

Proof. Immediate from Lemma 8 and Theorem 2. ■

As we can see, only minimum changes to the proposed algorithm are needed and a speed-up is found due to the structure of the extreme supported solutions under the bottleneck criterion function. Whether the algorithm from Sedeño–Noda and Raith (2015) can be used to find all the extreme supported solutions under a bottleneck criterion function is highly questionable. One would need to redefine the reduced costs $\overline{c}_{ij}$, the cost ratios θ_i, the shortest path tree $\mathcal{T}$ and tons of other variables and commands in the algorithm, and even then it remains uncertain whether such a ratio-labeling approach can be used to solve this problem.

10 Conclusion

In this thesis we considered a special subset of the solutions to the bi-objective shortest path problem. This subset being those solutions which minimize a convex combination of the considered criteria and whose cost are not a convex combination of the costs of other solutions in this set.

We first contributed to the literature by verifying the results from Sedeño–Noda and Raith (2015). We showed that their one-to-all algorithm is able to solve road network instances containing millions of extreme supported solutions in just a matter of minutes, confirming its applicibility to real life routing problems.

Our next contribution was the introduction of an $\mathcal{O}(Nm)$ algorithm to obtain the extreme supported solutions to the one-to-all acyclic BSP problem, given by a combination of a topological sorting phase along with a labeling and merging procedure. The algorithm turned out to not only be interesting from a theoretical perspective, but also from a computational perspective, outperforming the algorithm from Sedeño–Noda and Raith (2015) for sparse grid network instances. We then showed that this same algorithm can also be used to solve the ABSP problem with one or both objectives of the bottleneck type. For this algorithm a polynomial time complexity of $\mathcal{O}(m^2)$ in the number of arcs was achieved, due to the structure of the extreme supported solutions under this criterion function.

Our last contribution was to the BSP problem literature in general. While the arguments in this thesis are dedicated to finding the extreme supported solutions, most of them can almost directly be applied to finding the efficient solutions. Similar generalizations hold for other arguments made in this thesis, take for example the discussed lemmas. Most of them hold in the cyclic setting as well.

New research directions include further testing of the proposed algorithm. It might be interesting to see for what density of the graph we find a superior performance by our algorithm over the RLBSP, and for what density of the graph we find a similar performance. Another research direction would be to develop a fast ratio-labeling algorithm for the ABSP problem, since the RLBSP seems to be competitive with our algorithm for dense grid networks even without the exploitation of the acyclicness property. Finally, it might be interesting to see to what extend the methodology of this thesis can be used in a more general setting. One example would be the development of a (polynomial time) algorithm for finding the extreme supported solutions to the regular one-to-all BSP problem with at least one bottleneck objective.

Bibliography

R. K. Ahuja. Minimum cost-reliability ratio path problem. *Computers & operations research*, 15(1):83–89, 1988.

E. M. Arkin, J. S.B. Mitchell, and C. D. Piatko. Bicriteria shortest path problems in the plane. In *Proc. 3rd Canad. Conf. Comput. Geom*, pages 153–156, 1991.

R. Batta and S. S Chiu. Optimal obnoxious paths on a network: transportation of hazardous materials. *Operations Research*, 36(1):84–92, 1988.

M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2013.

R. Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

M. E. Captivo, Jo Climaco, . Figueira, r. Martins, and J. L. Santos. Solving bicriteria 0–1 knapsack problems using a labeling algorithm. *Computers & Operations Research*, 30(12): 1865–1886, 2003.

J. C. N. Climaco and E. Q. V. Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11(4):399–404, 1982.

T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

J. R. Current, C. S. Revelle, and J. L Cohon. The median shortest path problem: a multiobjective approach to analyze cost vs. accessibility in the design of transportation networks. *Transportation Science*, 21(3):188–197, 1987.

J. R. Current, C. S. Revelle, and J. L Cohon. An interactive approach to identify the best compromise solution for two objective shortest path problems. *Computers & Operations Research*, 17(2):187–198, 1990.

L. de Lima Pinto, C. T. Bornstein, and N. Maculan. The tricriterion shortest path problem with at least two bottleneck objective functions. *European Journal of Operational Research*, 198(2):387–391, 2009.

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

DIMACS. 9th dimacs implementation challenge - shortest paths. `http://www.dis.uniroma1.it/challenge9/download.shtml`, Last visited June 2017.

D. J. Eck. *Introduction to programming using Java*. David J. Eck, 2006.

M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *Or Spectrum*, 22(4):425–460, 2000.

E. Erkut and O. Alp. Designing a road network for hazardous materials shipments. *Com-*

puters & Operations Research, 34(5):1389–1405, 2007.

X. Gandibleux, F. Beugnies, and S. Randriamasy. Martins' algorithm revisited for multi-objective shortest path problems with a maxmin cost function. *4OR: A Quarterly Journal of Operations Research*, 4(1):47–59, 2006.

F. Guerriero and R. Musmanno. Label correcting methods to solve multicriteria shortest path problems. *Journal of optimization theory and applications*, 111(3):589–613, 2001.

C. Hallam, K.J. Harrison, and J.A. Ward. A multiobjective optimal path algorithm. *Digital Signal Processing*, 11(2):133–143, 2001.

H. W. Hamacher and G. Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52(4):209–230, 1994.

P. Hansen. Bicriterion path problems. In *Multiple criteria decision making theory and application*, pages 109–127. Springer, 1980.

M. I Henig. The shortest path problem with two objective functions. *European Journal of Operational Research*, 25(2):281–291, 1986.

A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11): 558–562, 1962.

J. D. Knowles and D. W. Corne. A comparison of encodings and algorithms for multiobjective minimum spanning tree problems. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 544–551. IEEE, 2001.

T. Lust and J. Teghem. The multiobjective traveling salesman problem: a survey and a new approach. In *Advances in Multi-Objective Nature Inspired Computing*, pages 119–141. Springer, 2010.

E. Q. V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

E. Q. V. Martins and J. L. E. Dos Santos. An algorithm for the quickest path problem. *Operations Research Letters*, 20(4):195–198, 1997.

J. Mote, I. Murthy, and D. L. Olson. A parametric approach to solving bicriterion shortest path problems. *European Journal of Operational Research*, 53(1):81–92, 1991.

M. Müller-Hannemann and K. Weihe. On the cardinality of the pareto set in bicriteria shortest path problems. *Annals of Operations Research*, 147(1):269–286, 2006.

K. Mulmuley and P. Shah. A lower bound for the shortest path problem. In *Computational Complexity, 2000. Proceedings. 15th Annual IEEE Conference on*, pages 14–21. IEEE, 2000.

A. Raith and M. Ehrgott. A comparison of solution strategies for biobjective shortest path

problems. *Computers & Operations Research*, 36(4):1299–1331, 2009.

A. Sedeño–Noda and A. Raith. A dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem. *Computers & Operations Research*, 57:83–94, 2015.

P. Serafini. Some considerations about computational complexity for multi objective combinatorial problems. In *Recent advances and historical development of vector optimization*, pages 222–232. Springer, 1987.

A. J.V. Skriver and K. A. Andersen. A label correcting approach for solving bicriterion shortest-path problems. *Computers & Operations Research*, 27(6):507–524, 2000.

M. Sniedovich. A new look at bellman's principle of optimality. *Journal of Optimization Theory and Applications*, 49(1):161–176, 1986.

S. Steiner and Tomasz R. Solving the biobjective minimum spanning tree problem using a k-best algorithm. Technical report, Citeseer, 2003.

R. E. Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6(2): 171–185, 1976.