European Journal of Operational Research 242 (2015) 788–797

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

An exact method for the biobjective shortest path problem for large-scale road networks

Daniel Duque, Leonardo Lozano, Andrés L. Medaglia[*]

Centro para la Optimización y Probabilidad Aplicada (COPA), Departamento de Ingeniería Industrial, Universidad de los Andes, Bogotá, Colombia

A R T I C L E I N F O

Article history:
Received 18 January 2013
Accepted 1 November 2014
Available online 6 November 2014

Keywords:
Routing
Multiobjective combinatorial optimization (MOCO)
Biobjective shortest path
Multiobjective shortest path
Pulse algorithm

A B S T R A C T

The Biobjective Shortest Path Problem (BSP) is the problem of finding (one-to-one) paths from a start node to an end node, while simultaneously minimizing two (conflicting) objective functions. We present an exact recursive method based on implicit enumeration that aggressively prunes dominated solutions. Our approach compares favorably against a top-performer algorithm on two large testbeds from the literature and efficiently solves the BSP on large-scale networks with up to 1.2 million nodes and 2.8 million arcs. Additionally, we describe how the algorithm can be extended to handle more than two objectives and prove the concept on networks with up to 10 objectives.

1. Introduction

Consider a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \{v_1, \ldots, v_i, \ldots, v_n\}$ is the set of nodes and $\mathcal{A} = \{(i,j) | v_i \in \mathcal{N}, v_j \in \mathcal{N}\}$ is the set of arcs. For all arcs $(i,j) \in \mathcal{A}$ let there be two nonnegative weights denoted by c_{ij} and t_{ij}. Henceforth, and without loss of generality, we refer to c_{ij} and t_{ij} as the cost and time of traversing arc $(i,j) \in \mathcal{A}$, respectively. The *Biobjective Shortest Path Problem* (BSP) is the problem of finding paths $\mathcal{P}$ from the start node $v_s \in \mathcal{N}$ to the end node $v_e \in \mathcal{N}$ that minimize two different (often conflicting) objective functions. The BSP can be formally defined as follows:

$$\min \mathbf{z}(\mathbf{x}) = (c(\mathbf{x}), t(\mathbf{x})) \tag{1}$$

s.t.,

$$\mathbf{x} \in \mathcal{X} \tag{2}$$

where $\mathbf{x}$ is a path $\mathcal{P}$ represented by a vector of (binary) arc flows x_{ij}, $(i,j) \in \mathcal{A}$; $c(\mathbf{x}) \triangleq \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij}$ is the cost of path $\mathbf{x}$; $t(\mathbf{x}) \triangleq \sum_{(i,j)\in\mathcal{A}} t_{ij}x_{ij}$ is the time of path $\mathbf{x}$; and $\mathcal{X}$ is the set of all paths from v_s to v_e. In (1) we (simultaneously) minimize the cost and time components of the vector function $\mathbf{z}(\mathbf{x})$. Since the existence of a path that simultaneously minimizes both objectives in (1) cannot be guaranteed, alternatively we seek for a set of paths with an acceptable tradeoff between the two objectives. Henceforth, we use functions $c(\cdot)$ and $t(\cdot)$ to represent the cost and time for complete solutions (i.e., a path $\mathcal{P}$ from v_s to v_e) or partial solutions (i.e., a path $\mathcal{P}$ from v_s to a certain node v_i), respectively.

This work aims to expand the body of knowledge of exact methods for the BSP. Our work shares its intuition with the *pulse algorithm* proposed by Lozano and Medaglia (2013) for the Constrained Shortest Path Problem (CSP), which has been successfully used as an algorithmic block for the multi-activity shift scheduling problem (Restrepo, Lozano, & Medaglia, 2012) and has been extended to the weight constrained shortest path problem with replenishment (Bolívar, Lozano, & Medaglia, 2014). To emphasize the fact that this work is an extension of a flexible solution framework, we purposely keep the *pulse* name in this paper.

The rest of the paper is organized as follows. Section 2 introduces relevant concepts for the BSP. Section 3 presents a literature review of the main solution strategies for the BSP. Section 4 introduces the pulse algorithm and the intuition behind it. Section 5 presents the core components of the algorithm. Section 6 compares the proposed algorithm against a top-performer algorithm by Raith (2010). Finally, Section 7 concludes the paper and outlines future work.

2. Basic concepts

This section introduces relevant concepts related to the biobjective shortest path problem. Let us recall that $\mathcal{X}$ is the set of all paths $\mathbf{x}$ from v_s to v_e. The image of any solution $\mathbf{x} \in \mathcal{X}$ on the objective space $\mathcal{Z}$ is a

[*] Corresponding author: Universidad de los Andes, Cr 1E No. 19A-10, ML711, Bogotá, Colombia Tel.: +57 13394949, ext:2880. URL http://wwwprof.uniandes.edu.co/~amedagli

E-mail address: amedagli@uniandes.edu.co, andres.medaglia@gmail.com (A. L. Medaglia).

http://dx.doi.org/10.1016/j.ejor.2014.11.003
0377-2217/© 2014 Elsevier B.V. All rights reserved.

vector denoted by $\mathbf{z}(\mathbf{x}) = (c(\mathbf{x}), t(\mathbf{x})) \in \mathcal{Z}$, where $c(\mathbf{x})$ and $t(\mathbf{x})$ are the values of each objective function (cost and time, respectively).

In the BSP, we look for a set of solutions that cannot improve one component of the objective vector $\mathbf{z}(\mathbf{x})$ without deteriorating the other one. These solutions are referred to as *efficient* solutions and are formally defined as follows:

Definition 2.1. A solution $\mathbf{x} \in \mathcal{X}$ is efficient if there does not exist another solution $\mathbf{x}' \in \mathcal{X}$ such that $c(\mathbf{x}') < c(\mathbf{x})$ and $t(\mathbf{x}') \leq t(\mathbf{x})$ or $c(\mathbf{x}') \leq c(\mathbf{x})$ and $t(\mathbf{x}') < t(\mathbf{x})$.

Efficient solutions could be either *supported* or *non-supported*. Supported solutions correspond to the optimal solutions of the mono-objective shortest path problem defined by the following linear (convex) combination of the objectives:

$$\min_{\mathbf{x}\in\mathcal{X}} \lambda_c c(\mathbf{x}) + \lambda_t t(\mathbf{x}) \tag{3}$$

where $(\lambda_c, \lambda_t) \in \Lambda$ are the weights given to the cost and time within the weight set $\Lambda = \{(\lambda_c, \lambda_t) \in \Re^2 | \lambda_c \geq 0, \lambda_t \geq 0, \lambda_c + \lambda_t = 1\}$. On the other hand, efficient solutions which are non-supported cannot be obtained by solving a shortest path problem with a weighted sum of the objectives as in (3).

Similar to the efficiency concept defined over the solution space, any given solution has a corresponding vector (point) in the objective space $\mathcal{Z}$ that can be either *dominated* or *non-dominated*. The following set of definitions clearly states the concepts of dominance and their relation with efficiency.

Definition 2.2. The image $\mathbf{z}(\mathbf{x})$ of an efficient solution $\mathbf{x}$ is said to be a non-dominated vector. If the solution is not efficient, then its image is a dominated vector in the objective space. The set of all non-dominated vectors is denoted by $\mathcal{Z}_N$.

Definition 2.3. Let $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ be two solutions representing feasible paths. If $c(\mathbf{x}') < c(\mathbf{x})$ and $t(\mathbf{x}') \leq t(\mathbf{x})$ or $c(\mathbf{x}') \leq c(\mathbf{x})$ and $t(\mathbf{x}') < t(\mathbf{x})$, then $\mathbf{z}(\mathbf{x})$ is said to be dominated by $\mathbf{z}(\mathbf{x}')$, and it is denoted by $\mathbf{z}(\mathbf{x}') \preceq \mathbf{z}(\mathbf{x})$.

Definition 2.4. Let $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ be two solutions representing feasible, but different paths $(\mathbf{x}' \neq \mathbf{x})$. If $c(\mathbf{x}') = c(\mathbf{x})$ and $t(\mathbf{x}') = t(\mathbf{x})$, then $\mathbf{x}'$ and $\mathbf{x}$ are said to be alternative paths.

Definition 2.5. The set of all efficient solutions X_E is called the true efficient set. Given an algorithm, the set of efficient solutions discovered so far at any given iteration is called the online efficient set and it is denoted by $\hat{X}_E$. If the algorithm is exact, once it meets its stopping criterion, $\hat{X}_E$ coincides with X_E. We also make the distinction between the online set of non-dominated vectors $\hat{\mathcal{Z}}_N$ and the true non-dominated set $\mathcal{Z}_N$.

3. Literature review

The BSP arises naturally on multiple real applications. In telecommunications, Clímaco and Pascoal (2012) presented network routing problems where it is necessary to find paths that minimize the total number of links while simultaneously minimize the bandwidth consumption. Pallotino and Scutellà (1998) stated that transportation problems often offer a tradeoff between two or more objectives, e.g., minimizing the arrival time to a final destination and the cost of the path. Müller-Hannemann and Weihe (2006) reported a railway routing problem that faces a compromise between monetary cost and travel time. Ehrgott, Wang, Raith, and Van Houtte (2012) presented a biobjective cyclist route choice model in which bike routes are designed based on the travel time and a suitability weight associated with each arc. Erkut and Verter (1998) presented a real-world hazmat transportation application, where the conflicting objectives are the path risk (i.e., probability of an accident) and its associated cost. Aside from its direct application, the BSP arises as a subproblem of combinatorial optimization models (Skriver & Andersen, 2000). In all the cases cited above, it is necessary to find a set of solutions that takes into account more than one objective simultaneously, rather than to find a single solution that focuses on a single objective.

Aside from its practical relevance, the BSP is also a challenging problem that is NP-hard (Serafini, 1986). Even though several researchers have proposed different methods for solving the BSP, there are two major solution strategies for the BSP: dynamic programming (DP) and ranking.

In the DP category, there are label correcting and label setting methods. Hansen (1980) and Martins (1984) were among the first authors who proposed a labeling approach for the BSP. The label correcting method is a straightforward extension of the mono-objective version (cf. Bertsekas, 1998), but with several labels at each node (Raith & Ehrgott, 2009). To find the efficient set of solutions, each node stores labels that represent tentative efficient paths. At the beginning, only the start node v_s is labeled. All labels at each node are extended through all the outgoing arcs, setting new labels over target nodes. When the label set of a node changes, the node is marked for reconsideration. When a node is reconsidered, all the dominated labels are deleted and the rest are extended. When the reconsideration heap is empty the algorithm finishes. Skriver and Andersen (2000) presented a label correcting algorithm that employs a node selection criterion for the reconsideration heap. Other versions, as the one presented by Guerriero and Musmanno (2001), employ a label selection criterion for the reconsideration heap. The label setting method works in a similar fashion. These algorithms always employ a label selection criterion and the main difference with label correcting is that only the selected label is extended through all the arcs. Raith and Ehrgott (2009) implemented a label setting algorithm using a binary heap for the labels storage. In this algorithm, the smallest lexicographically ordered label is selected from all nodes to be extended at each iteration. The extended label is compared with the labels at the target node of each arc and dominated labels are deleted. Several speedup strategies for DP approaches have been developed recently. Raith (2010) proposed bounded label correcting and setting algorithms. These bounded versions use the labels at the end node for the dominance test at each node (in addition to the node's own labels). Iori, Martello, and Pretolani (2010) presented a label setting policy that treats labels according to an aggregate function calculated for each label. Demeyer, Goedgebeur, Audenaert, Pickavet, and Demeester (2013) used the same idea of a bounded labeling algorithm (Raith, 2010) in a unidirectional/bidirectional label setting algorithm. The bidirectional DP extends labels forward from the start node and backward from the end node. When forward and backward labels reach the same node, the labels are combined and added into the online non-dominated set. Both searches are aborted as soon as the there are no forward nor backward labels dominating solutions of the online non-dominated set (i.e., there are no promising labels to extend). Even though Demeyer et al. (2013) reported speedups of the bidirectional DP over the unidirectional version on the common testbed instances used by Raith (2010) and Demeyer et al. (2013), computational times are better for the labeling approach of Raith (2010). Müller-Hannemann and Schnee (2007) and Disser, Müller-Hannemann, and Schnee (2008) also proposed speedup techniques for DP algorithms that exploit particular characteristics of time-dependent networks used in railroad routing.

In the ranking category, the near shortest path (NSP) method finds all the paths within a certain deviation from the shortest path length found by solving the weighted sum problem associated with the BSP. Carlyle and Wood (2005) presented a method that, besides its remarkable performance for solving the near shortest path problem, it outperforms other specialized algorithms solving the *k*-shortest path problem. Raith and Ehrgott (2009) compared different solution strategies including label setting and label correcting approaches, the near shortest path method, and a two-phase method based on the approach of Ulungu and Teghem (1995). The two-phase method by Raith and Ehrgott (2009) starts with an initialization procedure with the best solution for each single objective. In phase one, the algorithm focuses on supported efficient solutions; while in phase two, it finds the missing non-supported efficient solutions.

Aside from these methods, Geisberger, Kobitzsch, and Sanders (2010) proposed an algorithm that solves several mono-objective shortest path problems to compute supported efficient solutions. Recent research has focused on speedup techniques and preprocessing steps to accelerate mono-objective shortest path algorithms over large networks. A comprehensive survey of these advances is presented by Bast et al. (2014).

The BSP can be seen as a special case of the Multiobjective Shortest Path Problem (MSP); nevertheless, the BSP is by far, the most studied problem among MSPs (Chinchuluun & Pardalos, 2007). Martins (1984) presented one of the first label setting algorithms for the MSP. Later on, Carraway, Morin, and Moskowitz (1990) proposed a generalized DP approach for the MSP that includes probabilities on the objectives (i.e., the probability of successfully traversing the arc). Guerriero and Musmanno (2001) presented a labeling method that outperforms the one by Martins (1984) on networks with two, three, and four objectives. More recently, Paixão and Santos (2013) considered two labeling techniques on a large set of test problems with 6, 8, and 10 objectives. Although most algorithms for the BSP are extensible in theory to the general case, just few of them have been tested with more than two objectives (Chinchuluun & Pardalos, 2007). Aside from the implementation and computational challenges, the exponential growth of the efficient set might be the most significant hurdle to extend most of the current state-of-the-art algorithms for the MSP. Such large efficient sets (with an exponentially large number of efficient solutions) might turn out simply impossible to handle from a practical standpoint. Under a large-efficient set scenario, some algorithms tackle the MSP by finding a high-quality approximation of the efficient set with less computation effort (than that of an exact method). Along this line, multiobjective evolutionary algorithms (MOEAs, cf. Coello, Lamont, & Veldhuizen, 2007) have proven to be a valuable source to solve a wide range of multiobjective combinatorial problems (cf. Coello & Lamont, 2004). For a survey on multiobjective combinatorial optimization, the reader is referred to Ehrgott and Gandibleaux (2003).

4. The pulse algorithm: Intuition and overview

The pulse algorithm gets its name from a very simple, yet insightful analogy. Given a network, the algorithm sends a pulse from the start node v_s to the end node v_e. This pulse travels through the entire network storing the *partial path* $\mathcal{P}$ (an ordered sequence of visited nodes) and its cumulative objective functions, $c(\mathcal{P})$ and $t(\mathcal{P})$. Every pulse that reaches the end node v_e, is a feasible solution that might be efficient. Once a pulse reaches the end node, it recursively backtracks to continue its propagation through the rest of the nodes in the search for more efficient paths from v_s to v_e. If the pulse is let free, this recursive algorithm resembles a complete enumeration of all possible paths, which guarantees that the efficient set is always found. However, the practical value of the algorithm relies on the fact that it stops the exploration of any partial path whenever there is enough information that shows that the path will not lead to an efficient solution. This look-ahead mechanism *prunes* aggressively vast regions of the solution space and ultimately accelerates the exploration of the network. Algorithm 1 presents a high-level pseudocode of the pulse algorithm. For the `initialization` procedure, we run a one-to-all mono-objective shortest path algorithm, but more details are deferred until Section 5.2.

To avoid a complete enumeration and to control the pulse propagation, we use a set of strategies that prunes pulses without cutting off any efficient solution. Every time that a pulse arrives to a node,

Algorithm 1 Pulse algorithm

```
Input: $\mathcal{G}$, directed graph; $v_s$, start node; $v_e$, end node.
Output: $X_E$, true efficient set
1: $\mathcal{P} \leftarrow \{\}$
2: $c(\mathcal{P}) \leftarrow 0$
3: $t(\mathcal{P}) \leftarrow 0$
4: initialization($\mathcal{G}$)                              ▷ see Section 5.2
5: pulse($v_s, c(\mathcal{P}), t(\mathcal{P}), \mathcal{P}$)
6: return $X_E$
```

the algorithm uses a battery of strategies to determine whether the partial path being explored should be propagated or not. Most importantly, pruning a partial path does not discard just one solution, but all the solutions that contain this partial path. Hence, aggressive and effective strategies transform an explicit enumeration into an efficient implicit enumeration. This idea shares the spirit of algorithms like branch-and-bound that also perform implicit enumerations. Note that the algorithm propagation follows a depth-first search truncated by particular pruning strategies.

In particular, for the BSP, we define four pruning strategies: *cycles*, *nadir point*, *efficient set*, and *label*. Algorithm 2 presents the `pulse` recursive function which receives as parameters the node being visited v_i, the cumulative cost $c(\mathcal{P})$, the cumulative time $t(\mathcal{P})$, and the partial path $\mathcal{P}$. Lines 1–4 of Algorithm 2 apply the different pruning strategies to the incoming pulse; if the pulse is not pruned, line 5 stores the current $c(\mathcal{P})$ and $t(\mathcal{P})$ while line 6 adds the node v_i to the partial path. In lines 7–11, the pulse propagates over all nodes $v_j \in \Gamma^+(v_i)$, where $\Gamma^+(v_i)$ is the set of outgoing neighbors of v_i, adding c_{ij} to the cumulative cost and t_{ij} to the cumulative time.

Algorithm 2 Pulse function

```
Input: $v_i$, current node; $c(\mathcal{P})$, cumulative cost; $t(\mathcal{P})$, cumulative time; $\mathcal{P}$, current path.
Output: void
1: if isAcyclic($v_i, \mathcal{P}$) then                                          ▷ see Section 5.1
2:     if ¬checkNadirPoint($v_i, c(\mathcal{P}), t(\mathcal{P})$) then              ▷ see Section 5.2
3:         if ¬checkEfficientSet($v_i, c(\mathcal{P}), t(\mathcal{P})$) then        ▷ see Section 5.3
4:             if ¬checkLabels($v_i, c(\mathcal{P}), t(\mathcal{P})$) then          ▷ see Section 5.4
5:                 store($c(\mathcal{P}), t(\mathcal{P})$)                         ▷ see Section 5.4
6:                 $\mathcal{P}' \leftarrow \mathcal{P} \cup \{v_i\}$
7:                 for $v_j \in \Gamma^+(v_i)$ do
8:                     $c(\mathcal{P}') \leftarrow c(\mathcal{P}) + c_{ij}$
9:                     $t(\mathcal{P}') \leftarrow t(\mathcal{P}) + t_{ij}$
10:                    pulse($v_j, c(\mathcal{P}'), t(\mathcal{P}'), \mathcal{P}'$)
11:                end for
12:            end if
13:        end if
14:    end if
15: end if
16: return void
```

Every time the pulse function is invoked at the end node v_e, a partial path $\mathcal{P}$ becomes a complete solution $\mathbf{x}$ and we update the online efficient set $\hat{X}_E$. Note that the information about $\hat{X}_E$ has a global scope and it is not an attribute of the traveling pulse within the recursion. Algorithm 3 presents the pulse function when it is invoked over the end node v_e. Since a new solution has been found, the algorithm verifies if the new solution is efficient and updates the online efficient set accordingly.

At this point it is important to highlight the main differences between the pulse algorithm and traditional labeling algorithms. First, note that while labeling algorithms usually follow a breadth-first search, the pulse algorithm follows a depth-first search, increasing the chances of finding complete efficient solutions faster. Additionally, due to the recursive nature of the algorithm there is no need to explicitly store the state space (e.g., labels in a priority queue) as done in labeling algorithms. Finally, even though the pulse algorithm uses a label pruning strategy, these labels are never extended, nor the

Algorithm 3 Pulse function for the end node

```
Input: $v_e$, end node; $c(\mathcal{P})$, cumulative cost; $t(\mathcal{P})$, cumulative time; $\mathcal{P}$,
    current path.
Input: void
1: if ¬checkEfficientSet($v_e$, $c(\mathcal{P})$, $t(\mathcal{P})$) then
2:     $\mathcal{P} \leftarrow \mathcal{P} \cup \{v_e\}$
3:     $\mathbf{x} \leftarrow$ mapPathToSolution($\mathcal{P}$)
4:     updateEfficientSet($\mathbf{x}$)                ▷ see Section 5.3
5: end if
6: return void
```

correctness of the algorithm depends on an exhaustive dominance test. In contrast to labeling approaches, this allows us to limit the number of labels stored at each node.

5. Pruning strategies

The performance of the pulse algorithm is tightly coupled to the pruning strategies defined for the problem at hand. This section explains the different strategies implemented for the BSP.

5.1. Pruning by cycles

Because all weights on the arcs are nonnegative, any efficient solution cannot contain cycles. To avoid cycles in a path, every time we invoke the pulse function at node v_i, the algorithm checks a function that indicates whether a node has been visited or not. If node v_i lies already on the partial path, $\mathcal{P}$ is pruned by cycles.

5.2. Pruning by nadir point

Let $\mathbf{x}_c^*$ and $\mathbf{x}_t^*$ be the optimal solutions for the mono-objective shortest path problem with the cost and time objectives, respectively. The images for the optimal solutions in the objective space are $\mathbf{z}(\mathbf{x}_c^*) = (\bar{T}, \underline{C})$ and $\mathbf{z}(\mathbf{x}_t^*) = (\underline{T}, \bar{C})$. The nadir point, denoted by $\mathbf{z}^N = (\bar{T}, \bar{C})$, is a vector in the objective space that establishes an upper bound for each objective. Under alternative optimal solutions for the mono-objective shortest path problem, $\bar{T}$ and $\bar{C}$ are the smallest values among all alternative solutions of $\mathbf{x}_c^*$ and $\mathbf{x}_t^*$, respectively. Fig. 1 shows the minimizer vectors $\mathbf{z}(\mathbf{x}_c^*)$ and $\mathbf{z}(\mathbf{x}_t^*)$, and $\mathbf{z}^N$ in the objective space. Note that the nadir point can also be seen as the anti-ideal point in the objective space, whereas $\mathbf{z}^*$ is the ideal point.

Consequently, for any solution $\mathbf{x}$ with $c(\mathbf{x}) > \bar{C}$ or $t(\mathbf{x}) > \bar{T}$, its image $\mathbf{z}(\mathbf{x})$ is dominated and $\mathbf{x}$ is not efficient. Fig. 1 shows how any point that falls in the dark gray region (i.e., $c(\mathbf{x}) > \bar{C}$ or $t(\mathbf{x}) > \bar{T}$) is either dominated by $\mathbf{z}(\mathbf{x}_c^*)$ or $\mathbf{z}(\mathbf{x}_t^*)$.

Based on this idea of the nadir point, the algorithm aims to prune as early as possible any pulse exceeding either $\bar{C}$ or $\bar{T}$. To do so, we calculate the minimum cost (regardless of time) and the minimum time (regardless of cost) it takes from any node v_i to reach the end node v_e. These lower bounds are obtained by reversing the network (i.e., reversing the direction of every arc), thus creating a new directed graph $\mathcal{G}' = (\mathcal{N}, \mathcal{A}')$ where $\mathcal{A}' = \{(j, i) | (i, j) \in \mathcal{A}\}$ and the weights $c'_{ji} = c_{ij}$ and $t'_{ji} = t_{ij}$ for $(j, i) \in \mathcal{A}'$. The starting node in $\mathcal{G}'$ is the end node of the original network, $v'_s = v_e$. We run a one-to-all shortest path algorithm for each objective from v'_s to all nodes. Using this procedure we obtain the minimum time $\underline{t}(v_i)$ and minimum cost $\underline{c}(v_i)$ from any node v_i to the end node v_e in the original network. Knowing these lower bounds, we determine the maximum values $\bar{t}(v_i)$ and $\bar{c}(v_i)$ that a partial path $\mathcal{P}$ to node v_i can show upon arrival as $\bar{t}(v_i) := \bar{T} - \underline{t}(v_i)$ and $\bar{c}(v_i) := \bar{C} - \underline{c}(v_i)$ for the time and cost objectives, respectively. If any partial path $\mathcal{P}$ to node v_i shows that $t(\mathcal{P}) > \bar{t}(v_i)$ or $c(\mathcal{P}) > \bar{c}(v_i)$, then partial path $\mathcal{P}$ can be safely pruned because it will surely exceed either one or both upper bounds defined by the nadir point before reaching (or at) the end node. The procedure for calculating $\bar{t}(v_i)$ and $\bar{c}(v_i)$ is done at the initialization phase of Algorithm 1 (see line 4). Algorithm 4 shows the nadir point verification procedure.

Algorithm 4 Pruning by nadir point

```
Input: $v_i$, current node; $c(\mathcal{P})$, cumulative cost; $t(\mathcal{P})$, cumulative time.
Output: boolean
1: prune ← false
2: if $c(\mathcal{P}) > \bar{c}(v_i)$ or $t(\mathcal{P}) > \bar{t}(v_i)$ then
3:     prune ← true
4: end if
5: return prune
```

It is worth noting that Tung and Chew (1992) proposed similar preprocessing procedures to strengthen dominance tests in DP approaches. Also Machuca, Mandow, Pérez de la Cruz, and Ruiz-Sepulveda (2012) explored the usage of exact and heuristic dual (lower) bounds to improve computational efficiency of label setting algorithms.

5.3. Pruning by efficient set

Given that the algorithm uses implicit enumeration, the true efficient set is unveiled only at the end of the algorithm's execution. However, the online efficient set obtained at intermediate stages of the algorithm is useful to prune partial paths whose images are dominated by solutions in this set. This idea has been explored by some authors in DP approaches. For instance, Tung and Chew (1992) used the labels stored at the end node (i.e., the online non-dominated set) and lower bounds to strengthen the dominance tests at each node. Müller-Hannemann and Schnee (2007) also used lower bounds to accelerate a DP algorithm in the context of railroad transportation. Raith (2010) and Demeyer et al. (2013) do not use lower bounds, but include the labels stored at the end node in the dominance tests for several variants of labeling algorithms.

Henceforth, we will refer to the efficient set as the online efficient set $\hat{X}_E$ that is updated as the algorithm finds new solutions. Fig. 2 shows a typical evolution of the efficient set as the algorithm progresses. After the initialization, the efficient set starts with two points (i.e., $\mathbf{z}(\mathbf{x}_c^*)$ and $\mathbf{z}(\mathbf{x}_t^*)$) and there is a large promising region, spanning from the ideal up to the nadir point, where efficient solutions can map their images (see Fig. 2(a)). As the algorithm evolves, new solutions in $\hat{X}_E$ enlarge the dominated region (white space). At any stage of the algorithm, a new solution is labeled efficient if its image is in the non-dominated (gray) region. We can safely prune a partial path, if its image—after completing the path with the optimistic bounds—falls outside of the promising (non-dominated) region.

Consider $\hat{X}_E$ at a given intermediate stage of the algorithm. We can determine whether a partial path has the potential to become an efficient solution (or not) using the same lower bounds calculated for

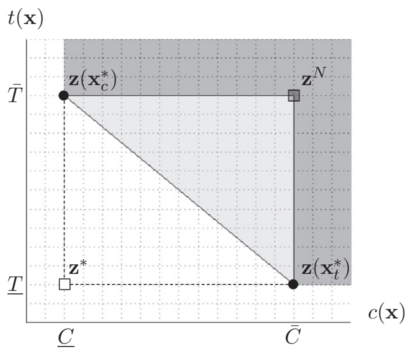


Fig. 1. Nadir point and lower and upper bounds.

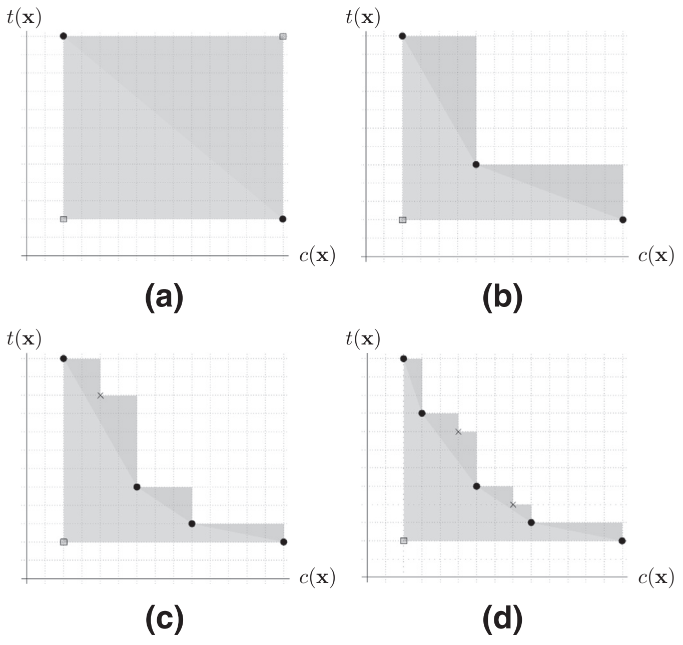


Fig. 2. Evolution of the non-dominated set as the algorithm progresses. Black points are associated to supported solutions and "×" to non-supported solutions. (a) Initial non-dominated set obtained after the initialization procedure. (b) Intermediate non-dominated set with three non-dominated points. (c) Intermediate non-dominated set with five non-dominated solutions. (d) True non-dominated set.

the previous strategy, namely, $\underline{t}(v_i)$ for time and $\underline{c}(v_i)$ for cost. Given a partial path $\mathcal{P}$ to node v_i, if there is a solution $\mathbf{x} \in \hat{X}_E$ such that $c(\mathcal{P}) + \underline{c}(v_i) \geq c(\mathbf{x})$ and $t(\mathcal{P}) + \underline{t}(v_i) \geq t(\mathbf{x})$, we can safely prune partial path $\mathcal{P}$, because even if it spends both the minimum cost and the minimum time to reach the end node, it will still be dominated by $\mathbf{x}$. Algorithm 5 shows the efficient set verification process.

Algorithm 5 Pruning by efficient set

```
Input: $v_i$, current node; $c(\mathcal{P})$, cumulative cost; $t(\mathcal{P})$, cumulative time.
Output: boolean
1: prune ← false
2: for $\mathbf{x} \in \hat{X}_E$ do
3:     if $c(\mathbf{x}) \leq c(\mathcal{P}) + \underline{c}(v_i)$ and $t(\mathbf{x}) \leq t(\mathcal{P}) + \underline{t}(v_i)$ then
4:         prune ← true
5:     end if
6: end for
7: return prune
```

To guarantee that we only store efficient solutions, whenever a partial path reaches the end node (becomes a solution $\mathbf{x}$) the efficient set $\hat{X}_E$ is updated as follows: if $\mathbf{z}(\mathbf{x})$ is dominated, solution $\mathbf{x}$ is discarded; if $\mathbf{z}(\mathbf{x})$ is non-dominated, $\mathbf{x}$ is added to the efficient set $\hat{X}_E$ and any solution dominated by $\mathbf{z}(\mathbf{x})$ is deleted from the set. In the case where alternative efficient solutions share the same objective vector, the algorithm only records one solution per objective vector. Note that as the algorithm approaches to the true efficient set, this pruning strategy becomes stronger. Algorithm 6 shows the procedure to update the efficient set once a new efficient solution is found.

It is important to highlight that the efficient set also contains the best solutions for each objective. Since the nadir point falls into this category, pruning by efficient set is a more general strategy that includes the test performed in the nadir point strategy. However, from a computational point of view, it is better to consider these two as separate strategies because the nadir point strategy is a simple check that requires considerably less computational effort. Note that each

Algorithm 6 Updating the efficient set

```
Input: $\mathbf{x}$, complete solution.
Output: void
1: for $\mathbf{x}' \in \hat{X}_E$ do
2:     if $c(\mathbf{x}) \leq c(\mathbf{x}')$ and $t(\mathbf{x}) \leq t(\mathbf{x}')$ then
3:         $\hat{X}_E \leftarrow \hat{X}_E \backslash \{\mathbf{x}'\}$
4:     end if
5: end for
6: $\hat{X}_E \leftarrow \hat{X}_E \cup \{\mathbf{x}\}$
7: return void
```

time that a solution is pruned by the nadir point strategy, the algorithm avoids the inspection of an often large list of efficient solutions required by the efficient set pruning strategy.

5.4. Pruning by labels

We can use labels to prove dominance relations over partial paths (subpaths from v_s to any node v_i) in the same way we prove dominance over complete solutions (see Definition 2.3). For each node v_i, we store a fixed number of labels. Each label saves a tuple of values associated with cost and time. The labels at node v_i are denoted by $\mathcal{L}(v_i) = \{(c_{il}, t_{il}) | l = 1, \ldots, Q\}$ where c_{il} and t_{il} are the cumulative cost and time for a partial path to v_i and Q denotes the number of labels at v_i. For an incoming pulse, the algorithm checks if the incoming partial path $\mathcal{P}$ is dominated or not; that is, if any label dominates $\mathbf{z}(\mathcal{P})$, the pulse is discarded by label pruning. Labels safely prune partial paths that may lead to inefficient paths, because any partial path within an efficient solution must be efficient too (cf. Proposition 9.4 and Corollary 9.5; Ehrgott, 2005). When a partial path $\mathcal{P}$ is not pruned by any of the strategies, the algorithm stores the cumulative cost $c(\mathcal{P})$ and cumulative time $t(\mathcal{P})$ in an empty slot of $\mathcal{L}(v_i)$ following a lexicographical order. If there are no empty slots, it overwrites any (randomly selected) label except those with the minimum cost and time. Algorithm 7 shows the label verification process.

Algorithm 7 Pruning by labels

```
Input: $v_i$, current node; $c(\mathcal{P})$, cumulative cost; $t(\mathcal{P})$, cumulative time.
Output: boolean
1: prune ← false
2: for $(c_{il}, t_{il}) \in \mathcal{L}(v_i)$ do
3:     if $c_{il} \leq c(\mathcal{P})$ and $t_{il} \leq t(\mathcal{P})$ then
4:         prune ← true
5:     end if
6: end for
7: return prune
```

6. Computational experiments on benchmark problems

We compared the computational performance of the pulse algorithm against the bounded label setting (bLSET) algorithm by Raith (2010) which is among the best performers for the BSP on real road networks. We implemented both algorithms in Java and compiled them with Eclipse SDK version 4.3 on a Windows 7 computer with a 2.6 GHz Intel Core *i*5 2540*M* (2 cores) CPU and 6 GB of RAM allocated to the memory heap size of the Java Virtual Machine. The initialization procedure was parallelized taking into account that we perform two independent shortest path executions, one for each objective. We coded the Dijkstra's algorithm using the double buckets implementation (DIKBD) presented by Cherkassky, Goldberg, and Radzik (1996) to solve the mono-objective shortest path problem. After fine tuning the algorithm, we fixed the maximum number of labels stored at each node to $Q = 20$. We sort the outgoing arcs of a node when we

Table 1
Computational results over real road networks DC, RI, and NJ from Raith and Ehrgott (2009).

Instance	Nodes	Arcs	$\|\mathcal{Z}_N\|$	bLSET time (s)	Pulse time (s)	Speedup
DC1	9559	39,377	2	0.11	$<$ **0.01**	27.25
DC2	9559	39,377	6	$<$ 0.01	$<$ 0.01	1.00
DC3	9559	39,377	3	$<$ 0.01	$<$ 0.01	1.00
DC4	9559	39,377	2	0.08	$<$ **0.01**	13.00
DC5	9559	39,377	1	$<$ 0.01	$<$ 0.01	1.00
DC6	9559	39,377	7	0.05	$<$ **0.01**	11.75
DC7	9559	39,377	2	0.02	**0.01**	2.00
DC8	9559	39,377	1	0.03	$<$ **0.01**	7.75
DC9	9559	39,377	6	0.05	$<$ **0.01**	11.75
Arithmetic mean						8.50
Geometric mean						4.50
RI1	53,658	192,084	3	0.05	$<$ **0.01**	11.75
RI2	53,658	192,084	15	0.83	**0.03**	25.88
RI3	53,658	192,084	2	0.20	**0.02**	8.83
RI4	53,658	192,084	17	0.41	**0.02**	18.45
RI5	53,658	192,084	16	0.60	**0.02**	29.75
RI6	53,658	192,084	3	2.30	**0.02**	127.61
RI7	53,658	192,084	3	1.34	**0.02**	74.67
RI8	53,658	192,084	4	0.66	**0.02**	34.58
RI9	53,658	192,084	22	0.22	**0.04**	5.92
Arithmetic mean						37.49
Geometric mean						24.22
NJ1	330,386	1,202,458	2	0.64	**0.13**	5.04
NJ2	330,386	1,202,458	6	0.25	**0.10**	2.58
NJ3	330,386	1,202,458	21	13.69	**0.10**	144.11
NJ4	330,386	1,202,458	5	0.97	**0.11**	9.15
NJ5	330,386	1,202,458	7	3.30	**0.11**	3.03
NJ6	330,386	1,202,458	12	14.11	**0.10**	146.98
NJ7	330,386	1,202,458	6	0.60	**0.10**	5.83
NJ8	330,386	1,202,458	13	5.30	**0.11**	48.62
NJ9	330,386	1,202,458	24	34.63	**0.23**	152.56
Arithmetic mean						57.54
Geometric mean						19.04

reach it for the first time based on the sum of both cost and time best bounds. All the execution times are reported in seconds and any run time that is less than 0.01 seconds appears in the tables as "$<$ 0.01". We established a maximum running time of one hour.

6.1. Large-scale real road networks

For this first experiment, we used the road networks from Washington (DC), Rhode Island (RI), and New Jersey (NJ) presented by Raith and Ehrgott (2009). For each road network, there is a set of nine instances which only difference is the randomly selected pair of start and end nodes. These are instances that range from 9559 nodes and 39,377 arcs to 330,386 nodes and 1,200,458 arcs. Table 1 presents the computational results for this set of networks. Column 1 shows the instance name; columns 2 and 3 present the number of nodes and arcs, respectively; column 4 shows the size of the non-dominated set; columns 5 and 6 show the computational time for our implementations of the bLSET and pulse algorithms (including the initialization time), respectively; and finally, column 7 shows the speedup calculated as the bLSET computational time over the pulse time. Times in bold highlight the faster algorithm.

Over these networks, the pulse algorithm outperforms the bLSET algorithm in 24 out of 27 instances, showing computational times consistently under 0.23 seconds in contrast to the bLSET algorithm that takes up to 34.6 seconds in NJ9, one of the largest instances in this testbed. For the DC instances, the pulse time remains under 0.005 seconds on seven out of the nine instances. In the larger RI instances, the pulse algorithm outperforms bLSET in all instances, achieving an average speedup of 37. Likewise, in the largest NJ instances the pulse algorithm outperforms bLSET in all instances, showing an average speedup of 57 times and achieving speedups of up to 150 times faster than the benchmark. Using the more conservative geometric mean (Bixby, 2002), the pulse is about 4, 24, and 19 times faster for DC, RI, and NJ instances, respectively. Note that it is on the larger instances where the pulse achieves the larger speedups.

6.2. Very large-scale real road networks

To better assess the scalability of the pulse algorithm, we conducted a second experiment on very large-scale read road networks that are significantly larger than those in the testbed by Raith and Ehrgott (2009). For this experiment, we use networks presented in the 9th DIMACS challenge (Demetrescu, Goldberg, & Johnson, 2006), namely, those representing New York City (NY), San Francisco Bay Area (BAY), Colorado (COL), Florida (FLA), and Northwest USA (NW). These networks range from 264,346 nodes and 733,846 arcs to 1,207,945 nodes and 2,840,208 arcs. The objectives considered for these networks correspond to the physical distance and transit time between nodes. Full datasets are available at http://www.dis.uniroma1.it/challenge9/download.shtml. For each of the five networks, we generated 30 random pairs of start and end nodes for a total of 150 instances. For each instance, we verified that either bLSET or pulse were able to solve it to optimality within 3600 seconds. Additionally, we categorized the 30 instances generated from each network into three clusters according to the size of the non-dominated set. To do so, we sorted the instances in increasing order and allocated the first 10 instances into the small cluster (S), the next 10 into the medium cluster (M), and the remaining 10 into the large cluster (L).

Table 2 presents the computational results for this experiment. Column 1 presents the cluster's name; column 2 presents the number of instances in the corresponding cluster; columns 3 and 4 show the number of nodes and arcs in the network, respectively; column 5 presents the average size of the non-dominated set; columns 6 and 7 show the average computational time and the number of instances solved to optimality using the bLSET algorithm; columns 8 and 9 show the same performance metrics for the pulse algorithm; column 10 shows the geometric mean of the individual speedups for the 30 instances of each network; and finally, column 11 shows the number of times that the pulse algorithm outperformed bLSET on each cluster. As for the initialization procedure, it is worth mentioning that it never spent more than 0.36 seconds, which for most of the cases is a negligible time.

Over these very-large instances, the pulse algorithm outperforms bLSET on 123 out of 150 instances, showing average speedups that range from 4.28 to 45.89 according to the geometric mean. Note that for the largest instances, the pulse algorithm achieves larger speedups and outperforms the bLSET algorithm in more cases. The pulse algorithm found the true efficient set on 144 out of 150 instances while bLSET found the true efficient set on 139 out of 150 instances. Moreover, on the six instances that the pulse algorithm missed the true efficient set, it was able to provide an approximation. On the other hand, bLSET was not able to solve 11 instances to optimality and it failed to provide any approximation of the true efficient set in 3600 seconds, meaning that no label ever reached the end node within the time limit. The quality of the approximation provided by the pulse algorithm is discussed in detail in Section 6.3.

6.3. Detailed assessment of the pulse algorithm

This section presents an introspective assessment of the pulse algorithm. First, we use a solution quality metric to analyze the evolution of the online efficient set into the true efficient set. Second, we present several metrics to evaluate the relative effectiveness of the pruning strategies in a subset of instances.

Table 2
Computational results over real road networks from the 9th DIMACS challenge.

Cluster	n	Nodes	Arcs	Average	bLSET		Pulse		Geometric mean	Pulse wins
				$\lvert\mathcal{Z}_N\rvert$	Average time (s)	Solved	Average time (s)	Solved	of speedups	
NY-S	10			34.10	62.39	10	**0.32**	10		9
NY-M	10	264,346	733, 846	147.40	301.16	10	**52.32**	10	7.25	10
NY-L	10			422.70	**881.26**	10	1367.66[a]	7		4
BAY-S	10			8.80	6.78	10	**0.16**	10		4
BAY-M	10	321,270	800, 172	49.90	55.24	10	**5.70**	10	4.28	10
BAY-L	10			171.80	317.43	10	**105.55**	10		8
COL-S	10			18.20	7.30	10	**0.20**	10		8
COL-M	10	435,666	1, 057, 066	87.10	**233.03**	10	381.94[a]	9	9.61	8
COL-L	10			328.40	865.76	10	**508.76**	10		7
FLA-S	10			14.70	330.12	10	**0.35**	10		9
FLA-M	10	1,070,376	2, 712, 798	94.10	566.15[a]	9	**347.91**	10	45.89	10
FLA-L	10			552.30	2627.43[a]	4	**888.59**[a]	9		8
NW-S	10			39.00	260.73	10	**1.99**	10		10
NW-M	10	1,207,945	2, 840, 208	124.20	1109.98[a]	8	**81.96**	10	21.70	9
NW-L	10			281.60	1443.66[a]	8	**438.54**[a]	9		9
						139/150		144/150		123/150

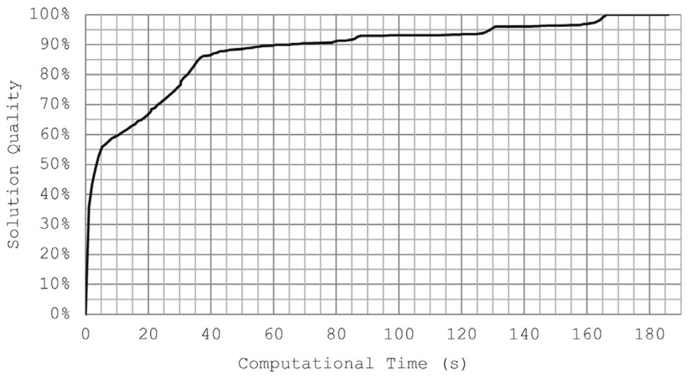
[a] Average time is calculated with a computational time of 3600 seconds for unsolved instances.

Our first experiment in this section, analyzes the evolution of the online efficient set at different points in time. Fig. 3 presents the evolution of the solution quality for instances BAY1 and NW7. The solution quality of an instance is based on the *dominated space* metric by Zitzler and Thiele (1998), which measures the area in the objective space dominated by the non-dominated vectors. Based on this metric, we define the solution quality as the ratio between the dominated space of the online non-dominated set found by the algorithm within the time threshold and the dominated space of the true non-dominated set. A solution quality of 100 percent means that the algorithm found the true non-dominated set, and a value below 100 percent represents the fraction of the true dominated space covered by the online non-dominated set, that is, a proxy of the quality of the approximation of the true non-dominated set.

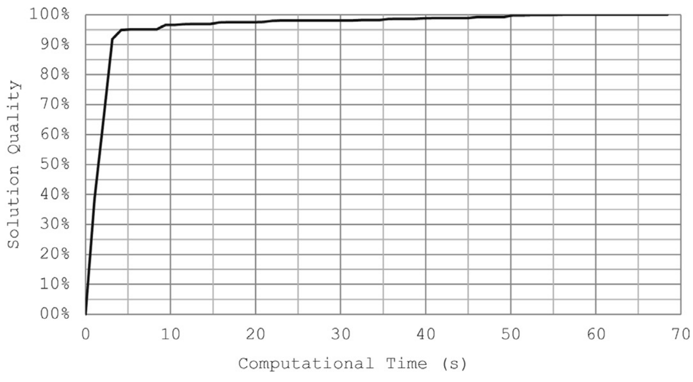
Fig. 3(a) shows the evolution of the solution quality for instance BAY1. For this instance, bLSET obtains the true efficient set in 65 seconds. Although the pulse algorithm takes a longer time to achieve the true non-dominated set, given that computational budget the pulse algorithm is able to find an approximate non-dominated set with a quality of roughly 90 percent. For instance NW7, the online efficient set converges to the true efficient set in 68 seconds as shown on Fig. 3(b). What is remarkable is that in just three seconds the quality of the online non-dominated set is well above 90 percent, and 40 seconds later, over 99 percent. Moreover, Fig. 4 shows the evolution of the online non-dominated set for instance NW7. Each subfigure compares the true non-dominated set (black dots) with different online non-dominated sets (gray dots) at different times. From these figures, it can be seen how the algorithm swiftly approximates the true non-dominated set. The bulk of the non-dominated set is shaped during the first 10 seconds (see Fig. 4(a)–(d)), while the rest of the time is spent on slight changes to fully converge to the true non-dominated set (see Fig. 4(e)–(h)). What this experiment tells us is that the pulse algorithm rapidly provides a good approximation of the non-dominated set and that if we impose a time budget, the algorithm can provide heuristically a high-quality approximation of the true non-dominated set.

We conducted yet another experiment to evaluate the relative effectiveness of the pruning strategies. In Table 3, for a subset of instances, we count the number of times each strategy prunes a partial path. Column 1 presents the name of the instance; and columns 2–4 show the relative effectiveness of each strategy as the fraction of paths pruned by the pulse algorithm.

For the first set of road networks by Raith and Ehrgott (2009), the nadir point strategy prunes on average 52.1 percent of the paths, while the label and efficient set pruning strategies account for 20.7 percent and 27.2 percent, respectively. On the second set of DIMACS instances, we observed a completely different behavior. In this set, the label strategy prunes on average 63 percent of the paths while the nadir point strategy just prunes 2.1 percent of the paths. We ascribe this behavior to the fact that on the first set of instances, the values of the arc attributes vary widely compared to those of the second set. With wide arc variations, the nadir point can effectively prune paths when the pulse propagates through arcs with large values for any given objective. Despite the fact that this result gives us a rough

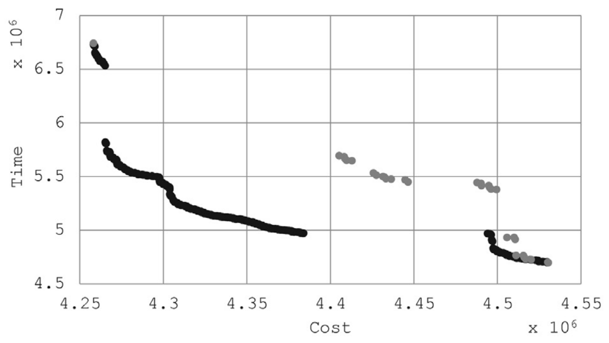


(a) BAY1

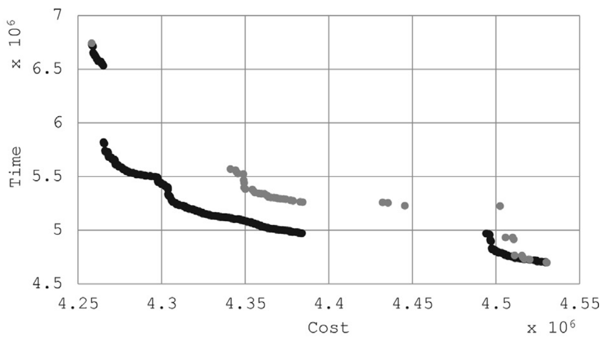


(b) NW7

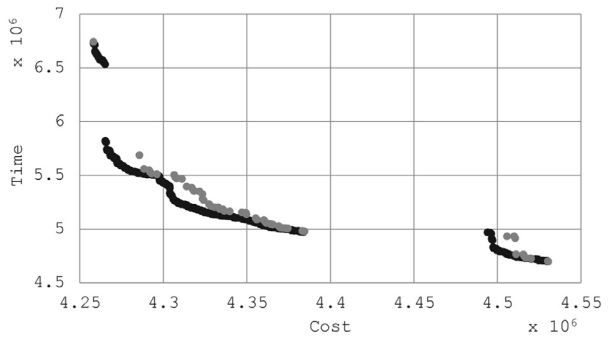
Fig. 3. Evolution of the solution quality (dominated space metric) over time.



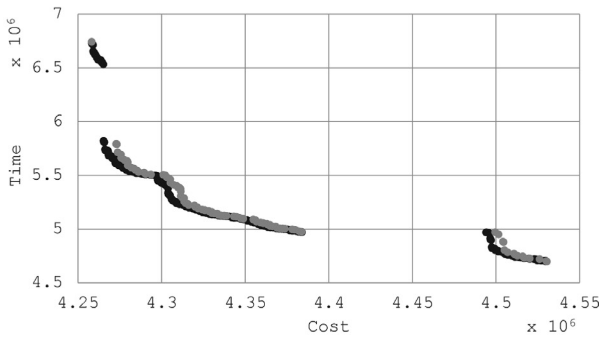
(a) 1 second



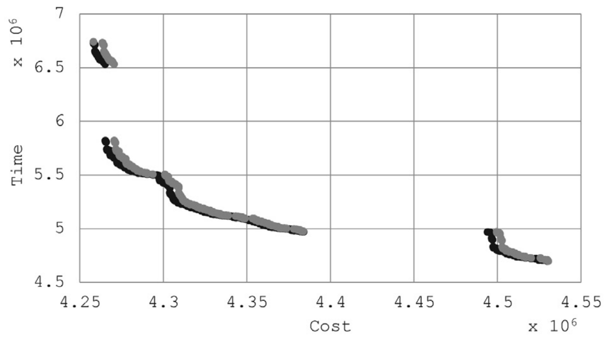
(b) 2 seconds



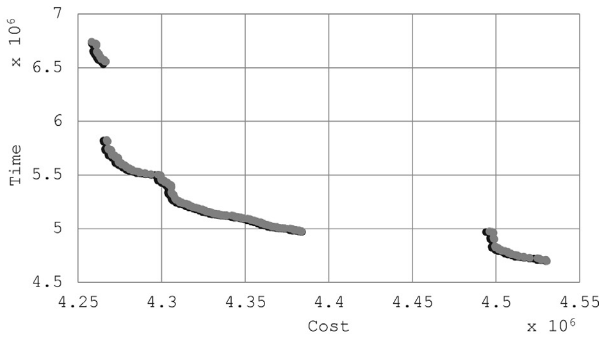
(c) 3 seconds



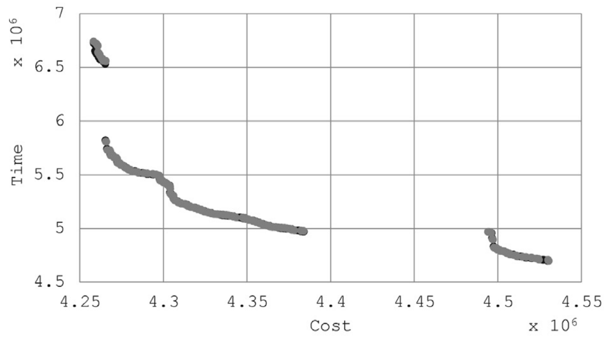
(d) 10 seconds



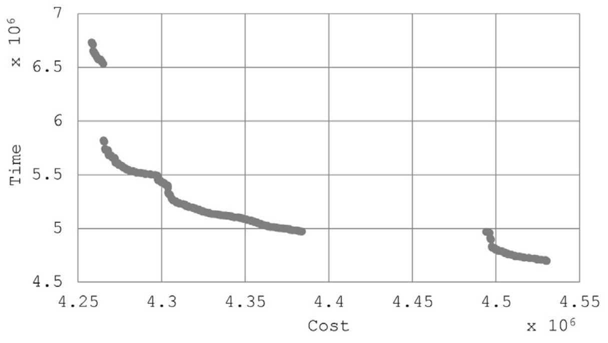
(e) 20 seconds



(f) 40 seconds



(g) 50 seconds



(h) 68 seconds

Fig. 4. Evolution of the non-dominated set over time. Black dots represent the true non-dominated set and gray dots represent the online non-dominated set.

idea of which strategies are pruning more than others, this relative effectiveness metric has its own drawbacks. Under the exploration scheme followed by the pulse algorithm, it is not the same to prune a partial path at an early stage than a partial path that is close to become a complete solution. A partial path that is pruned close to completion does not reduce the solution space as much as an early partial path that can be the origin of millions of paths. Nevertheless, the pruning count provides us with some valuable insight into which operations are considered the hot spots and should be subject to a more efficient coding.

Table 3
Relative effectiveness of the pruning strategies.

Instance	Pruning strategy		
	Label (in percent)	Nadir point (in percent)	Efficient set (in percent)
DC2	20.0	72.7	7.3
DC6	16.1	74.0	9.9
RI4	12.1	47.6	40.2
RI9	19.9	41.1	39.0
NJ3	20.2	38.8	41.0
NJ9	35.8	38.2	26.0
Arithmetic mean	20.7	52.1	27.2
NY1	39.1	4.0	56.9
NY4	66.9	1.1	32.0
BAY6	58.1	0.2	41.7
BAY9	64.0	3.6	32.4
COL5	68.7	1.4	29.9
COL9	63.7	3.6	32.7
FLA7	64.3	0.4	35.2
FLA9	68.1	0.6	31.3
NW7	76.6	0.3	23.1
NW10	60.9	5.7	33.4
Arithmetic mean	63.0	2.1	34.9

6.4. *Extending the algorithm for the Multiobjective Shortest Path (MSP) problem*

This section shows how the intuition of the proposed algorithm and its pruning strategies can be easily extended to handle more than two objectives. Formally, the MSP is defined as follows:

$$\min \mathbf{z}(\mathbf{x}) = (c_1(\mathbf{x}), \ldots, c_k(\mathbf{x}), \ldots, c_p(\mathbf{x})) \tag{4}$$

s.t.,

$$\mathbf{x} \in \mathcal{X} \tag{5}$$

where $\mathcal{X}$ is the set of all paths $\mathbf{x}$ from v_s to v_e and $c_k(\mathbf{x})$ is the kth objective.

First, we discuss how the cycle, label, and efficient set pruning strategies can be extended with minor modifications. The cycle pruning strategy does not depend on the number of objectives at all, so no change is needed. The dominance tests performed in the label pruning strategy are easily extended by enlarging the labels dimension on each node (one dimension per each objective). The efficient set pruning strategy can be extended running the initialization procedure for each objective. In this case, a vector of lower (dual) bounds $\underline{\mathbf{c}}(v_i) = (\underline{c}_1(v_i), \ldots, \underline{c}_k(v_i), \ldots, \underline{c}_p(v_i))$ is calculated for each node, where $\underline{c}_k(v_i)$ is the value of the shortest path (lower bound) for the kth objective from node v_i to the end node v_e.

In contrast, extending the nadir point pruning strategy is not straightforward, because calculating this point for three or more objectives is not as direct as in the biobjective case (Ehrgott, 2005). Instead, we calculate a *pseudo nadir point* $\mathbf{z}^{\hat{N}}$ and adapt the strategy for multiple objectives. Let $\mathbf{z}^{\hat{N}} = (z_1^{\hat{N}}, \ldots, z_k^{\hat{N}}, \ldots, z_p^{\hat{N}})$, where $z_k^{\hat{N}} = \max_{k'=1,\ldots,p}\{c_k(\mathbf{x}_{k'}^*)\}$ and $\mathbf{x}_k^*$ is the optimal solution of the mono-objective shortest path problem for the kth objective. This vector $\mathbf{z}^{\hat{N}}$ contains the worst values for each objective among all vectors $\mathbf{z}(\mathbf{x}_1^*), \ldots, \mathbf{z}(\mathbf{x}_p^*)$, henceforth called the *minimizer vectors* for each objective. For example, let us consider for $p = 3$ the minimizer vectors $\mathbf{z}(\mathbf{x}_1^*) = (4, 4, 7)$, $\mathbf{z}(\mathbf{x}_2^*) = (8, 2, 10)$, and $\mathbf{z}(\mathbf{x}_3^*) = (12, 7, 3)$. The pseudo nadir point is $\mathbf{z}^{\hat{N}} = (12, 7, 10)$. For the MSP, the pruning strategy discards a partial path from v_s to v_i, $\mathcal{P}_{s,i}$, if $c_k(\mathcal{P}_{s,i}) + \underline{c}_k(v_i) > z_k^{\hat{N}}$ for all $k = 1, \ldots, p$, where $c_k(\mathcal{P}_{s,i})$ is the cumulative value for the kth objective. Note that the main difference with the initial pruning strategy for the BSP is that the condition to prune must hold *for all* the objectives simultaneously. The following theorem demonstrates that this modification to the nadir pruning strategy does not cut off any solution of the true efficient set.

Table 4
Computational results for the multiobjective extension of the pulse algorithm on the testbed by Beasley and Christofides (1989).

Instance	Nodes	Arcs	Number of objectives					
			$p = 3$		$p = 5$		$p = 10$	
			Time (s)	$\lvert\mathcal{Z}_N\rvert$	Time (s)	$\lvert\mathcal{Z}_N\rvert$	Time (s)	$\lvert\mathcal{Z}_N\rvert$
rcsp5	100	990	0.00	4	0.00	4	0.00	16
rcsp7	100	999	0.13	42	0.12	394	3.78	4279
rcsp13	200	2080	0.00	23	0.00	65	0.03	223
rcsp15	200	1960	0.01	24	0.17	219	16.81	4177
rcsp21	500	4847	0.00	5	0.01	50	0.01	123
rcsp23	500	4868	0.04	55	0.83	617	101.61	9735

Theorem 6.1. *Let $z_k^{\hat{N}} = \max_{k'=1,\ldots,p}\{c_k(\mathbf{x}_{k'}^*)\}$ be the kth component of the pseudo nadir point and $\mathbf{z}(\mathbf{x}_1^*), \ldots, \mathbf{z}(\mathbf{x}_p^*)$ be the minimizer vectors for each objective. For a partial path $\mathcal{P}_{s,i}$ from node v_s to node v_i, if $c_k(\mathcal{P}_{s,i}) + \underline{c}_k(v_i) > z_k^{\hat{N}}$ holds for all objectives $k = 1, \ldots, p$, $\mathcal{P}_{s,i}$ can be safely pruned without cutting off any efficient solution.*

Proof. Let $\mathcal{P}_{s,e} = \mathcal{P}_{s,i} \cup \mathcal{P}_{i,e}$ be any complete path beginning with partial path $\mathcal{P}_{s,i}$. For any $\mathcal{P}_{i,e}$ it holds that $c_k(\mathcal{P}_{i,e}) \geq \underline{c}_k(v_i)$ for all $k = 1, \ldots, p$, because $\underline{c}_k(v_i)$ is a lower bound for the kth objective. Since $c_k(\mathcal{P}_{s,i}) + \underline{c}_k(v_i) > z_k^{\hat{N}}$ for all $k = 1, \ldots, p$, $c_k(\mathcal{P}_{s,e}) = c_k(\mathcal{P}_{s,i}) + c_k(\mathcal{P}_{i,e}) \geq c_k(\mathcal{P}_{s,i}) + \underline{c}_k(v_i) > z_k^{\hat{N}}$ for all $k = 1, \ldots, p$. Given that $z_k^{\hat{N}} \geq z_k(\mathbf{x}_{k'}^*)$ for all $k = 1, \ldots, p$ and for all $k' = 1, \ldots, p$, then $c_k(\mathcal{P}_{s,e}) > z_k^{\hat{N}} \geq z_k(\mathbf{x}_{k'}^*)$ for all $k = 1, \ldots, p$ and for all $k' = 1, \ldots, p$. Hence, the image in the objective space of any complete path $\mathcal{P}_{s,e}$ that begins with partial path $\mathcal{P}_{s,i}$ is dominated by all the minimizer vectors $\mathbf{z}(\mathbf{x}_1^*), \ldots, \mathbf{z}(\mathbf{x}_p^*)$ and partial path $\mathcal{P}_{s,i}$ can be discarded. □

As a proof of concept, we conducted an experiment with the multiobjective version of the pulse algorithm over a testbed for the Resource-Constrained Shortest Path Problem (RCSP) proposed by Beasley and Christofides (1989). We adapted those instances considering the distance and the resource consumption over the arcs as different objectives for the MSP. From the 24 instances available, we discarded those with only two objectives, and for each one of the remaining instances, we considered 3, 5, and 10 objectives. Table 4 summarizes the results for this set of instances for the MSP. Columns 1–3 show the instance name, followed by the number of nodes and arcs. Columns 4 and 5 show the computational time and the size of the non-dominated set for each instance with three objectives. Similarly, columns 6 and 7, and columns 8 and 9, present the information for the instances with 5 and 10 objectives, respectively.

By adapting the pulse algorithm for the MSP, we were able to solve instances with up to 10 objectives. For 3, 5, and 10 objectives, all instances were solved in less than 0.13, 0.83, and 101.61 seconds, respectively. As the number of objectives grows, so does the size of the true non-dominated set and the computational time required to find it. This behavior is expected since it is difficult to prove dominance among vectors when several objectives are considered. For instance, to prune a partial path by the label or efficient set strategy, it needs to be dominated in all objectives (as many as ten). With larger efficient sets, it might be impractical to handle such amount of solutions; however, with the pulse algorithm, a high quality approximation of the non-dominated set can be obtained with limited computational budget and a manageable size.

7. Conclusions and future work

We developed a new exact approach for the BSP that performs well on large-scale real road networks with up to 1,207,945 nodes and 2,840,208 arcs. The intuition of the algorithm is very easy to understand, backed on the simple idea of a pulse propagating through a network. To implement it, there are few considerations that must be taken into account, like the number of labels at each node and the criterion used to sort the outgoing arcs of each node. Although the algorithm is based on the idea of (implicit) enumeration, the proposed pruning strategies dramatically accelerate the exploration of the networks by exploring implicitly vast regions of the solution space. Our approach compares favorably against a top-performer algorithm by Raith (2010) in terms of the execution times. On the instance set from Raith and Ehrgott (2009), the proposed algorithm reaches speedups of up to 152 times and achieves an average speedup of 57 times over all instances. On the very-large scale instances from the DIMACS dataset, the pulse algorithm performed faster than bLSET on 123 out of 150 instances with a geometric mean of speedups of roughly 45 and 21 on the two largest problem sets. Aside from the good performance on the BSP experiments, we show the extensibility of the pulse algorithm to multiple objectives ($p \geq 3$) on instances from the literature with up to ten objectives.

Profiling our algorithm, we noticed that the key strategies are pruning by efficient set and nadir point for the proposed instances by Raith and Ehrgott (2009); and label and efficient set strategies for the DIMACS instances. As the algorithm progresses, the efficient set strategy is strengthened as new solutions complement the quick check made by the nadir point strategy and partial paths are more likely to be dominated.

With the straightforward intuition behind the pulse, it is simple to extend our algorithm to several network problems. Some of the future work includes studying acceleration strategies to improve the algorithm's performance, including non-additive cost functions, and further experimentation with multiple objectives. Finally, we would like to explore the stochastic variants of the BSP.

Acknowledgments

We would like to thank Professor Matthias Ehrgott at University of Auckland (New Zealand) for his generosity in sharing with us his testbed for the BSP. We would also like to thank to the anonymous referees for their insightful comments that helped us to improve this paper.

References

Bast, H., Delling, D., Goldberg, A. V., Müller-Hannemann, M., Pajor, T., Sanders, P., et al. (2014). Route planning in transportation networks. Technical Report MSR-TR-2014-4. Microsoft Research, Microsoft Corporation.

Beasley, J. E., & Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, *19*, 379–394.

Bertsekas, D. P. (1998). *Network optimization: Continuous and discrete models*. Belmont, MA: Athena Scientific.

Bixby, R. E. (2002). Solving real-world linear programs: A decade and more of progress. *Operations Research*, *50*(1), 3–15.

Bolívar, M. A., Lozano, L., & Medaglia, A. L. (2014). Acceleration strategies for the weight constrained shortest path problem with replenishment. *Optimization Letters*, *8*(8), 2155–2172.

Carlyle, W. M., & Wood, R. K. (2005). Near-shortest and K-shortest simple paths. *Networks*, *46*(2), 98–109.

Carraway, R. L., Morin, T. L., & Moskowitz, H. (1990). Generalized dynamic programming for multicriteria optimization. *European Journal of Operational Research*, *44*(1), 95–104.

Cherkassky, B. V., Goldberg, A. V., & Radzik, T. (1996). Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, *73*, 129–174.

Chinchuluun, A., & Pardalos, P. M. (2007). A survey of recent developments in multiobjective optimization. *Annals of Operations Research*, *154*(2), 29–50.

Clímaco, J. C. N., & Pascoal, M. M. B. (2012). Multicriteria path and tree problems: discussion on exact algorithms and applications. *International Transactions in Operational Research*, *19*, 63–98.

Coello, C. A., & Lamont, G. B. (2004). *Applications of multi-objective evolutionary algorithms*. Singapore: World Scientific.

Coello, C. A., Lamont, G. B., & Veldhuizen, D. A. V. (2007). *Evolutionary algorithms for solving multi-objective problems*. New York: Springer.

Demetrescu, C., Goldberg, A., & Johnson, D. (2006). 9th DIMACS implementation challenge—Shortest paths. www.dis.uniroma1.it/ challenge9/.

Demeyer, S., Goedgebeur, J., Audenaert, P., Pickavet, M., & Demeester, P. (2013). Speeding up Martins' algorithm for multiple objective shortest path problems. *4OR*, *11*(4), 323–348.

Disser, Y., Müller-Hannemann, M., & Schnee, M. (2008). Multi-criteria shortest paths in time-dependent train networks. In C. McGeoch (Ed.), *Experimental algorithms. Lecture Notes in Computer Science, Vol. 5038* (pp. 347–361). Berlin Heidelberg: Springer.

Ehrgott, M. (2005). *Multicriteria optimization* (2nd ed.). Springer.

Ehrgott, M., & Gandibleaux, X. (2003). Multiobjective combinatorial optimization—Theory, methodology, and applications. *International series in operations research & management science* (pp. 369–444). Dordrecht: Kluwer Academic Publishers.

Ehrgott, M., Wang, J. Y.T., Raith, A., & Van Houtte, C. (2012). A bi-objective cyclist route choice model. *Transportation Research Part A*, *46*, 652–663.

Erkut, E., & Verter, V. (1998). Modeling of transport risk for hazardous materials. *Operations Research*, *46*(5), 625–642.

Geisberger, R., Kobitzsch, M., & Sanders, P. (2010). Route planning with flexible objective functions. *Proceedings of the 12th workshop on algorithm engineering and experiments* (pp. 124–137)). Philadelphia, PA: SIAM.

Guerriero, F., & Musmanno, R. (2001). Label correcting methods to solve multicriteria shortest path problems. *Journal of Optimization Theory and Applications*, *111*(3), 589–613.

Hansen, P. (1980). Bicriterion path problems. In G. Fandel, & T. Gal (Eds.), *Multiple criteria decision making theory and application. Lecture Notes in Economics and Mathematical Systems, Vol. 177* (pp. 109–127). Berlin Heidelberg: Springer.

Iori, M., Martello, S., & Pretolani, D. (2010). An aggregate label setting policy for the multi-objective shortest path problem. *European Journal of Operational Research*, *207*, 1489–1496.

Lozano, L., & Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, *40*(1), 378–384.

Machuca, E., Mandow, L., Pérez de la Cruz, J. L., & Ruiz-Sepulveda, A. (2012). A comparison of heuristic best-first algorithms for bicriterion shortest path problems. *European Journal of Operational Research*, *217*(1), 44–53.

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

Müller-Hannemann, M., & Schnee, M. (2007). Finding all attractive train connections by multi-criteria Pareto search. In F. Geraets, L. Kroon, A. Schoebel, D. Wagner, & C. Zaroliagis (Eds.), *Algorithmic methods for railway optimization. Lecture Notes in Computer Science, Vol. 4359* (pp. 246–263). Berlin Heidelberg: Springer.

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

Paixão, J. M., & Santos, J. L. (2013). Labelling methods for the general case of the multi-objective shortest path problems—A computational study. In A. Madureira, C. Reis, & V. Marques (Eds.), *Intelligent Systems, Control and Automation: Science and Engineering. Computational intelligence and decision making* (pp. 489–502). The Netherlands: Springer.

Pallotino, S., & Scutellà, M. G. (1998). Shortest path algorithms in transportation models: classical and innovative aspects. In P. Marcotte, & S. Nguyen (Eds.), *Centre for Research on Transportation. Equilibrium and advanced transportation modeling* (pp. 245–281). Dordrecht: Kluwer Academic Publishers.

Raith, A. (2010). Speed–up of labelling algorithms for biobjective shortest path problems. In *Proceedings of the 45th annual conference of the ORSNZ, Auckland, New Zealand* (pp. 313–322). Operations Research Society of New Zealand.

Raith, A., & Ehrgott, M. (2009). A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, *36*, 1299–1331.

Restrepo, M. I., Lozano, L., & Medaglia, A. L. (2012). Constrained network-based column generation for the multi-activity shift scheduling problem. *International Journal of Production Economics*, *140*(1), 466–472.

Serafini, P. (1986). Some considerations about computational complexity for multi objective combinatorial problems. In J. Jahn, & W. Krabs (Eds.), *Recent advances and historical development of vector optimization. Lecture Notes in Economics and Mathematical Systems* (pp. 222–232). Berlin: Springer.

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

Tung, C. T., & Chew, K. L. (1992). A multicriteria Pareto-optimal path algorithm. *European Journal of Operational Research*, *62*, 203–209.

Ulungu, E. L., & Teghem, J. (1995). The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Science*, *20*, 149–165.

Zitzler, E., & Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms—A comparative case study. In A. Eiben, B. Thomas, M. Schoenauer, & H. Schwefel (Eds.), *Parallel problem solving from nature PPSN V. Lecture Notes in Computer Science, Vol. 1498* (pp. 292–301). Berlin, Heidelberg: Springer.