A Dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem.

ANTONIO SEDEÑO-NODA (asedeno@ull.edu.es)
Departamento de Matemáticas, Estadística e Investigación Operativa. Universidad de La Laguna, C.P. 38271, San Cristóbal de La Laguna, Santa cruz de Tenerife, España.

ANDREA RAITH (a.raith@auckland.ac.nz)
Corresponding Author. Department of Engineering Science, The University of Auckland, Private Bag 92019, Auckland, New Zealand. Phone +64 (9) 3737599 extension 81977.

Abstract: We address the problem of determining all extreme supported solutions of the biobjective shortest path problem. A novel Dijkstra-like method generalizing Dijkstra's algorithm to this biobjective case is proposed. The algorithm runs in O(N(m+$n$log$n$)) time to solve one-to-one and one-to-all biobjective shortest path problems determining all extreme supported non-dominated points in the outcome space and one supported efficient path associated with each one of them. Here n is the number of nodes, m is the number of arcs and N is the number of extreme supported points in outcome space for the one-to-all biobjective shortest path problem. The memory space required by the algorithm is O(n+m) for the one-to-one problem and O(N+m) for the one-to-all problem. A computational experiment comparing the performance of the proposed methods and state-of-the-art methods is included.

Keywords Biobjective path problems; Supported efficient paths; Label-setting algorithm.

1. Introduction

In a directed network with arbitrary lengths, the *shortest path* (SP) problem is the problem of finding directed paths of shortest length from an origin node to all other nodes, or detecting a directed cycle of negative length. The SP problem is one of the most fundamental problems in network optimization. Numerous algorithms that solve SP problems and several real-world applications are reviewed in Ahuja et al. [1]. Also, the SP problem can be classified into one-to-one or one-to-all problems depending on whether the aim is to determine the shortest path from one origin node to one destination node or the shortest paths from one origin node to all other nodes in the network.

Algorithms for the standard single objective SP problems can generally be divided into two major groups: (1) Shortest path simplex algorithms that determine the optimal tree from an

initial feasible tree, making simplex pivots with non-tree arcs which do not satisfy *Bellman's optimality condition*, and (2) labeling methods that iteratively make distance labels *permanent* (guaranteeing they represent the length of an optimal path) for all nodes.

Labeling methods are also partitioned into *label setting* and *label correcting* methods. Label setting methods are characterized by setting one distance label of a known node permanent in each iteration of the method, while label correcting methods consider all distance labels as temporary until the end of the algorithm. For one-to-one SP problems label setting methods terminate when the distance label of the destination node becomes permanent. For one-to-all SP problems label setting methods terminate once the distance labels of all nodes in the network have become permanent.

In the literature, biobjective and multiobjective shortest path (BSP and MSP) problems have received considerable attention. There are different classes of solution approaches including labeling algorithms, the first of which were introduced by [2, 3] for BSP and MSP problems. More recently, computational comparisons [4, 5] investigate the performance of different labeling methods for MSP problems. Other methods include ranking approaches applied to BSP and MSP first introduced by [6], where solutions are ranked until it can be guaranteed all efficient solutions are found. A two phase method has been applied to BSP problems by [7]. It distinguishes two phases in the solution of a BSP problem, where in Phase 1 the so-called supported solutions are found (that are relatively easy to obtain as they can be found by solving weighted sum problems, for example). Then, the remaining (non-supported) solutions are obtained in Phase 2. An extensive computational comparison of different types of algorithms to solve BSP is conducted in [8].

While it is possible to solve large problem instances of BSP problems within reasonable computation time, problem instances become considerably harder to solve as network size increases. In particular, obtaining non-supported solutions can be costly. We focus here on the computation of supported solutions and show this can be achieved quickly and effectively for large real-world networks. We address solving one-to-one and one-to-all BSP problems separately.

Other researchers also focus on obtaining supported solutions of BSP and MSP problems. For example, White [9] discusses parametric shortest path problems, and Mote, Murthy, et. al. [7] solve a parametric version of the BSP problem to obtain supported paths in Phase 1 of the two phase method. Xie and Waller [10, 11] solve BSP and MSP problems approximately by using a parametric approach and follow this by a constrained shortest path method, leading to an overall approach which is not guaranteed to find all non-supported solutions. Henig [12]

considers decision making in the context of BSP problems, where it is assumed that a decision maker has a utility function that allows her to identify her preferred solution. A section of [12] is dedicated to the case for which only extreme supported efficient solutions of BSP have to be obtained, where three approaches are described. One solves a parametric problem, the other uses a dichotomic approach to explore weights and obtain all extreme efficient solutions as weighted sum problems, and one is based on labeling and works with sets of extreme labels.

Here, we propose a new algorithm to generate extreme supported solutions of BSP. While our algorithms only obtain a subset of solutions of a BSP problem, the obtained extreme supported solutions may be sufficient for decision making under some circumstances: the number of efficient solutions of BSP may be overwhelming and extreme supported solutions may be sufficient from a decision maker's point of view. Extreme supported solutions also give the decision maker an idea of the shape of the set of efficient solutions and may help identify regions of interest that can be further explored for instance using interactive decision making techniques capable of obtaining non-supported solutions, using approaches such as the ones proposed in [13, 14]. Also, Henig [12] notes that for quasiconvex utility functions the decision maker's choice is guaranteed to be among the supported efficient solutions. Another use of the proposed algorithm is in the two phase method. In Phase 1 only extreme supported solutions are required, and our proposed algorithms is a fast and remarkably reliable (in terms of runtime) method to solve Phase 1 of the two phase method for BSP.

The proposed algorithm is a Dijkstra-like algorithm that generalizes Dijkstra's algorithm for the BSP problem. In fact the proposed algorithm is a ratio-labeling algorithm storing labels associated with each extreme supported solution of the BSP problem for each node in the network. The labels used in the algorithm are precisely the slopes of the supported efficient frontier in the outcome space instead of the classical distance labels of nodes. The running times of the proposed algorithms are $O(N(m+n\log n))$. Moreover, many improvements of Dijkstra's algorithm can be directly applied to the proposed algorithms. We introduce two algorithms: one solving the one-to-one BSP problem using $O(n+m)$ space and the other one solving the one-to-all BSP problem using $O(N+m)$ space. The one-to-all algorithm stores additional labels to keep track of all extreme supported solutions. We compare the proposed algorithms and state-of-the-art algorithms for both one-to-one and one-to-all BSP problems. Numerical tests, reported in section 5 of this paper, show that the one-to-one BSP algorithm is robust and scalable in practice and the one-to-all BSP algorithm becomes the state-of-the-art

algorithm on synthetic and road networks (i.e. it is able to determine one hundred millions of extreme supported efficient solutions on the largest tested road network).

The paper is organized as follows: Section 2 describes the BSP problem and introduces some known results from the literature. In section 3 BSP is formulated as a parametric programming problem and a detailed study of the resolution of this problem is included in order to compute all extreme supported points in the outcome space of the BSP problem with a Dijkstra-like label setting strategy. Section 4 proposes the ratio-labeling algorithm for one-to-one BSP problems, and the ratio-labeling algorithm for storing the labels to keep track of all extreme supported solutions of the one-to-all BSP problem is included in a subsection. Section 4 also provides the worst-case time and space complexities of the proposed new algorithms and an example showing how the algorithm works. In section 5, computational experiments comparing performance of the proposed algorithms and other known algorithms are discussed. Finally, section 6 concludes with final comments and possible future avenues of investigation.

2. The biobjective shortest path problem.

Given a directed network $G = (V, A)$, let $V = \{1,..., n\}$ be the set of n nodes and A be the set of m arcs. Two real-valued costs $c_{ij} = (c_{ij}^1, c_{ij}^2)$ are associated with each arc $(i, j) \in A$. In a road network, these values can represent distance and time, respectively. We denote by $\Gamma_i^- = \{ j \in V \mid (j, i) \in A \}$ and by $\Gamma_i^+ = \{ j \in V \mid (i, j) \in A \}$ the set of predecessor and successor nodes for all nodes $i \in V$. Node s is the *origin* node and t the *destination* node. Let $i, j \in V$ be two distinct nodes of $G = (V, A)$, we define a directed path p_{ij} as a sequence $\langle i_1, (i_1, i_2), i_2, \cdots, i_{l-1}, (i_{l-1}, i_l), i_l \rangle$ of nodes and arcs satisfying $i_1 = i$, $i_l = j$ and for all $1 \le w \le l-1$, $(i_w, i_{w+1}) \in A$. The length of a directed path p is the sum of the arc lengths of the arcs that make up the path, that is, $c(p) = \sum_{(i,j) \in p} c_{ij}$. Let P be the set of paths from s to t in G.

Assumption 1 (w.l.o.g.). The network G contains a directed path from origin node s to any node $i \in V - \{s\}$ (if there is no path in G to some node i, then node i can be removed from G since it cannot lie on any s-t path).

If a flow x_{ij} is associated with each arc (i, j) then the following linear programming problem represents the *biobjective shortest path* (BSP) problem (see Ahuja et al. [1]):

$$\text{Minimize} \quad c(x) = \left(\sum_{(i,j)\in A} c_{ij}^1 x_{ij}, \sum_{(i,j)\in A} c_{ij}^2 x_{ij} \right) \tag{1}$$

subject to

$$\sum_{j\in\Gamma_i^+} x_{ij} - \sum_{j\in\Gamma_i^-} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ 0 & \forall i \in V - \{s,t\} \\ -1 & \text{if } i = t \end{cases} \tag{2}$$

$$x_{ij} \geq 0, \qquad \forall (i,j) \in A \tag{3}$$

where for an optimal solution x_{ij} has value 1 for all arcs (i, j) on the shortest path, and 0 otherwise. The above problem is a special case of the minimum cost network flow (MCNF) problem. The network simplex algorithm solves the above problem by taking advantage of the fact that every basis in the MCNF problem is also a spanning tree $T \subseteq A$ of *G*. Let *X* be the polyhedron defined by constraints (2)-(3) (*decision space*) and let its image under the objective function be *C* = *c*(*X*) (*outcome space*). The next two literature results hold (Ahuja et al. [1]): (*i*) *Any feasible solution of the BSP problem is a vertex of X and vice-versa* and (*ii*) *Every vertex of X is associated with a directed spanning tree rooted at s*.

A *directed out-spanning tree* is a spanning tree rooted at node *s* such that the unique path in the tree from root node *s* to every other node is a directed path. The predecessor node of node *i* in the tree is denoted by $pred_i(T)$, for all nodes $i \in V - \{s\}$. In the remainder of the paper, we refer to a directed out-spanning tree as tree.

Distance labels associated with a tree *T* are obtained by setting $d_s^v(T) = 0$ and solving $c_{ij}^v + d_i^v(T) - d_j^v(T) = 0$, $\forall (i,j) \in T$ for *v* = 1,2. Given a tree *T* and associated distance labels, reduced costs are defined as $\bar{c}_{ij}^v(T) = c_{ij}^v + d_i^v(T) - d_j^v(T)$, $\forall (i,j) \in A$ for *v* = 1,2.

Assumption 2. The network *G* does not contain a directed cycle with negative length for each single-objective SP problem, i.e. taking into account costs c^1 and c^2, respectively.

Definition 1. A path (feasible solution) $p \in P$ ($x \in X$) is called *efficient* if there does not exist any $p' \in P$ ($x' \in X$) with $c^1(p') \leq c^1(p)$ $(c^1(x') \leq c^1(x))$ and $c^2(p') \leq c^2(p)$ $(c^2(x') \leq c^2(x))$ with at least one inequality being strict. The image $c(p)$ $(c(x))$ of and efficient path *p* (or solution *x*) is called *non-dominated point*.

Definition 2. *Supported efficient paths* (*supported efficient solutions*) are those efficient paths (efficient solutions) that can be obtained as optimal paths (solutions) of a weighted sum problem $\min_{p \in P} \left(\lambda_1 c^1(p) + \lambda_2 c^2(p) \right)$ $\left(\min_{x \in X} \left(\lambda_1 c^1(x) + \lambda_2 c^2(x) \right) \right)$ for some $\lambda_1 > 0$ and $\lambda_2 > 0$. All other efficient paths (solutions) are called *non-supported*.

The *supported non-dominated points* lie on the lower-left boundary of the convex hull (*conv(C)*) of the feasible set *C* in *outcome space*, whereas non-supported solutions lie in the interior of *conv(C)*. This is illustrated in Figure 1.

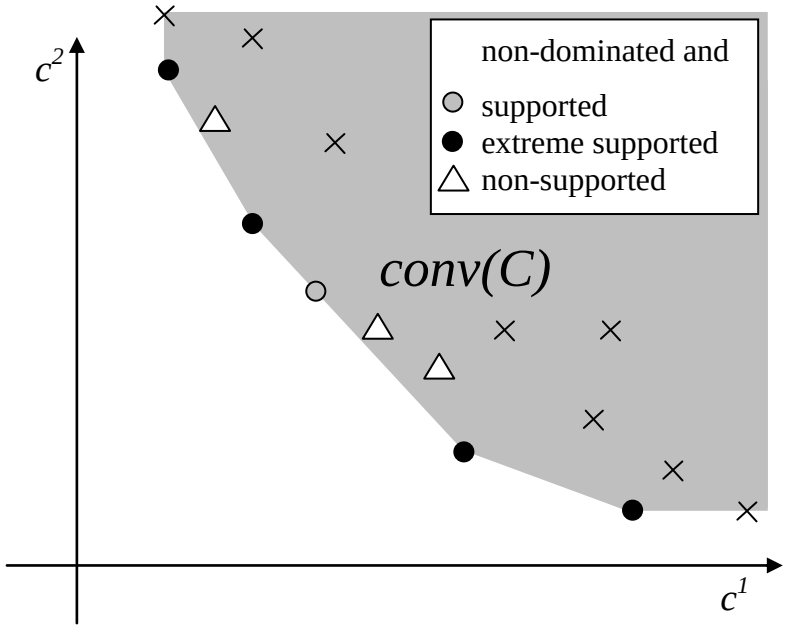

Figure 1. Illustration of *conv(C)* in outcome space.

The focus of this paper is to design a fast algorithm to determine all paths (solutions) associated with *extreme supported non-dominated points* in the BSP problem. Those paths (solutions) are denoted *extreme supported path (extreme supported solution)*. That is, we want to determine supported efficient paths (supported efficient solutions) whose images are vertices of the convex hull of the supported non-dominated points. We ensure the proposed algorithm computes one supported efficient path for each extreme supported non-dominated point. From assumption 2, it follows that for each extreme supported non-dominated point, a supported efficient path exists that is a simple path (i.e. a path without repeating nodes).

Additionally, it is easy to show that any supported efficient path, being optimal solution of the weighted sum problem for fixed $\lambda_1 > 0$ and $\lambda_2 > 0$, satisfies the *optimality principle*. That

is, any sub-path p_{ij} of a given supported efficient path p_{st} is a supported efficient path within the set of paths from node *i* to node *j* for the same values of λ_1, λ_2. The proof is immediate from the fact that the objective function of the weighted sum problem corresponds to that of a single objective shortest path problem where the arcs have length $\lambda_1 c^1 + \lambda_2 c^2$.

Computing all supported efficient *s*-*t* paths can also be achieved by identifying all supported efficient *s*-*i* paths for all $i \in V - \{s\}$. Therefore, we recast our problem to identify all extreme supported solutions of the biobjective one-to-all shortest path tree problem, that is,

$$\text{Minimize} \quad c(x) = \left(\sum_{(i,j)\in A} c_{ij}^1 x_{ij}, \sum_{(i,j)\in A} c_{ij}^2 x_{ij} \right) \qquad (1')$$

subject to

$$\sum_{j\in\Gamma_i^+} x_{ij} - \sum_{j\in\Gamma_i^-} x_{ji} = \begin{cases} n-1 & \text{if } i = s \\ -1 & \forall i \in V - \{s\} \end{cases} \qquad (2')$$

$$x_{ij} \geq 0 \qquad \forall (i,j) \in A \qquad (3')$$

Formulation (1') - (3') is a biobjective shortest path tree problem which is also used for example by [7]. It is well known that both formulations (1) – (3) and (1') – (3') will naturally result in integer solutions (corresponding to a shortest path or shortest path tree, respectively, see Ahuja et al. [1]).

3. Solving the BSP problem as parametric program.

Instead of solving the weighted sum problem $\min_{x\in X} \left(\lambda_1 c^1(x) + \lambda_2 c^2(x)\right)$ with $\lambda_1 + \lambda_2 = 1$ and $\lambda_1 \geq 0, \lambda_2 \geq 0$, the problem $\min_{x\in X} \left(c^1(x) + \theta c^2(x)\right)$ with $\theta \in [0, +\infty)$ can be solved alternatively. For the linear programming formulation, this leads to a *parametric linear program.*

Solving a biobjective linear program such as (1')-(3') above as a parametric program works by initially obtaining a lexicographic solution to the problem, i.e. an efficient solution is obtained which is optimal for $\theta = 0$. The parametric method then solves the problem by iteratively pivoting variables while ensuring that all optimal solutions for increasing values of θ are obtained, which correspond to supported efficient solutions of the biobjective linear program, until ultimately the other lexicographic solution is reached.

Note that we obtain a shortest path tree minimizing $c^1(x)$ when we solve the parametric program for $\theta = 0$, but the corresponding shortest path tree could contain a dominated path as

the second objective is not taken into account. To avoid this, the method starts with an optimal shortest path tree T^* derived from the optimal solution of the parametric program for $\theta = \varepsilon > 0$ where ε is sufficiently small [15]. This initial solution T^* is obtained by solving the lexicographic optimization problem $\text{lex}\min_{x \in X} \left(c^1(x), c^2(x) \right)$ (see Isermann [16]). Path p_1 derived from T^* is a supported efficient path minimizing the first objective. The image of this path is a supported non-dominated extreme point of the convex hull of the outcome space. For this tree T^*, we denote the distance labels $d_i = (d_i^1, d_i^2) = (d_i^1(T^*), d_i^2(T^*))$ and the predecessor labels $pred_i = pred_i(T^*)$ for any node $i \in V$ (see Ahuja *et al.*[1]). Note that a path that has a first objective value equal to d_i^1 and a second objective value smaller than d_i^2 cannot arise for any node *i* in *V* as T^* corresponds to a lexicographically optimal solution. This tree T^* will be optimal for a range of values of θ, and the optimality interval for T^* is the range of θ-values for which T^* *is an optimal solution* of the parametric program.

The reduced cost values of arc (*i, j*) represent the change in the distance labels of node *j* if arc (*i, j*) replaces arc ($pred_j$, *j*) in the current tree. When including arc (*i, j*) in the tree the current label of node *j*, (d_j^1, d_j^2), becomes label $(d_j^1 + \bar{c}_{ij}^1, d_j^2 + \bar{c}_{ij}^2)$. Note that $\bar{c}_{ij}^1 = 0$ and $\bar{c}_{ij}^2 = 0$ for any arc $(i, j) \in T^*$ and $\bar{c}_{ij}^1 \geq 0$ for any arc $(i, j) \in A$. The current optimal solution of the parametric linear program remains optimal as long as the reduced costs (in terms of the objective of the parametric linear program), remain non-negative. This is the case as long as the following remains true:

$$\bar{c}_{ij}^1 + \theta \bar{c}_{ij}^2 \geq 0 \text{ for all } (i, j) \in A .$$

In order to move from the current efficient solution to another efficient one the first objective has to worsen, or $\bar{c}_{ij}^1 \geq 0$, and the second one has to improve, or $\bar{c}_{ij}^2 < 0$. Therefore, the tree T^* remains optimal for all θ in the range $0 \leq \theta \leq \theta^k$ where $\theta^k = \min_{(i,j) \in A} \left\{ \theta_{ij}^k \right\}$ and

$$\theta_{ij}^k = \begin{cases} -\bar{c}_{ij}^1 / \bar{c}_{ij}^2 & \text{if } \bar{c}_{ij}^2 < 0 \\ +\infty & \text{otherwise} \end{cases}$$

That is, if $\bar{c}_{ij}^2 < 0$ the ratio associated with arc (*i, j*) with index *k* (where *k* = 1 initially) is $-\bar{c}_{ij}^1 / \bar{c}_{ij}^2$; otherwise this ratio is infinite. To determine the tree corresponding to the next supported non-dominated point, the next arc to be included in the tree is identified as:

$$(x, y) = \arg \text{lex} \min_{(i,j) \in A} \left\{ (\theta_{ij}^k, \bar{c}_{ij}^2) : \theta_{ij}^k < +\infty \right\} .$$

That is, we must determine the arc with minimum finite ratio, which then enters the tree, and an arc from the current optimal tree leaves it. If more than one arc with minimum ratio exists, we select one with smallest (or most negative) value of $\overline{c}_{ij}^{2}$ (this is why lex min appears in the above expression). Arc $(pred_y, y)$ is then replaced by arc (*x*, *y*) giving the next solution with index *k+1*.

Instead of following the standard steps of a parametric programming method from here, we now discuss how to integrate the above ideas with a Dijkstra-like label setting shortest path method. Having identified the arc (*x*, *y*) to include into the shortest path tree, we would next update the distance label of node *y*. We propose to identify arcs and node labels iteratively which are updated in a Dijkstra-like fashion based on ratios θ as outlined in the following. Eventually updated distance labels optimal with respect to the current ratio may reach the target node *t* indicating a new shortest path is found. This requires updating reduced costs and ratios θ as well as distance labels. Therefore, we investigate the ratios θ_{ij}^{k+1} for all $(i, j) \in A$ when the arc (*x*, *y*) substitutes arc ($pred_y$, *y*). In this case, (d_y^1, d_y^2) becomes $(d_y^1 + \overline{c}_{xy}^1, d_y^2 + \overline{c}_{xy}^2)$ because the only change occurs in the distance label of node *y*. Therefore, only the ratios of arcs arriving at, or leaving from, node *y* can change. In the following we observe how ratios θ associated with arcs (*i*, *y*) from predecessor nodes $i \in \Gamma_y^-$ change, see (1)-(4) below, and also how they change for arcs (*y*, *i*) associated with successor nodes $i \in \Gamma_y^+$, see (5)-(7) below.

Considering arcs (*i*, *y*) for all nodes $i \in \Gamma_y^-$, the new ratio is $\theta_{iy}^{k+1} = -\dfrac{\overline{c}_{iy}^1 - \overline{c}_{xy}^1}{\overline{c}_{iy}^2 - \overline{c}_{xy}^2}$ wherever $\overline{c}_{iy}^2 - \overline{c}_{xy}^2 < 0$; otherwise it is $+\infty$. Since

$$\theta_{xy}^k = -\frac{\overline{c}_{xy}^1}{\overline{c}_{xy}^2} \le \theta_{iy}^k = -\frac{\overline{c}_{iy}^1}{\overline{c}_{iy}^2} \qquad \text{(A)},$$

we obtain that:

(1) $$\theta_{iy}^{k+1} = -\frac{\overline{c}_{iy}^1 - \overline{c}_{xy}^1}{\overline{c}_{iy}^2 - \overline{c}_{xy}^2} = \frac{\overline{c}_{xy}^1}{\overline{c}_{xy}^2} \frac{\left(-\frac{\overline{c}_{iy}^1}{\overline{c}_{xy}^1}\overline{c}_{xy}^2 + \overline{c}_{xy}^2\right)}{\overline{c}_{iy}^2 - \overline{c}_{xy}^2} \ge \frac{\overline{c}_{xy}^1}{\overline{c}_{xy}^2} \frac{\left(-\overline{c}_{iy}^2 + \overline{c}_{xy}^2\right)}{\overline{c}_{iy}^2 - \overline{c}_{xy}^2} = -\frac{\overline{c}_{xy}^1}{\overline{c}_{xy}^2} = \theta_{xy}^k \text{ by (A)},$$

(2) $$\theta_{iy}^{k+1} = -\frac{\overline{c}_{iy}^1 - \overline{c}_{xy}^1}{\overline{c}_{iy}^2 - \overline{c}_{xy}^2} = \frac{\overline{c}_{iy}^1}{\overline{c}_{iy}^2} \frac{\left(-\overline{c}_{iy}^2 + \overline{c}_{iy}^2 \frac{\overline{c}_{xy}^1}{\overline{c}_{iy}^1}\right)}{\overline{c}_{iy}^2 - \overline{c}_{xy}^2} \ge \frac{\overline{c}_{iy}^1}{\overline{c}_{iy}^2} \frac{\left(-\overline{c}_{iy}^2 + \overline{c}_{xy}^2\right)}{\overline{c}_{iy}^2 - \overline{c}_{xy}^2} = -\frac{\overline{c}_{iy}^1}{\overline{c}_{iy}^2} = \theta_{iy}^k \text{ by (A)},$$

(3) If $\theta_{xy}^k = \theta_{iy}^k$, then $\theta_{iy}^{k+1} = +\infty$, because $\overline{c}_{iy}^2 \ge \overline{c}_{xy}^2$ and $\overline{c}_{iy}^2 - \overline{c}_{xy}^2 \ge 0$,

(4) If $\theta_{iy}^{k} = +\infty$, then $\theta_{iy}^{k+1} = +\infty$, because $\overline{c}_{iy}^{2} \geq 0$ and $\overline{c}_{iy}^{2} - \overline{c}_{xy}^{2} > 0$.

We conclude that the ratio of any arc (*i*, *y*) with $i \in \Gamma_{y}^{-}$ increases (2)-(3) or keeps (4) its previous value and, according to (1), it always is greater than or equal to the ratio of the arc (*x*, *y*). In particular, if the ratio of the arc (*i*, *y*) equals the ratio of the arc (*x*, *y*), the ratio of (*i*, *y*) becomes +∞ (3).

Considering arcs (*y*, *i*) for all nodes $i \in \Gamma_{y}^{+}$, the new ratio is $\theta_{yi}^{k+1} = -\dfrac{\overline{c}_{yi}^{1} + \overline{c}_{xy}^{1}}{\overline{c}_{yi}^{2} + \overline{c}_{xy}^{2}}$ wherever $\overline{c}_{yi}^{2} + \overline{c}_{xy}^{2} < 0$; otherwise it becomes $+\infty$. Since

$$\theta_{xy}^{k} = -\frac{\overline{c}_{xy}^{1}}{\overline{c}_{xy}^{2}} \leq \theta_{yi}^{k} = -\frac{\overline{c}_{yi}^{1}}{\overline{c}_{yi}^{2}} \qquad \text{(B)},$$

we obtain that:

(5) $$\theta_{yi}^{k+1} = -\frac{\overline{c}_{yi}^{1} + \overline{c}_{xy}^{1}}{\overline{c}_{yi}^{2} + \overline{c}_{xy}^{2}} = \frac{\overline{c}_{xy}^{1}}{\overline{c}_{xy}^{2}} \frac{\left(-\dfrac{\overline{c}_{yi}^{1}}{\overline{c}_{xy}^{1}} \overline{c}_{xy}^{2} - \overline{c}_{xy}^{2} \right)}{\overline{c}_{yi}^{2} + \overline{c}_{xy}^{2}} \geq \frac{\overline{c}_{xy}^{1}}{\overline{c}_{xy}^{2}} \frac{\left(-\overline{c}_{yi}^{2} - \overline{c}_{xy}^{2} \right)}{\overline{c}_{yi}^{2} + \overline{c}_{xy}^{2}} = -\frac{\overline{c}_{xy}^{1}}{\overline{c}_{xy}^{2}} = \theta_{xy}^{k} \text{ by (B)},$$

(6) $$\theta_{yi}^{k+1} = -\frac{\overline{c}_{yi}^{1} + \overline{c}_{xy}^{1}}{\overline{c}_{yi}^{2} + \overline{c}_{xy}^{2}} = \frac{\overline{c}_{yi}^{1}}{\overline{c}_{yi}^{2}} \frac{\left(-\overline{c}_{yi}^{2} - \overline{c}_{yi}^{2} \dfrac{\overline{c}_{xy}^{1}}{\overline{c}_{yi}^{1}} \right)}{\overline{c}_{yi}^{2} + \overline{c}_{xy}^{2}} \leq \frac{\overline{c}_{yi}^{1}}{\overline{c}_{yi}^{2}} \frac{\left(-\overline{c}_{yi}^{2} - \overline{c}_{xy}^{2} \right)}{\overline{c}_{yi}^{2} + \overline{c}_{xy}^{2}} = -\frac{\overline{c}_{yi}^{1}}{\overline{c}_{yi}^{2}} = \theta_{yi}^{k} \text{ by (B)},$$

(7) If $\theta_{xy}^{k} = \theta_{yi}^{k}$, then $\theta_{yi}^{k+1} = \theta_{xy}^{k} = \theta_{yi}^{k}$ following the same arguments as in (5).

That is, the ratio of any arc (*y*, *i*) with $i \in \Gamma_{y}^{+}$ decreases (6) or keeps (7) its previous value and, according to (5), it always is greater than or equal to the ratio of the arc (*x*, *y*). In particular, if the ratio of the arc (*y*, *i*) equals the ratio of the arc (*x*, *y*), the ratio of (*y*, *i*) does not change (7).

From (1)-(7) above, we can conclude that distance labels of paths optimal with respect to a current (minimal) ratio θ will iteratively be propagated through the tree as eventually all arcs with a current minimal ratio will be included in the path tree. If the propagated labels reach the target node *t*, a path that is optimal with respect to θ (and hence is a supported efficient solution) has been obtained. This is summarized in the following theorem:

Theorem 1. *The set of extreme supported non-dominated points of the biobjective shortest s-i path problem for all nodes* $i \in V - \{s\}$ *can be determined starting from an optimal path tree minimizing* $\text{lex}\min(c^1(x), c^2(x))$ *and making a sequence of arc interchanges, i.e. replacing* (*pred*$_y$, *y*) *by* (*x*, *y*), *where* $(x, y) = \arg\text{lex}\min_{(i,j)\in A}\{(\theta_{ij}, \overline{c}_{ij}^2) : \theta_{ij} < +\infty\}$ *, until the ratio of any arc in A is* $+\infty$.

Proof. Since the set of supported efficient paths is connected (see Steuer [15]), we can enumerate them by a sequence of arc interchanges (*pred*$_y$, *y*) by (*x*, *y*). From assumption 2, there are no negative length cycles in *G*, and therefore, any arc interchange produces a tree that is adjacent to the previous one and both represent supported efficient solutions of problem (1') – (3'). Therefore, starting from the supported efficient path tree corresponding to the shortest path tree minimizing $c^1(x)$, we identify the inferior boundary of the convex hull of the outcome space by iteratively identifying the arc $(x, y) = \arg\text{lex}\min_{(i,j)\in A}\{(\theta_{ij}^k, \overline{c}_{ij}^2) : \theta_{ij}^k < +\infty\}$ to make the interchange replacing (*pred*$_y$, *y*) by (*x*, *y*). This is true since the ratio of this arc indicates the minimum slope (from the left to right in outcome space, see also Figure 1) to reach an adjacent path tree in each iteration. Also, the *s*-*y* path tree obtained after the interchange is an optimal solution of the parametric program (1') – (3') with $\theta = \theta_{xy}$. This last claim is true since any other arc has a ratio greater than or equal to θ_{xy} and never will become inferior to θ_{xy} as shown in equations (1)-(7). Furthermore, any arc (*y*, *i*) with ratio θ_{xy} before the interchange, and improving the second objective, keeps its ratio value after the interchange, see equation (7).

Identifying the extreme supported non-dominated points means finding paths from *s* to *y* for all $y \in V - \{s\}$ such that the smallest ratio θ for which it is an optimal solution (or path tree) of (1') – (3') is greater than the corresponding smallest ratio θ of the previous extreme supported non-dominated solution (or path tree). Therefore, the next supported extreme solution (and corresponding non-dominated point) is obtained as soon as ratios θ associated with all arcs are all strictly greater than the ratio for which the previous extreme supported solution was obtained.

As arc interchanges are made in order of increasing ratio θ, and ratios of arcs emanating from a node *y* never decrease after an arc interchange, see equations (1)-(7), we know that all

supported efficient solutions are visited until eventually all ratios θ are $+\infty$ indicating the lexicographically minimal solution for the second objective is found. □

While not necessary for the correctness of Theorem 1, we note that the criterion to select the arc (*x*, *y*) determines the arc with minimum ratio and, if more than one arc exists, the criterion selects the one with most negative value of $\overline{c}^2_{xy}$ to accelerate the computation of the extreme supported non-dominated points.

From this result, in the next section we introduce a labeling algorithm that enumerates all extreme supported non-dominated points of the biobjective one-to-one *s*-*t* paths.

4. A ratio-labeling algorithm for one-to-one *s*-*t* BSP

The algorithm initially computes a shortest path tree considering the first objective function. For this tree, distance labels $d_i = (d_i^1, d_i^2)$ and predecessor labels $pred_i$ for any node $i \in V$ are stored. Note that these labels correspond to the solution of a $\operatorname{lex\,min}_{p \in P} \left(c^1(p), c^2(p) \right)$ problem. This problem can be solved by adapting Dijkstra's [17] algorithm in such a way that the label of any node *i* is modified when the distance label d_i^1 is improved or when there is a tie in d_i^1-values but d_i^2 can be improved.

Starting with the initial shortest path tree, the proposed algorithm works by updating the labels d_i and $pred_i$ corresponding to the current path tree. An implementation of Dijkstra's algorithm needs a heap to store labels of each node *i* in *V*. These labels are

$$\left(\theta_i, \hat{c}_i^2 \right) = \operatorname{lex\,min}_{j \in \Gamma_i^-} \left\{ \left(-\overline{c}_{ji}^1 / \overline{c}_{ji}^2, \overline{c}_{ji}^2 \right) : \overline{c}_{ji}^2 < 0 \right\}$$

That is, the algorithm stores, in a heap *H*, the minimum ratio $-\overline{c}_{ji}^1 / \overline{c}_{ji}^2$ of the incoming arcs (*j*, *i*) of node *i* such that $\overline{c}_{ji}^2 < 0$ and $\hat{c}_i^2 = \overline{c}_{ji}^2$, for any node *i*. In case two or more arcs arriving at node *i* have the same minimal ratio, the algorithm stores the ratio of the arc with smaller value of $\overline{c}_{ji}^2$ in the heap *H*. Also, for each node *i*, the algorithm keeps track of ratio θ_i, reduced cost $\hat{c}_i = (\hat{c}_i^1, \hat{c}_i^2)$ and the candidate predecessor $Cpred_i$ obtained from the above ratio expression to be able to update labels easily. The algorithm uses an array or vector *Cpath* to store the current *s*-*t* path for which the optimality interval in the parametric linear program is under investigation. Variable *lastratio* is used to identify the lower bound of the optimality

interval for each extreme supported non-dominated path. The algorithm keeps track of distances for both objectives for the current *s*-*t* path in *d*1 and *d*2; node predecessors in the tree are stored in *pred* and node distances are stored in d^1, d^2. *Cpath* stores the current path.

The following heap operations are needed in our algorithm (see [18]): *CreateHeap*(*H*), *Insert*({*labels*}, *H*), *Find-min*(*H*), *Decrease-key*({*labels*}, *H*), and *Delete-min*(*H*).

Algorithm 1: *Ratio-Labeling BSP* (RLBSP) **Algorithm**;

(1) Let T^* be an optimal tree of $\operatorname{lex}\min_{p\in P}\left(c^1(p), c^2(p)\right)$ and store d and *pred* labels of T^*;

(2) *CreateHeap*(*H*); Use labels *pred* to store the current *s*-*t* path in *Cpath*; *lastratio* = 0; $d1 = d_t^1$; $d2 = d_t^2$;

(3) Set $\theta_i = +\infty; Cpred_i = 0;$ for all $i \in V - \{s\}$;

(4) Compute $(\theta_i, \hat{c}_i^2, Cpred_i) = \operatorname{lex}\min_{j\in\Gamma_i^-}\left\{(-\bar{c}_{ji}^1 / \bar{c}_{ji}^2, \bar{c}_{ji}^2, j) : \bar{c}_{ji}^2 < 0\right\}\ \forall i \in V - \{s\}$;

(5) **For** all $i \in V - \{s\}$ **do** **If** $(Cpred_i \neq 0)$ **then** *Insert*$\left(\left(\theta_i, \hat{c}_i^2, i\right), H\right); \hat{c}_i^1 = \bar{c}_{Cpred_i i}^1$;

(6) **While** $(H \neq \varnothing)$ **do**

(7) $\left(\theta_i, \hat{c}_i^2, i\right) =$ *find-min*(*H*); $d_i = d_i + \hat{c}_i$; $pred_i = Cpred_i$;

(8) *Delete-min*(*H*);

(9) **If** (*i*==*t*) **then**

(10) **If** $(\theta_i > lastratio)$ **then**

(11) Use *Cpath* to print the current path with distances (*d*1,*d*2) being optimal for the parametric program with θ in [lastratio, θ_i]; *lastratio* = θ_i;

(12) Use labels *pred* to store the current *s*-*t* path in *Cpath*; $d1 = d_t^1$; $d2 = d_t^2$;

(13) $\theta_i = +\infty; Cpred_i = 0;$ Compute $(\theta_i, \hat{c}_i^2, Cpred_i) = \operatorname{lex}\min_{j\in\Gamma_i^-}\left\{(-\bar{c}_{ji}^1 / \bar{c}_{ji}^2, \bar{c}_{ji}^2, j) : \bar{c}_{ji}^2 < 0\right\}$;

(14) **If** $(Cpred_i \neq 0)$ **then** *Insert* $\left(\left(\theta_i, \hat{c}_i^2, i\right), H\right); \hat{c}_i^1 = \bar{c}_{Cpred_i i}^1$;

(15) **For** all $j \in \Gamma_i^+$ **do**

(16) **If** $(\bar{c}_{ij}^2 < 0)$ **then**

(17) **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**

(18) **If** $(Cpred_i == 0)$ **Then** *Insert*$\left(\left(-\bar{c}_{ij}^1 / \bar{c}_{ij}^2, \bar{c}_{ij}^2, j\right), H\right)$;

(19) **Else** *decrease-key*$\left(\left(-\bar{c}_{ij}^1 / \bar{c}_{ij}^2, \bar{c}_{ij}^2, j\right), H\right)$;

(20) $\theta_j = -\bar{c}_{ij}^1 / \bar{c}_{ij}^2; \hat{c}_j = \bar{c}_{ij}; Cpred_j = i;$

(21) Use *Cpath* to print the current path with distances (*d*1,*d*2) being optimal for the parametric program with θ in [lastratio, $+\infty$];

Algorithm 1 above is a standard label setting (Dijkstra) algorithm except for lines 9-14. In lines 13-14 the next ratio for node *i*, when node *i* becomes the candidate in the current solution, is calculated and stored (case (2)-(4)). Lines 15-21 represent cases (6)-(7) in section 3 dealing with outgoing arcs of node *i*. Lines 9-12 identify whether the current minimum element in the heap corresponds to node *t* (line 9) and verifies if the current path is associated with an extreme supported non-dominated point (line 10). If both are the case, the algorithm prints the current path stored in *Cpath* and updates *lastratio* (lines 11-12 and 21). The current

s-*t* path is always stored in *Cpath*. In other words, the algorithm identifies one *s*-*t* path for each extreme supported non-dominated point in the outcome space of the BSP problem. The sequences of the calculated ratios are the slopes of the segments in the inferior convex hull (non-dominated frontier). The algorithm can be modified to store all *s*-*i* paths for each extreme supported non-dominated point in the outcome space of the one-to-all BSP with origin *s* and destination *i*, for all nodes $i \in V - \{s\}$ (as shown in next subsection).

If we denote by $N = \sum_{i=1}^{n} N_i$ the number of extreme supported non-dominated points in the outcome space of BSP, where N_i is the number of extreme supported non-dominated solutions of the *s*-to-*i* BSP, we can conclude that

Theorem 2. *The RLBSP algorithm runs in* O(N (*m*+*nlogn*)) *time and uses* O(*n*+*m*) *space*.

Proof. The solution of the $\operatorname{lex}\min_{p \in P} \left(c^1(p), c^2(p) \right)$ using Dijkstra's algorithm [17] needs *O*(*m*+*n*log*n*) time (see [2]). Note that the size of the heap *H* in the RLBSP algorithm is at most *n*, for example $Cpred_i$ takes value 0 only when node *i* is extracted from the heap. In addition, any heap operation takes constant time with the exception of the *Delete-min* operation which requires O(log*n*) time when a Fibonacci heap is used (see Fredman and Tarjan [19]). We need to know the maximum number of iterations between two trees containing two paths being extremes in the outcome space. For that, suppose that an iteration starts with a different value of θ compared to the previous iteration. This means that the algorithm determined a new extreme supported non-dominated point in the previous iteration representing an s-to-*i* path for one node $i \in V - \{s\}$. Let $\theta_{\min}$ be the minimum ratio extracted from the heap. We must determine the maximum number of iterations until a ratio greater than $\theta_{\min}$ is extracted from the heap. Note that at most *n*-1 nodes have a ratio in the heap with value $\theta_{\min}$. Therefore, after at most *n*-1 iterations, the ratio of some node *j* extracted from the heap becomes greater than $\theta_{\min}$, otherwise there is a negative length cycle where all the arcs have ratio $\theta_{\min}$ in the cycle and negative reduced costs for the second objective. The latter is a contradiction to assumption 2. Therefore, the proposed algorithm has at most nN iterations, and since $n = |V|$ the worst-case time complexity of the algorithm is

$$O(N(\sum_{i \in V}(\log n + |\Gamma_i^-| + |\Gamma_i^+|)) + n) = O(N(n \log n + m)).$$

The term n outside the sum represents the time needed to store or print *Cpath* for each extreme point. Finally, it is easy observe that the space used by the algorithm is O(n+m) when a candidate *s*-*t* path for each extreme supported non-dominated point must be calculated. □

4.1 An example.

In this section, we show the execution of the RLBSP algorithm on the small graph shown in Figure 2. The graph contains four nodes and six arcs. There are four simple paths from node 1 to node 4 that are 1→2→3→4 with lengths (1,9), 1→2→4 with lengths (3,3), 1→3→4 with lengths (4,2) and 1→3→2→4 with lengths (8,1). All these paths are extreme supported paths and their images, or length vectors according to the two length functions, are extreme supported non-dominated points in the outcome space.

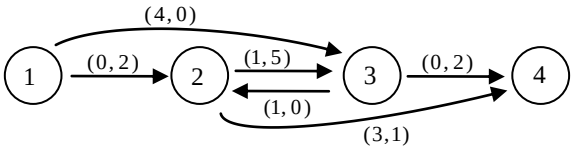

Figure 2. Graph with two lengths on the arcs.

The labels used in the algorithm are shown in Table 1. The first column *iter* indicates the iteration of the algorithm. An iteration of the algorithm corresponds to one execution of lines 7-20 in Algorithm 1. In each row of the table, the modified variables at the end of the iteration are highlighted. The initialization of the algorithm in lines 1-5 is shown as iteration 0 (*iter* = 0) in the table. The final row in the table shows the execution of the line 21. This last line prints the last extreme supported path. The second column, *Heap,* represents the heap *H* that stores the labels (θ^i, $\hat{c}_i^2$, *i*). The third column, *i*, identifies the node extracted in line 7, that is, the node with the minimum ratio in the heap. The remaining columns, with the exception of the last column, show the values of labels (d_i^1, d_i^2), $(Cpred_i, pred_i)$ and θ^i for any node *i* in {2,3,4} (node 1 is not included because it is the origin node of the paths and there are not arcs arriving at node 1). Finally, the last column shows the path stored in *Cpath* and the values of (*d*1, *d*2) for each printed path in line 11 or 21.

Initially, the shortest path tree contains the arcs {(1,2), (2,3), (3,4)}. The labels for this tree are (d_i^1, d_i^2) and the vector $pred_i$ shown for *iter* = 0. The current *s*-*t* path stored in *Cpath* is

$1\rightarrow2\rightarrow3\rightarrow4$ with ($d1$,$d2$) = (1,9), and *lastratio* initially has value zero. The heap contains the labels (1/3,-6,4) and (3/7,-7,3) associated with arcs (2,4) and (1,3) because their reduced costs are (2,-6) and (3,-7), respectively. Arc (3,2) has reduced costs (5,2) and the tree arcs have reduced costs (0,0). Clearly, these arcs have a non-negative reduced costs associated with the second length and, therefore, they are not used in the calculation of the ratios θ^i for any node *i* in {2, 3, 4}. Thus, the initial ratios are $\theta^2 = +\infty$, $\theta^3 = -\frac{3}{-7} = \frac{3}{7}$, $\theta^4 = -\frac{2}{-6} = \frac{1}{3}$ with the vector *Cpred* = (0,0,1,2). In the remaining rows of the table, the changes of variables by the algorithm appear in bold. In the first iteration, the label (1/3,-6,4) is extracted from the heap. Since $i = t = 4$ and *lastratio* is zero, which is less than $\theta^4 = \frac{1}{3}$, the current *Cpath* = $1\rightarrow2\rightarrow3\rightarrow4$ is printed on the screen. This path is optimal for the parametric program with θ in $\left[0, \frac{1}{3}\right]$. Now, *Cpath* is updated to be $1\rightarrow2\rightarrow4$ with ($d1$,$d2$) = (3,3), but its optimality interval is still unknown. The reduced costs of the arcs (2,4) and (3,4) become (0,0) and (-2,6), respectively. $Cpred_4$ is 0 and $\theta^4 = +\infty$, because the reduced cost associated with the second objective function is now non-negative for any arc arriving at node 4. In iteration 2, label (3/7,-7,3) is extracted from the heap. In this case *i* is 3, (d_3^1, d_3^2) becomes (4, 0) and $pred_3$ is node 1. The reduced costs of arcs (3,2), (3,4), (1,3) and (2,3) are (5,-2), (1,-1), (0,0) and (-3,7), respectively. Therefore, the labels (1,-1,4) and (5/2,-2,2) associated with nodes 4 and 2 are inserted in the heap. In iteration 3, the label (1,-1,4) is extracted from the heap with *i* being 4 and, therefore, identifying the optimality interval $\left[\frac{1}{3}, 1\right]$ for *Cpath* = $1\rightarrow2\rightarrow4$. Once the distance and predecessor labels are updated, *Cpath* is $1\rightarrow3\rightarrow4$ with ($d1$,$d2$) = (4,2). Now, the reduced costs of the arcs (2,4) and (3,4) are (-1,1) and (0,0), respectively. In iteration 4, label (5/2,-2,2) is extracted from the heap with *i* being 2. In this case, the reduced costs of arc (2,4) become (4,-1) when the distance and predecessor labels are updated. Therefore, $\theta^4 = 4$ and the label (1,-1,4) is inserted in the heap. In iteration 5, this label is extracted from the heap and now we know that the optimality interval of *Cpath* $1\rightarrow3\rightarrow4$ is $[1, 4]$. Once the labels are updated, we obtain that the current *Cpath* is $1\rightarrow3\rightarrow2\rightarrow4$ and the heap becomes empty, since all arcs have a non-negative reduced cost for the second objective function. Therefore, the last extreme supported path is printed in line 21 of the algorithm. In this case, this path is *Cpath* = $1\rightarrow3\rightarrow2\rightarrow4$ with ($d1$,$d2$) = (8,1) which is optimal for the interval $[4, +\infty]$.

Table 1. Execution of the RLBSP algorithm on the graph of Figure 1.

iter	*Heap*	*i*	(d_i^1, d_i^2)				$(Cpred_i, pred_i)$			θ^i			*Cpath*; (*d*1,*d*2)
			1	2	3	4	2	3	4	2	3	4	
0	(1/3,-6,4), (3/7,-7,3)		(0,0)	(0,2)	(1,7)	(1,9)	(0,1)	(1,2)	(2,3)	∞	3/7	1/3	
1	(3/7,-7,3)	4	(0,0)	(0,2)	(1,7)	**(3,3)**	(0,1)	(1,2)	**(0,2)**	∞	3/7	∞	1→2→3→4; (1,9)
2	(1,-1,4), (5/2,-2,2)	3	(0,0)	(0,2)	**(4,0)**	(3,3)	(**3**,1)	**(0,1)**	(**3**,2)	5/2	∞	1	
3	(5/2,-2,2)	4	(0,0)	(0,2)	(4,0)	**(4,2)**	(3,1)	(0,1)	**(0,3)**	5/2	∞	∞	1→2→4; (3,3)
4	(1,-1,4),	2	(0,0)	**(5,0)**	(4,0)	(4,2)	**(0,3)**	(0,1)	(**2**,3)	∞	∞	4	
5		4	(0,0)	(5,0)	(4,0)	**(8,1)**	(0,3)	(0,1)	**(0,2)**	∞	∞	∞	1→3→4; (4,2)
													1→3→2→4; (8,1)

4.2 A one-to-all ratio-labeling algorithm keeping track of all extreme supported solutions.

To address the one-to-all BSP problem some modifications to the RLBSP algorithm proposed in the previous section are necessary which are outlined here. With the following exceptions, the notation is identical to that used in algorithm RLBSP: Storing a path associated with each supported extreme non-dominated point requires a set of labels *L* for each node $i \in V$ denoted L_i. Individual labels at node *i* are $L_i^1, L_i^2, \cdots, L_i^{N_i}$. Labels L_i^k with *k* varying in {1,..,N_i} for all $i \in V$ must be dynamically created (as the value of N_i is unknown before the end of the algorithm). In particular label L_i^k stores the information of the *k*th extreme supported point associated with the s-*i* BSP. For label L_i^k ={*j*, *r*, d_i^1, d_i^2 } *j* is the node predecessor of node *i* in the *k*th solution, *r* denotes the index of extreme supported point of the node *j* that allows to obtain the *k*th solution for node *i*, and d_i^1, d_i^2 are the distance labels of the node *i*. The values of *j* and *r* associated with the label allow identifying the *k*th path of the s-*i* BSP problem. Additionally, the algorithm uses the values of $lastratio_i$ for storing the last ratio of the extreme supported point corresponding to an *s*-*i* path, and N_i to store the number of extreme supported points obtained for node *i*.

Note that the RLBSP2 algorithm allows to identify the set of extreme supported non-dominated points of the one-to-all BSP problem in O(*N*(*m*+*n*log*n*)) time. To solve the one-to-all BSP problem a major modification in the proposed algorithm consists of deleting line 9. In this case, the algorithm requires O(*N*+*m*) space in the worst-case.

Note that the reduced cost of any arc $(i, j) \in A$ is computed as $\overline{c}_{ij}^{\,p}(T) = c_{ij}^p + L_i^{N_i}.dp - L_j^{N_j}.dp$ with *p*=1,2. The only differences between the RLBSP and RLBSP2 algorithms appear in lines 2 and 9-12, that is, the updating operation of labels *L*. Therefore, the RLBSP2 algorithm runs

in O(N(m+$n$log$n$)) time since accessing the labels L_i^k takes constant time. Clearly, the memory space used by the algorithm is determined by the labels L, that is, O(N + m) memory space.

Algorithm 2: *Ratio-Labeling BSP2* (RLBSP2) **Algorithm**;

(1) Let T^* be an optimal tree of $\operatorname{lex}\min_{p\in P}\left(c^1(p), c^2(p)\right)$ and store d and *pred* labels of T^*;

(2) $lastratio_i = 0$, $L_i^1 = (pred, 1, d_i^1, d_i^2)$ and N_i=1 for all $i \in V$ with $pred_i \neq 0$; Otherwise L_i = NULL and N_i = 0;

(3) Set $\theta_i = +\infty$; $Cpred_i = 0$; for all $i \in V - \{s\}$; *CreateHeap*(H);

(4) Compute $(\theta_i, \hat{c}_i^2, Cpred_i) = \operatorname{lex}\min_{j\in\Gamma_i^-}\left\{(-\bar{c}_{ji}^1 / \bar{c}_{ji}^2, \bar{c}_{ji}^2, j) : \bar{c}_{ji}^2 < 0\right\}$ $\forall i \in V - \{s\}$;

(5) **For** all $i \in V - \{s\}$ **do** **If** $(Cpred_i \neq 0)$ **then** *Insert*$\left(\left(\theta_i, \hat{c}_i^2, i\right), H\right)$; $\hat{c}_i^1 = \bar{c}_{Cpred_i i}^1$;

(6) **While** $(H \neq \varnothing)$ **do**

(7) $\left(\theta_i, \hat{c}_i^2, i\right)$ = *find-min*(H);

(8) *Delete-min*(H);

(9) **If** $(\theta_i > lastratio_i)$ **then**

(10) $N_i = N_i + 1$; $L_i^{N_i} = (Cpred_i, N_{Cpred_i}, L_i^{N_i-1}.d1 + \hat{c}_i^1, L_i^{N_i-1}.d2 + \hat{c}_i^2)$;

(11) **Else**

(12) $L_i^{N_i} = (Cpred_i, N_{Cpred_i}, L_i^{N_i}.d1, L_i^{N_i}.d2)$; $L_i^{N_i}.d1 = L_i^{N_i}.d1 + \hat{c}_i^1$; $L_i^{N_i}.d2 = L_i^{N_i}.d2 + \hat{c}_i^2$;

(13) $lastratio_i = \theta_i$; $\theta_i = +\infty$; $Cpred_i = 0$;

(14) Compute $(\theta_i, \hat{c}_i^2, Cpred_i) = \operatorname{lex}\min_{j\in\Gamma_i^-}\left\{(-\bar{c}_{ji}^1 / \bar{c}_{ji}^2, \bar{c}_{ji}^2, j) : \bar{c}_{ji}^2 < 0\right\}$;

(15) **If** $(Cpred_i \neq 0)$ **then** *Insert* $\left(\left(\theta_i, \hat{c}_i^2, i\right), H\right)$; $\hat{c}_i^1 = \bar{c}_{Cpred_i i}^1$;

(16) **For** all $j \in \Gamma_i^+$ **do**

(17) **If** $(\bar{c}_{ij}^2 < 0)$ **then**

(18) **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**

(19) **If** $(Cpred_i == 0)$ **Then** *Insert*$\left(\left(-\bar{c}_{ij}^1 / \bar{c}_{ij}^2, \bar{c}_{ij}^2, j\right), H\right)$;

(20) **Else** *decrease-key*$\left(\left(-\bar{c}_{ij}^1 / \bar{c}_{ij}^2, \bar{c}_{ij}^2, j\right), H\right)$;

(21) $\theta_j = -\bar{c}_{ij}^1 / \bar{c}_{ij}^2$; $\hat{c}_j = \bar{c}_{ij}$; $Cpred_j = i$;

We use the next *PrintPath* procedure to determine the kth supported extreme path from node s to node i with distances labels $L_i^k.d1$ and $L_i^k.d2$:

Procedure PrintPath(s, i, k, L);

(1) **If** $(s \neq i)$ **then**

(2) PrintPath(s, $L_i^k.j$, $L_i^k.r$, L);

(3) **Print** "i";

The computational effort of the recursive procedure *PrintPath* is O(*n*) time since any supported path is a simple path. Therefore, the process of printing / identifying all supported paths requires O(*nN*) time.

In the next section, we examine the performance of the presented RLBSP.

5. Computational Results

The computational experiments are split into two sections. We separately present results for one-to-one *s*-*t* BSP and one-to-all BSP.

5.1 One-to-one BSP problems

To investigate the computational performance of the proposed RLBSP algorithm, we compare its runtime with that of alternative algorithms capable of finding supported extreme solutions of BSP from [8]. Those algorithms are the parametric network simplex (PARA), i.e. the parametric simplex for biobjective linear programs implemented in a network simplex framework; and the dichotomic Phase 1 method that iteratively solves single-objective shortest path problems based on the weighted sum scalarization from [8]. To solve single-objective shortest path problems we adapt the heap (DIKH) and double bucket (DIKBD) shortest path algorithms from [20] as well as using a label correcting algorithm (LCOR) from [21]. We label the dichotomic algorithm with DIKH (DIKBD, LCOR) shortest path algorithm DDIKH (DDIKBD, DLCOR).

Computational experiments are conducted on a Laptop running Ubuntu 12.04 with Intel® Core™ i5 CPU M 560 @ 2.67GHz × 4. Algorithms are implemented in C, and compiled using the gcc compiler (version 4.6.3) with –O4 compile option.

We use large road networks from the DIMACS Shortest Path Implementation Challenge [22] for our computational experiments and grid networks created similar to those in [8], but of larger dimensions. Characteristics of the road network instances are shown in Table 2. It should be noted these road network instances are much larger than those considered in [8]. Arc costs represent distance and travel time.

Grid networks represent rectangular grids with arcs between each node and its immediate neighbors in the grid, arc costs are selected randomly between 1 and 10. Paths lead from one side of the grid to the other. The instances in [8] are small and do not allow to distinguish performance of algorithms that focus on supported efficient extreme solutions. We therefore generate larger problem instances with grid height and width as shown in Table 3.

Table 2: Characteristics of road network instances from [22].

Name	Number of nodes	Number of arcs
NY	264346	733846
BAY	321270	800172
COL	435666	1057066
FLA	1070376	2712798
NE	1524453	3897636
CAL	1890815	4657742
LKS	2758119	6885658

Table 3: Characteristics of grid instances.

Name	Height	Width	Number of nodes	Number of arcs
G1-G7	300	300,350,…,600	90002 to 180002	35940 to 718800
G8-G14	350	300,350,…,600	105002 to 210002	419400 to 838800
G15-G21	400	300,350,…,600	120002 to 240002	479400 to 958800
G22-G28	450	300,350,…,600	135002 to 270002	539400 to 1078800
G29-G35	500	300,350,…,600	150002 to 300002	599400 to 1198800
G36-G42	550	300,350,…,600	165002 to 330002	659400 to 1318800
G43-G49	600	300,350,…,600	180002 to 360002	719400 to 1438800

5.1.1 *Results for road networks*

We randomly choose 100 different origin-destination pairs as the performance of algorithms depends on how far origin and destination node are apart. All algorithms are tested on the same set of origin-destination pairs. It should be noted that the parametric network simplex is very slow, hence we only run it for a single origin-destination pair to give an idea of its excessive runtime (no result is reported for the largest network LKS).

Results are shown in Table 4 where average, minimum and maximum observed runtimes are given (best average runtimes are bold). From Table 4 we note that PARA clearly solves the problem very slowly, this is because of the large overhead of the network simplex, where each basic feasible solution corresponds to a tree in the network. Hence, many basis exchanges simply modify the tree, but do not affect the actual *s-t* shortest path making the approach highly inefficient in solving BSP. There is also significant overhead in maintaining the basic tree structure of the network simplex. We note the significant improvement of performance when comparing RLBSP and PARA.

Table 4: Computational results for one-to-one BSP on road networks: average, minimum and maximum, number of extreme supported solutions (N_t).

		DLCOR	DDIKBD	DDIKH	PARA*	RLBSP	N_t
NY	avg	1.23	5.29	**0.73**	1428.99	1.11	12.05
	min	0.10	0.00	0.01	-	0.83	1
	max	2.92	19.46	2.59	-	1.53	29
BAY	avg	6.50	7.22	**0.67**	1456.87	0.99	9.17
	min	0.98	0.01	0.01	-	0.68	1
	max	24.22	27.52	2.07	-	1.46	20
COL	avg	14.20	11.54	**1.07**	2176.52	1.68	11.41
	min	0.59	0.01	0.01	-	1.23	1
	max	275.81	51.45	4.98	-	2.76	33
FLA	avg	45.74	95.87	**4.15**	17844.62	5.03	15.01
	min	3.11	0.04	0.04	-	3.77	1
	max	243.05	266.78	13.06	-	7.52	36
NE	avg	24.46	106.00	**9.84**	40764.08	13.65	23.02
	min	3.07	0.09	0.11	-	9.42	3
	max	64.33	356.13	35.50	-	19.93	56
CAL	avg	272.07	229.65	**9.92**	50913.74	12.78	18.15
	min	16.22	0.06	0.06	-	8.24	1
	max	1306.47	746.11	27.72	-	18.79	36
LKS	avg	189.88	457.73	**31.01**	N/A	40.50	36.00
	min	21.26	0.20	0.24	-	27.98	4
	max	661.35	1542.28	106.42	-	62.58	83

* PARA results for one origin-destination pair

Despite the BAY network being slightly larger than the NY network, DDIKH and RLBSP perform slightly better, on average, for BAY than for NY which is likely due to the network structure leading to a lower number of extreme supported solutions which indicate the BAY problems may be easier to solve despite larger network size. This is similar for NE and CAL networks.

We further observe that the combination of the dichotomic approach with DIKH performs best. Comparing finally with RLBSP we note that the average performance of RLBSP is worse than that of the dichotomic approach, but the maximum runtime of RLBSP is always shorter than that of the other approaches. To understand this we further analyze the problems by the distance between origin and destination as shown in Figure 3 for the NY instance. This illustrates that the runtime of RLBSP is relatively stable and does not appear to depend on distance between origin and destination node. For DDIKH (and the other dichotomic algorithms), we see a much higher dependence of runtime and distance between origin and destination node. We observe similar behavior for the other road network instances. In Table 5 we present the results from Table 4 by shortest path length: results are now grouped in three

categories, according to closest origin-destination pairs, those with medium distance, and those with longest distance; each group contains a third of the test runs. Table 5 clearly demonstrates the strength of RLBSP for origin-destination pairs that are far apart. This makes the RLBSP algorithm particularly attractive in real-world routing applications because it has stable runtime performance and consistently performs better for problems that are more difficult.

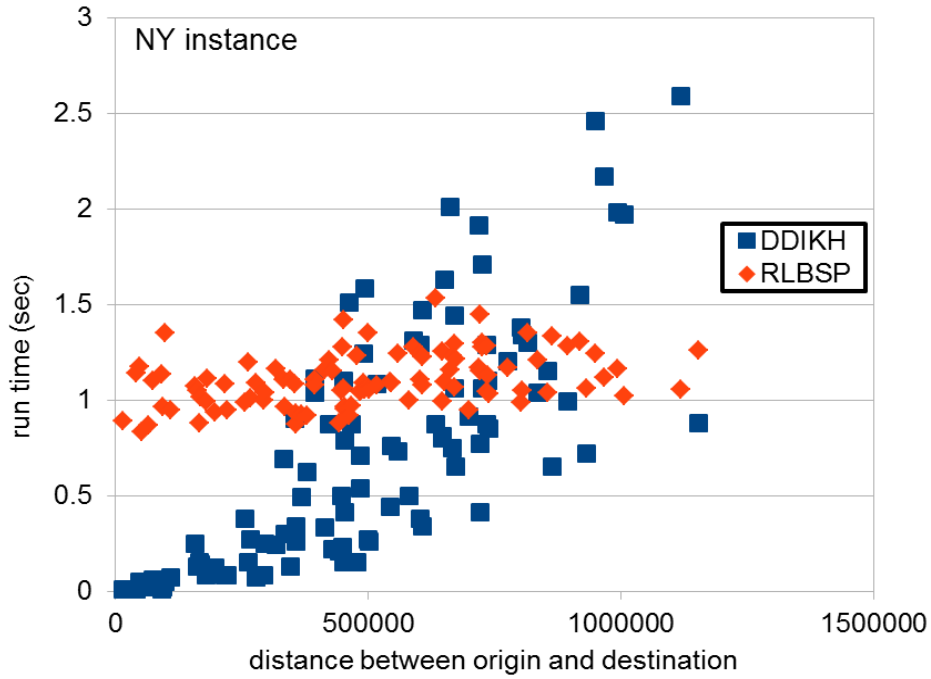

Figure 3. Run time of DDIKH and RLBSP for NY road network instance by distance between origin and destination node.

In Figure 4 we similarly observe that as the number of extreme efficient solutions found increases, the runtime of the dichotomic method DDIKH tends to increase. However, this is not the case for RLBSP, which, again, shows very stable runtimes. Analyzing results by low, medium and high number of efficient extreme solutions found, see Table 6, again shows that the performance of RLBSP is superior for the more difficult problems with a high number of efficient extreme solutions. In Table 6 the group with a low number of efficient solutions has up to approximately one third of the maximum number of efficient solutions encountered for this instance, the medium group between one third and two thirds of the efficient solutions, and the high group contains the rest. The ranges of efficient solutions in each group are also shown in Table 6 for each instance.

In conclusion, RLBSP is clearly an improvement compared to PARA. In fact, PARA runtimes were so excessive that we only showed results for a single origin-destination pair. While the dichotomic approach with heap-based Dijkstra (DDIKH) performs well, the

strength of RLBSP becomes apparent in reliably stable runtimes that are superior in particular for difficult problem instances with a large distance between origin and destination node and with many efficient solutions.

Table 5. Road network average runtimes: by closeness of origin and destination

	s-t distance group	DLCOR	DDIKBD	DDIKH	RLBSP
NY	close	0.69	0.98	**0.18**	1.03
	medium	1.36	5.22	**0.72**	1.12
	long	1.63	9.55	1.28	**1.17**
BAY	close	4.75	1.49	**0.17**	0.99
	medium	5.69	5.86	**0.56**	0.96
	long	8.99	14.10	1.26	**1.01**
COL	close	5.23	4.31	**0.41**	1.57
	medium	9.48	8.14	**0.79**	1.67
	long	27.49	21.87	1.98	**1.79**
FLA	close	34.43	33.96	**1.59**	4.87
	medium	43.12	99.52	**4.18**	4.87
	long	59.27	152.41	6.62	**5.35**
NE	close	12.73	20.83	**2.16**	13.69
	medium	22.35	97.18	**8.78**	13.07
	long	37.90	197.22	18.32	**14.18**
CAL	close	118.26	68.06	**2.91**	12.10
	medium	232.88	236.86	**10.38**	12.87
	long	440.20	465.15	19.86	**14.01**
LKS	close	84.34	85.54	**6.18**	39.35
	medium	175.92	396.96	**25.77**	38.57
	long	140.49	1542.28	106.42	**62.58**

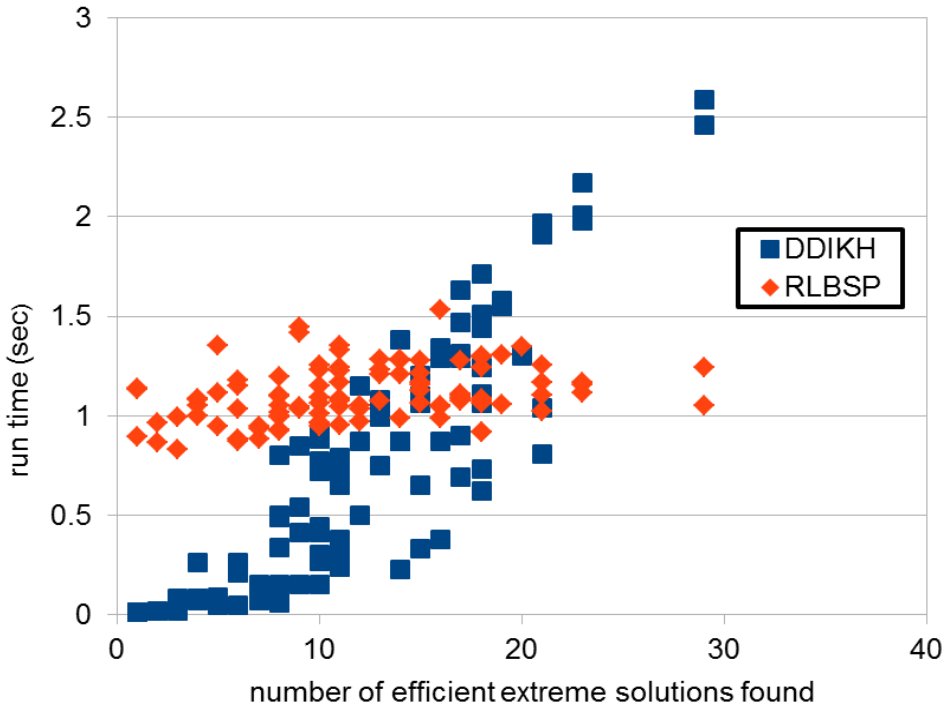

Figure 4. run time of DDIKH and RLBSP for NY road network instance by number of efficient extreme solutions found.

Table 6. road network average runtimes: by number of efficient solutions (N_t)

	N_t Group	N_t Range	DLCOR	DDIKBD	DDIKH	RLBSP
NY	low	1-10	0.69	1.62	**0.27**	1.05
	medium	11-20	1.53	7.05	**0.95**	1.15
	high	21-30	2.31	14.08	1.88	**1.14**
BAY	low	1-7	3.48	1.93	**0.22**	0.94
	medium	8-13	6.35	7.68	**0.73**	1.01
	high	14-20	12.73	16.46	1.42	**1.02**
COL	low	1-11	7.18	5.50	**0.55**	1.59
	medium	12-22	17.91	18.53	**1.66**	1.79
	high	23-33	143.40	47.00	4.48	**1.99**
FLA	low	1-12	31.27	33.21	**1.64**	4.81
	medium	13-24	51.32	124.88	5.19	**5.12**
	high	25-36	79.02	219.51	9.63	**5.54**
NE	low	1-18	12.28	33.94	**3.27**	12.92
	medium	19-36	26.41	119.24	**10.74**	13.49
	high	37-56	46.53	232.02	22.15	**15.72**
CAL	low	1-13	109.96	47.48	**2.26**	12.20
	medium	14-26	273.41	267.55	**11.51**	13.20
	high	27-41	496.36	578.39	24.57	**13.71**
LKS	low	1-27	86.81	94.36	**6.86**	40.50
	medium	28-54	201.65	529.78	**34.65**	38.18
	high	55-83	386.34	1079.69	75.02	**45.77**

5.1.2 Results for grid networks

Interestingly, results are very different for grid networks. We illustrate runtimes for the different instances in Figure 5. The proposed RLBSP approach consistently performs worse than the dichotomic approach combined with any shortest path algorithm. Here, the dichotomic approach performs best with a label correcting shortest path algorithm, which was different for road networks.

The problems have between 49 and 103 efficient solutions. The observed behavior may be explained by the fact that the grid networks have a very peculiar network structure, and arc cost functions that do not reflect real-world costs.

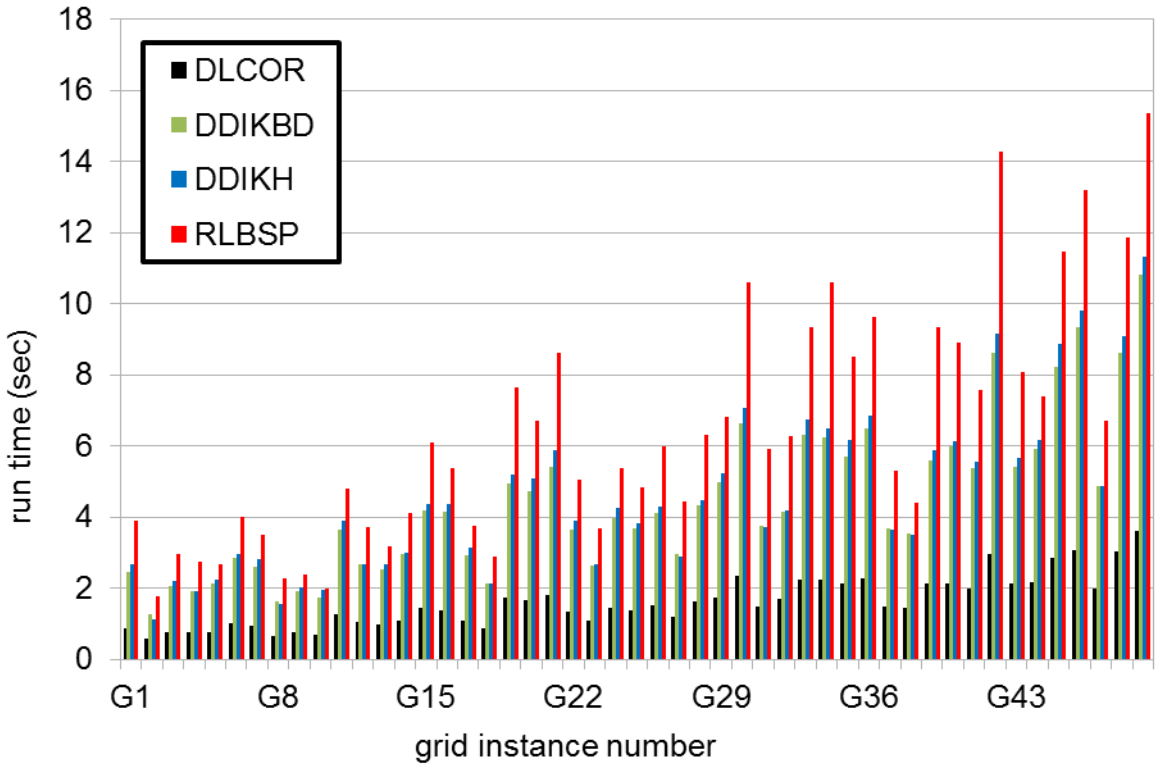

Figure 5. run time of DLCOR, DDIKBD, DDIKH and RLBSP for grid network instances.

5.2 One-to-all BSP problems

Encouraged by the stable runtime of RLBSP, which is easily adjusted to obtain paths from origin node *s* to all other nodes in the network, we now consider the one-to-all biobjective shortest path problem. We note that Guerriero and Musmanno [4], for example, also analyze the performance of standard multiobjective shortest path algorithms by solving one-to-all multiobjective shortest path problems.

For computational tests we can still compare PARA to RLBSP (in this section, we refer to RLBSP2 as RLBSP). However, we saw in section 5.1 that PARA is excessively slow and thus

omit further testing of PARA. The dichotomic approaches from section 5.2 cannot solve the one-to-all problem as they relied on *s*-*t* shortest path algorithms in each iteration. Instead, we suggest to adjust a biobjective label setting algorithm, see for example [8], to only retain supported labels in each iteration instead of retaining all non-dominated labels. This is a straightforward adjustment of a standard biobjective label setting algorithm, which can then be used to find extreme supported solutions of one-to-all BSP. We denote this modified biobjective label setting algorithm for finding supported extreme non-dominated path SLSET. As in [8] the algorithm is implemented using a binary heap in lexicographic order to manage extreme supported non-dominated labels.

5.2.1 Results for road networks

Table 7 shows average, minimum and maximum runtimes for the different road networks each based on 100 runs each with randomly chosen origin node. The number of extreme supported solutions found is also given, and it should be noted that this is the sum of the number of all *s*-*i* solutions found for all nodes $i \in V - \{s\}$. As expected, there is a large number of such solutions. Comparing Tables 4 and 7 shows that one-to-all RLBSP does take two to three times longer than one-to-one RLBSP on average. For the one-to-all problem (Table 7) we note the clearly superior performance of RLBSP compared to SLSET. As problem instances become larger both SLSET and RLBSP take longer to solve the problem with NY and BAY (NE and CAL) again being an exception - likely due to numbers of extreme supported solutions being similar or lower for BAY (CAL) which is a better indicator of problem difficulty than network size. Average runtimes of SLSET increase at a faster rate than those of RLBSP. Furthermore, the maximum runtime for RLBSP is always less than the minimum one of SLSET for any problem instance solved showing the superiority of RLBSP for the one-to-all BSP problem.

Table 7. Computational results for one-to-all BSP on road networks: average, minimum and maximum, number of extreme supported solutions (*N*).

		SLSET	RLBSP	*N*
NY	avg	16.45	**2.30**	3007848.45
	min	6.79	1.50	2136756
	max	37.03	3.23	4030225
BAY	avg	10.72	**1.98**	2891352.75
	min	3.74	1.31	2045515
	max	23.68	2.74	3816554
COL	avg	21.61	**3.73**	5189011.40
	min	6.64	2.47	3735277
	max	83.59	6.46	8025144

FLA	avg	119.17	**12.80**	15186195.22
	min	33.68	7.41	9669456
	max	571.92	28.01	29019702
NE	avg	520.16	**37.06**	35698609.17
	min	249.72	23.21	24576843
	max	1138.89	53.14	48569782
CAL	avg	516.72	**34.91**	35116514.50
	min	184.59	20.31	22340853
	max	1350.79	54.24	49076178
LKS	avg	**	131.97	104675211.78
	min	**	83.30	73077937.00
	max	**	225.36	160617349.00

**results omitted as too time consuming, partly due to extensive memory requirement

While results are not shown in Table 7, we have also conducted experiments with a standard biobjective label setting algorithm to solve the one-to-all problem, which identifies all efficient solutions (including non-supported ones). Such a standard algorithm already spends a very long time (746.33 seconds) on the first problem instance of NY, i.e. the smallest road network considered here. To compare SLSET takes 10.67 seconds to solve this problem instance, and RLBSP takes 2.30 seconds. This illustrates that finding all solutions, including non-supported ones, of one-to-all BSP is computationally challenging for large real-world problem instances. However, the problem can be solved for large instances using RLBSP if only supported solutions are required.

As shown in Figure 6 we can also analyze runtimes and number of extreme supported solutions (N) found which again confirms that while the runtime of RLBSP increases slightly with increasing number of extreme supported solutions found, the rate of increase seen is very low compared to that of the SLSET method.

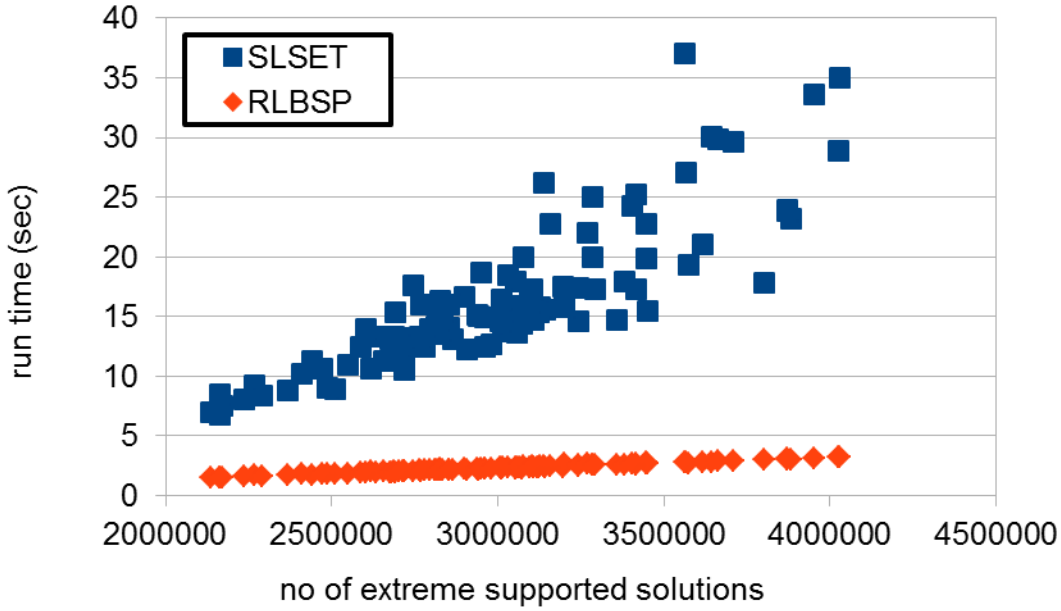

Figure 6. Run time of SLSET and RLBSP for NY road network by number of efficient extreme solutions found.

5.2.2 Results for grid networks

We repeat our computational tests of one-to-all BSP for grid networks. Other than one-to-one results for grid networks, we observe that for one-to-all BSP RLBSP consistently outperforms SLSET for all problem instances as shown in Figure 7.

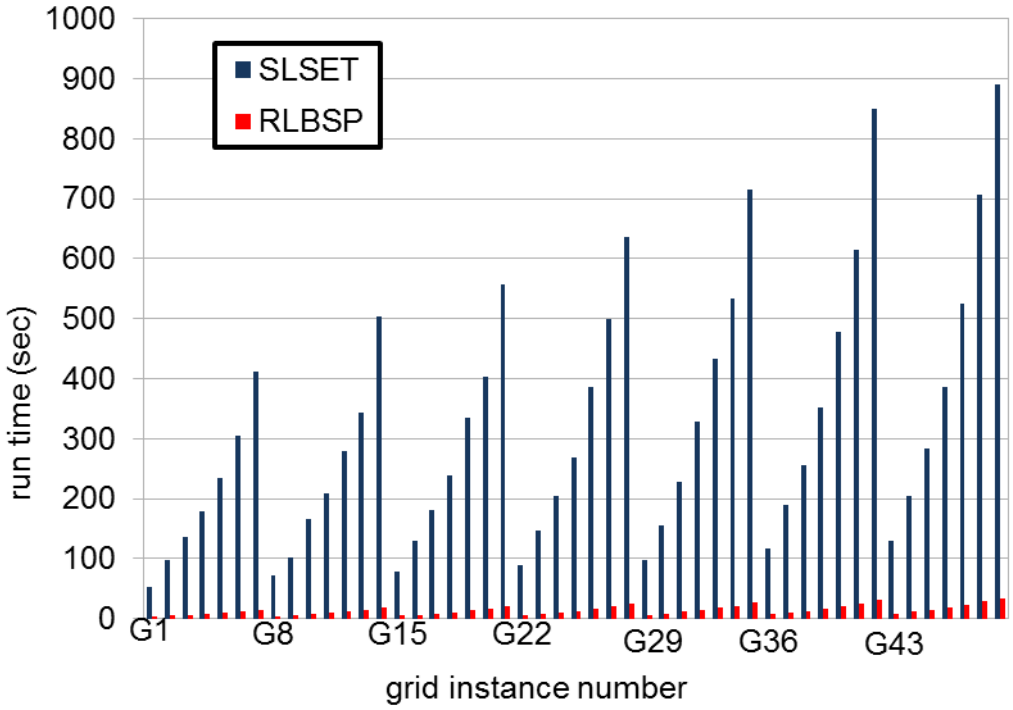

Figure 7. Run time of SLSET and RLBSP for grid network instances.

RLBSP is also less affected by increasing problem difficulty than SLSET: over all grid instances, the runtime of RLBSP is between 3.33 and 33.88 seconds whereas that of SLSET varies between 53.95 and 889.9 seconds.

From Figure 8 we observe that as the runtime of SLSET increases the runtime of RLBSP increases less, when expressed as a percentage of runtime of SLSET. RLBSP takes only between 3.7% and 6.4% of the runtime of SLSET, in particular for the more difficult problem instances RLBSP only takes around 4% (or under) the time of SLSET.

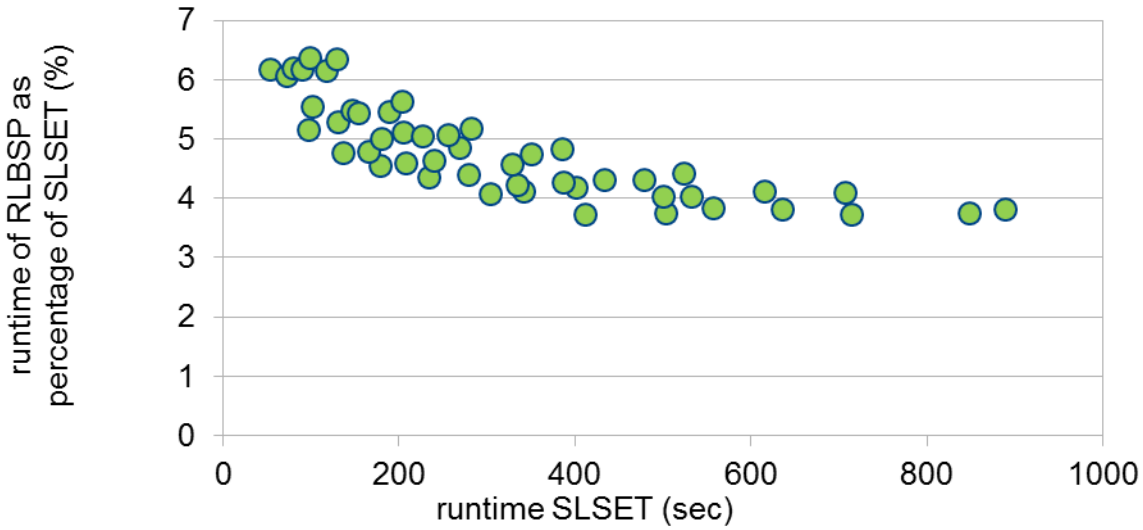

Figure 8. Run time of RLBSP as a percentage of runtime of SLSET for grid networks.

As for road networks we observe that RLBSP is particularly suited to solving the one-to-all BSP problem in grid networks.

6. Conclusions.

We propose a novel Dijkstra-like method to find all extreme supported solutions of BSP problems. A strength of the algorithm is a very effective adaptation of Dijkstra's algorithm based on the ideas of a parametric network simplex method for biobjective linear programs. The proposed Dijkstra-like algorithm determines the extreme supported solutions of the one-to-one and one-to-all BSP problem by computing, for each node, the ratio labels (the slope or trade-off of the two objective functions between two consecutive extreme supported solutions) instead of working with distance labels as the classical Dijkstra method. To do that, we prove that the sequence of the values of the ratios associated with each extreme supported solution is non-decreasing, that is, the ratio of any arc (node) never decreases below that of the current minimum ratio (see Section 3, expressions (1)-(7)). In the classical Dijkstra method, the distance labels of any nodes never decrease below that of the current minimum

distance label. The result is an efficient algorithm generalizing Dijkstra's method to find extreme supported efficient solutions to the BSP problem. Moreover, the space needed by our algorithm is minimal since it does not need to store the paths associated with each extreme supported solution. In its place, the algorithm stores predecessor labels that allow to build one extreme supported path for each extreme supported solution in O(n) time. This property is fundamental since to store hundreds of millions of paths in large networks containing millions of nodes in RAM is not possible on a conventional personal computer.

Our computational results distinguish one-to-one and one-to-all BSP problems. In the case of one-to-one BSP we find that a dichotomic approach that repeatedly solves parametric (weighted sum) shortest path problems is able to identify all supported extreme solutions of BSP more quickly than RLBSP, on average. For grid networks this is true for all test runs, whereas for road networks we observe that RLBSP shows superior performance for the more difficult problems where origin and destination are far apart, or where there are many efficient solutions. RLBSP is shown to be very reliable for all road network instances where its runtime remains very close to its average runtime no matter where the origin and destination nodes are, whereas the other tested algorithms are much more affected by choice of origin and destination nodes. For one-to-all BSP problems, the corresponding variant of RLBSP (RLBSP2 in section 4.1) clearly outperforms an implementation of a biobjective label setting algorithm that has been modified to only retain supported extreme efficient labels at all nodes. Here, RLBSP shows superior performance throughout the computational tests performed. Again, we note that RLBSP is affected by problem difficulty (as measured in numbers of extreme supported solutions) to a much smaller degree than the biobjective label setting algorithm we compare it to.

Our RLBSP algorithm could be beneficial particularly when solving large BSP problems where obtaining all supported solutions takes too much time, as long as obtaining the set of supported efficient solutions is considered sufficient. Another use would be the integration (as Phase 1 algorithm) in a two phase method [7] where it may perform better than alternative algorithms.

Acknowledgments

This research was partially supported by Marsden grant UOA1008 / 9075 362506 and by the European Union Seventh Framework Programme (FP7-PEOPLE-2009-IRSES) under

grant agreement n° 246647 and from the New Zealand Government under grant 649378 v2 as part of the OptALI project.

References

[1] Ahuja R, Magnanti T, Orlin JB. Network Flows: theory, algorithms and applications. Englewood Cliffs, New Jersey: Prentice-Hall; 1993.

[2] Hansen P. Bicriterion Path Problems. In Multiple Criteria Decision Making, Theory and Application. Proceedings of the 3rd International Conference, Hagen/Königswinter 1979, Springer Verlag, Berlin, 1980, 177, 109-127.

[3] Martins E. On a multicriteria shortest path problem. Eur Journal of Oper Res 1984;16:236-245.

[4] Guerriero F, Musmanno R. Label Correcting Methods to Solve Multicriteria Shortest Path Problems. J Optim Theor and Appl 2001; 111:589-613.

[5] Paixão J, Santos J. Labeling Methods for the General Case of the Multi-objective Shortest Path Problem -- A Computational Study Optimization Methods and Software, Computational Intelligence and Decision Making, Springer Netherlands; 2013.

[6] Martins E, Clímaco J. On the determination of the nondominated paths in multiobjective network problem. Method of Oper Res 1981;40:255-258.

[7] Mote J, Murthy I, Olson DL. A parametric approach to solving bicriterion shortest path problems. Eur Journal of Oper Res 1991;53:81-92.

[8] Raith A, Ehrgott M. A comparison of solution strategies for biobjective shortest path problems. Comput and Oper Res 2009;36:1299-1331.

[9] White D. The Set of Efficient Solutions for Multiple Objective Shortest Path Problems Comput and Oper Res 1982;9:101-107.

[10] Xie C, Waller ST. Optimal Routing with Multiple Objectives: Efficient Algorithm and Application to the Hazardous Materials Transportation Problem. Comput Aided Civil and Infrastruct Eng 2012; 27, 77.

[11] Xie C, Waller ST. Parametric search and problem decomposition for approximating Pareto-optimal paths. Transp Res Part B 2012;46:1043-1067.

[12] Henig M. The Shortest Path Problem with Two Objective Functions. Eur Journal of Oper Res 1985; 25:281-291

[13] Current JR, Revelle CS, Cohon JL. An interactive approach to identify the best compromise solution for two objective shortest path problems. Comput and Oper Res 1990;17:187-198.

[14] Coutinho-Rodrigues J, Clímaco J, Current J. An interactive bi-objective shortest path approach: searching for unsupported nondominated solutions. Comput and Oper Res 1999; 26:789-798.

[15] Steuer R. Multiple Criteria Optimization. Theory, Computation, and Application. New York: Wiley;1985.

[16] Isermann H. Proper efficiency and the linear vector maximum problem. Oper Res 1974;22:189-191.

[17] Dijkstra E. A note on two problems in connection with graphs. Numeric Math 1959; 1:269-271.

[18] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms. 3rd ed. Cambridge, Massachusetts:the MIT press;2009.

[19] Fredman ML, Tarjan RE. Fibonacci heaps and their uses in improved network optimization algorithms. J of the ACM 1987;34(3):596-615.

[20] Cherkassy B, Goldberg A, Radzik T. Shortest path algorithms: Theory and experimental evaluation. Math Program 1996;73:129-174.

[21] Carlyle WM, Wood RK. Near-shortest and k-shortest simple paths. Networks 2005; 46(2):98–109.

[22] DIMACS, 9th DIMACS Implementation Challenge - Shortest Path. http://www.dis.uniroma1.it/challenge9/download.shtml, last visited May 2013.