4OR manuscript No.
(will be inserted by the editor)

Speeding up Martins' algorithm for multiple objective shortest path problems

Sofie Demeyer · Jan Goedgebeur · Pieter Audenaert · Mario Pickavet · Piet Demeester

Received: date / Accepted: date

Abstract The latest transportation systems require the best routes in a large network with respect to multiple objectives simultaneously to be calculated in a very short time. The label setting algorithm of Martins efficiently finds this set of Pareto optimal paths, but sometimes tends to be slow, especially for large networks such as transportation networks. In this article we investigate a number of speedup measures, resulting in new algorithms. It is shown that the calculation time to find the Pareto optimal set can be reduced considerably. Moreover, it is mathematically proven that these algorithms still produce the Pareto optimal set of paths.

Keywords Multiobjective Shortest Path Problem · Labeling Algorithm · Stop Condition · Bidirectional Routing · Pareto Optimal Set

Mathematics Subject Classification (2000) 90C29 · 05C85 · 05C38 · 68Q25 · 90B06

1 Introduction

This paper presents two adaptations to a label setting algorithm, which was presented by Martins in 1984 [24], to solve the multiple objective shortest path problem. This is to date still a reference work to solve the multiple objective shortest path problem making use of graph algorithms. This section starts with a description of the problem tackled in this algorithm. Then an overview is given of what can be found in literature about multiple objective shortest path problems and their solutions. Finally, the contributions and objectives of this research are explained.

Dept. of Information Technology (INTEC), Ghent University - IBBT

Gaston Crommenlaan 8/201, B-9050 Ghent
Tel.: +32-9-331 49 77
Fax: +32-9-331 48 99
E-mail: sofie.demeyer@intec.ugent.be

1.1 Problem description

In this section, we present one of the possible applications of the research presented in this article, namely routing in transportation networks.

With the evolvement of GPS systems, routing in transportation networks gains more attention. One of the main characteristics of these transportation networks is that they tend to be large. For example the TIGER/Line network, covering the USA, consists of approximately 24 million nodes and 29 million links and the PTV Europe network contains approximately 19 million nodes and 23 million links [1]. In most applications, a preprocessing phase reduces the size of these networks, but even then, they are still extensive. On the other hand, these networks tend to be sparse with an average node degree between 2 and 3.

Moreover, in recent years, multimodal transportation (i.e. using multiple modes of transportation) becomes a valuable alternative to avoid road traffic jams. Multimodal networks are often represented by a layered network model in which each modus has its own transportation network and trans-shipments between the modes are modeled by trans-shipment links connecting nodes from different modes of transportation [9]. This results in even more extensive networks than the unimodal ones.

At the same time, logistic applications need to find the best route as soon as possible while taking into account multiple objectives. The transport needs to be both as fast, as cheap, as ecological, etc. as possible. The problem tackled in this article is how to calculate in a very short time the multiple objective shortest paths between an origin and a destination in a transportation network. As there are multiple objectives that need to be optimized, there is no single optimal path, but a set of paths that is called Pareto optimal. A path is called Pareto optimal if no objective can be ameliorated without deteriorating another objective (see section 2.1 for a more formal definition).

For this, we will further improve the MOSP (multiple objective shortest path) algorithm of Martins, which has proven to be an efficient way to find the complete Pareto optimal set of paths [12], especially for lower density networks such as transportation networks. Unfortunately, this algorithm, in its original form, has to examine the whole network to find the Pareto optimal set of routes. This is quite time consuming for large networks, such as (multimodal) transportation networks. Therefore, we will attempt to limit the search area of the algorithm, namely by formulating a stop condition to interrupt the search process and by searching the network bidirectionally (i.e. simultaneously from the origin and the destination).

In this article we assume that all objectives need to be minimized and all concepts are defined according to this assumption. Moreover, it should be noted that the research presented here assumes that all costs are non-negative and that there are no cycles with cost 0.

1.2 Literature overview

Multiple objective shortest path problems, which are known to be NP-hard [39], started to gain attention in the 70s and the beginning of the 80s with the works of

Vincke [43] and Hansen [21], in which they focused on the case with two objectives, the bi-objective shortest path problem. Since then, several exact methods, for both the general multi-objective and the bi-objective shortest path problem, have been developed to find all Pareto optimal paths between two nodes in a network.

The most straight-forward (and easiest) way to deal with multiple objectives is making use of single objective shortest path techniques with a weighted objective function. Unfortunately, this method only finds a subset of all Pareto-optimal paths. If we present all Pareto-optimal paths in a decision space, the paths found by this algorithm are situated on the convex hull. The other Pareto-optimal paths are situated in the so-called duality gaps. References [13] and [37] provide a more detailed discussion on this subject.

Exact methods to find the set of Pareto-optimal paths for the shortest path problem with two objectives have been thoroughly studied. As we focus in this article on the general multi-objective shortest path algorithm, we would just like to refer to the works of Skriver [40] and Raith and Ehrgott [37] for an overview of most of the bi-objective shortest path algorithms.

In [7], Clímaco and Pascoal give an overview to solve the multi-objective shortest path problems. According to the taxonomy presented in their article, the research presented in this paper can be categorized in the class of multicriteria path problems with additive metrics and a-posteriori aggregation of preferences method. In this class, two major categories can be distinguished: ranking techniques and labeling techniques.

Ranking techniques, like for example [6] and [2], use a ranking method for listing the paths in a decreasing order according to one of the objectives. Subsequently, from this list of paths a set of non-dominated paths is determined. Three groups of path ranking methods can be determined: deletion algorithms ([25], [22], [29]) in which at the start of each iteration the path found in the previous iteration is eliminated from the network, labeling algorithms ([28], [26], [34]) in which nodes may contain K labels (for the K paths) and that resemble the K shortest path algorithms, and deviation algorithms ([23], [14], [27]) in which a new path is considered to be a deviation from the previous path.

Labeling techniques are well known to solve the single objective shortest path problem ([10], [3]). Similar to the single objective case, two categories can be distinguished: label-setting algorithms and label-correcting algorithms. An exact label-setting algorithm for the bi-objective shortest path problem was proposed by Hansen [21]. This work was then generalized by Martins for the multiple objective shortest path problem [24]. To this date, this article is still a reference work for the multi-objective shortest path algorithms.Next to label-setting algorithms, a number of label-correcting algorithms were presented. Similarly, this research started with bicriteria shortest path problems ([43], [5], [41]) and was then generalized for multiple criteria [38]. Guerriero and Musmanno studied the impact of label selection versus node selection in label-correcting algorithms [20]. For a detailed theoretical study of the properties of label correcting and label setting algorithm to solve the multi-objective shortest path problem, we would like to refer to [30].

Most of the research described above assumes that all objectives are additive, but also other objective types can be used. One of the most popular is a combination of both additive objectives and objectives of the bottleneck type (i.e. minmax, maxmin).

Gandibleux et al. [16] generalized the algorithm of Martins to the case in which one of the objectives is a bottleneck. In [8] and [35] an algorithm is presented to solve the tricriteria shortest path problem in which at least two objectives are of the bottleneck type. A twofold extension to this latter research has been proposed by Bornstein et al. [4]. As we will focus on additive objectives in this research, we will not elaborate further on this.

Next to these exact methods, a number of well known heuristics have been applied to the multi-objective shortest path problem. In [32], for example, evolutionary algorithms are used to find a subset of the Pareto optimal paths.

A complete overview of multiple objective shortest path algorithms, both for bicriteria and multicriteria problems can be found in [17]. In this article we will focus on the exact label setting algorithm of Martins [24] with additive objectives, and propose a number of adaptations to speed up the calculation in the case where we want to find all non-dominated paths between a single origin node and a single destination node. In the following section more details are given on the objectives and the contributions that are made in this research.

1.3 Contributions and objectives

Our research focuses on exact and fast algorithms to solve multiple objective shortest path problems with n objectives, which are all sum problems. This problem can be described mathematically as

$$min_{p \in P_{s,t}}(z_1(p), z_2(p), ..., z_n(p))$$

in which $P_{s,t}$ represents the set of all paths between a node s and a node t while $z_i(p)$ is the cost of path p with respect to objective i. While some of the algorithms presented in the literature are especially for the bicriterion problem ([40], [37]), we opted not to limit the number of objectives (as for example in [20] and [38]). Moreover, we want to speed up the calculations without losing the guarantee of finding all Pareto optimal paths, in contrast with for example the heuristic search algorithm of [32].

This article focuses on two exact speedup techniques, namely a well-defined stop condition and searching the network bidirectionally. It is shown that the two proposed improvements achieve a considerable speedup compared to other exact graph algorithms, while at the same time maintaining the optimality of the solution.

In the first improvement a speedup is achieved by stopping the search as soon as all Pareto optimal paths between the origin and the destination are found, instead of searching the whole network. The second one searches the network from both the origin and the destination simultaneously, resulting in a speedup factor of more than 2 (where the bidirectional algorithm of Dijkstra achieves a speedup factor of around 2). This is clarified experimentally.

2 Unidirectional multiple objective shortest path algorithm

In order to better understand the remainder of the article, the unidirectional algorithm from which we started, is presented briefly in this section, together with a well-

defined stop condition that finishes the search process as soon as all Pareto optimal paths are found. The first subsection discusses some basic (mathematical) concepts that are used by the algorithm. Subsequently, in the second subsection, the multiple objective shortest path algorithm is presented. In order to avoid investigating the whole network, as the basic version of this algorithm does, a speedup can be achieved by making use of an efficient stop condition which is presented in subsection 2.3, together with a proof of the correctness of this stop condition.

2.1 Basic concepts

A network is defined as a graph $G(V,A)$ with V the set of nodes and A the set of directed links. Each link (x,y), connecting node x and node y, has assigned to it a number of values representing the costs for the different objectives, which is represented by a vector $\overline{c_{(x,y)}} = [c_{xy1}, c_{xy2}, \ldots, c_{xyn}]$. In this paper we will assume that no parallel links are allowed. Nevertheless, parallel links would have no impact on the operation of the algorithms presented here.

A path or a route in a graph is defined as a vector with a cost of the path for every objective together with an ordered list of nodes, for which holds that each pair of consecutive nodes is connected by a link. So it can be represented as $[\bar{c}, P]$ with $\bar{c} = [c_1, c_2, \ldots, c_n] = [c_i]$ the cost vector and $P = [v_1, \ldots, v_k]$ the ordered list of nodes with $\forall i \in \{1, \ldots, k-1\} : (v_i, v_{i+1}) \in A$. Here the path contains k nodes and thus $k-1$ links.

Similarly to the algorithm of Dijkstra [10], the algorithms presented in this article make use of labels to indicate the 'distance' to a certain node. A label contains a cost value for every objective and a reference to a previous label. In this way, a label can be represented as $L = [v, \bar{c}, Prev_L]$ with v the node to which this label is assigned, $\bar{c} = [c_1, c_2, \ldots, c_n]$ the value vector and $Prev_L$ the reference to the previous label. The reference to the previous label enables reconstructing the path afterwards by backtracking it. It should be noticed that, in this way, from every label a path can be constructed. While in the algorithm of Dijkstra paths are reconstructed by backtracking nodes, here this happens at label level.

In order to compare labels with each other, two relationships need to be defined on the cost vectors: dominance and ordering.

We will first define the dominance relationship between vectors in order to come to a definition of Pareto optimality, as the result of the MOSP algorithm is a Pareto optimal set.

Definition 1 (dominance)
The vector $[a_1, \ldots, a_n]$ dominates the vector $[b_1, \ldots, b_n]$ if and only if

$$(\forall i \in \{1, \ldots, n\} : a_i \leq b_i) \wedge (\exists i \in \{1, \ldots, n\} : a_i < b_i)$$

A path P_1 dominates another path P_2 if and only if the cost vector of P_1 dominates the one of P_2.
A label L_1 dominates another label L_2 if and only if the vector of L_1 dominates the one of L_2.

Definition 2 (Pareto optimality)
A path is Pareto optimal if and only if there exists no feasible path which is better in one of the objectives and not worse in all the other objectives.
A set of paths S is called Pareto optimal if and only if none of the elements of S is dominated by another feasible path.

So, for example, if we have the paths A, B and C with cost vectors $\bar{a} = [5,9,3]$, $\bar{b} = [2,1,3]$ and $\bar{c} = [5,3,2]$ respectively, then A is dominated by both B and C. The paths B and C, on the other hand, form a set of Pareto optimal paths within the set $\{A,B,C\}$.

At certain points in the algorithm, the smallest/largest label needs to be determined, so an ordering needs to be defined. We opted for a lexicographical ordering according to following definition.

Definition 3 (lexicographic ordering)
The vector $[a_1,\ldots,a_n]$ is lexicographically smaller than (denoted by $<_l$) the vector $[b_1,\ldots,b_n]$ if and only if

$$\exists k \in \{1,\ldots,n\} : (\forall i < k : a_i = b_i) \wedge (a_k < b_k)$$

A label L_1 is lexicographically smaller than a label L_2 if and only if the vector of L_1 is smaller than the one of L_2

Now, if we want to order the vectors $\bar{a}$, $\bar{b}$ and $\bar{c}$, as presented previously, from small to large then we get the sequence $\bar{b} <_l \bar{c} <_l \bar{a}$.

This relationship is a total ordering relation, as it is reflexive, anti-symmetric and transitive and two elements can always be compared to each other.

The lexicographic ordering is not the best ordering for multi-objective shortest path algorithms. In [33] it is shown that a weighted sum aggregate ordering results in a remarkable reduction of the calculation time. Nevertheless, we opted for a lexicographic ordering as it is easy to incorporate and the aggregate ordering requires a normalization of the objectives (and according fine tuning of the parameters) which may differ from network to network.

2.2 The basic algorithm

The basic algorithm presented in this section is based on the multiple objective shortest path algorithm of Martins [24], which is, in turn, based on the algorithm of Dijkstra [10]. Instead of only one label, nodes will now contain a set of labels. Moreover, this set only contains non-dominated labels. So, when adding a new label to a node, this label is only added when it is not dominated by one of the existing labels. Moreover, labels that are dominated by the new label are removed from the set. Below the pseudo code of the algorithm is given, which calculates for every node of the network the set of Pareto optimal paths from a specific origin. For clarity reasons, an explanation of the pseudo code is given in table 1.

Algorithm 1

Table 1 Explanation of the pseudo code used in algorithm 1.

`T`	the set of temporary labels
`[<node>,[v[1],...,v[n]],<label>]`	a label represented by its owner (node), a vector of values (i.e. the cost of the path from the origin to this node) and the previous label
`<node>.addLabel(<label>)`	adds a label to a node while removing all dominated labels
`T.removeMin()`	removes the minimum label from T according to the lexicographic ordering
`T.add(<label>)`	adds a label to the temporary set T
`<label>.owner()`	returns the node which is the owner of the label
`<node>.neighbors()`	returns the set of nodes which are adjacent to the node
`getLinkBetween(<node1>,<node2>)`	returns the link from node1 to node2
`[a[i]+b[i]]`	the pointwise sum of a and b. This is a short notation for $[a_1+b_1,\ldots,a_n+b_n]$

```
T := EMPTYSET;
originLabel := [origin,[0,...,0],NULL];
origin.addLabel(originLabel);
T.add(originLabel);

while(T is not empty){
     label := T.removeMin();
     owner := label.owner();
     neighbours := owner.neighbours();
     forall(nb in neighbors){
           link := getLinkBetween(owner, nb);
           newLabel := [nb,[label[i]+link.cost[i]], label];
           if(newLabel not dominated by any of nb.labels()){
             nb.addLabel(newLabel);
             T.add(newLabel);
           }
     }
}
```

Two phases can be distinguished: an initialization and the search phase. At initialization all nodes have empty label sets, except for the origin node which contains the null label, i.e. a label containing the null vector and a null reference. This label is added to the temporary set T. The algorithm runs until this temporary set is empty. At every iteration the minimum label, according to the lexicographic ordering relationship, is determined and new labels are constructed for the neighbors of its owner (the owner of a label indicates the node which contains this label). $[label[i]+link.cost[i]]$ denotes a vector which represents the pointwise sum of the vectors *label* and *link.cost*. If not dominated by any other, these new labels are added to both the corresponding neighbor and the temporary set. Moreover, as a part of the addLabel function, labels from the specific neighbor which are dominated by the new label are removed. If this new label is added to the label set of its owner, it is also added to the temporary set.

The main difference with the algorithm of Dijkstra is that this algorithm works at label level instead of node level. The temporary set contains labels and instead of changing the label of a node, the Pareto optimal set of a node is altered by adding new labels and removing dominated ones. Moreover, parents of labels are labels themselves.

2.3 Stop condition

The algorithm presented in section 2.2 investigates the whole network in order to find a set of Pareto optimal paths from one node to all others. In this research, we are interested in all Pareto optimal paths between a single origin and a single destination node. Investigating the whole network then is superfluous as the Pareto optimal set is often already found before the algorithm terminated. In this section we present a stop condition, which is similar to the one in [42], [11] and [36], also known as 'dominance by early results' or multi-objective A*. The basic idea is that labels that are dominated by one of the labels of the destination node can never lead to a Pareto optimal path, and thus are not worth investigating further. This follows from the fact that all costs are non-negative and the objectives are of the additive type.

In contrast with the multi-objective A* algorithm, we opted not to check this condition for every label when removed from the temporary set, but to keep track of the minimum values for each objective of all labels present in the temporary set and check whether this vector is dominated by one of the destination labels. As the vector with the minimum values only differs little in each iteration, the number of comparison operations needed to check the dominance can be limited. This means that labels which are dominated by one of the result labels (i.e. the labels of the destination node) may still be investigated, but this does not outweigh the smaller number of comparison operations.

Assume $[min_i(T)]$ represents the pointwise minimum of every objective value of all labels in T, resulting in the vector $[min_1(T), min_2(T), \ldots, min_n(T)]$. It can be shown that if this minimum vector is dominated by any of the destination labels, the search can be finished.

Stop Condition 1 (unidirectional s-t algorithm)
The search process can be stopped as soon as $[min_i(T)]$ *is dominated by at least one of the destination labels* $R \in L_{destination}$.

This can be proven (by contradiction) as follows.

Proof Assume that, once the stop condition holds, there still exists a label C with vector $\bar{c} = [c_i]$ which is an element of the temporary set T and which is not dominated by any of the destination labels. As this label is part of T:

$$c_i \geq min_i(T), \forall i$$

The stop condition says that $[min_1(T), \ldots, min_n(T)]$ is dominated by at least one of the destination labels $R = [destination, [r_i], Prev_R] \in L_{destination}$, which means that

$$min_i(T) \geq r_i, \forall i$$

with at least one j for which

$$min_j(T) > r_j$$

This results in

$$c_i \geq r_i, \forall i$$

with at least one j for which

$$c_j > r_j$$

According to the definition of dominance, this means that C is dominated by R, which means that our initial assumption is false and that there exist no non-dominated labels in T. □

Applying this stop condition to the basic algorithm can easily be done by substituting the line '`while(T is not empty)`' with '`while(T is not empty and` $min_i(T)$ `is not dominated by R in destination.labels())`'

3 Bidirectional multiple objective shortest path algorithm

As searching the network bidirectionally has been proven useful when speeding up algorithms, in this section a bidirectional multiple objective shortest path algorithm is presented, together with the proof that this algorithm always returns the Pareto optimal set of paths.

3.1 The algorithm

Multiple possibilities exist to speed up shortest path queries [15], like goal-directed search [19], bidirectional search [31], hierarchical search [18], etc. Some of them have already been applied to the multi-objective shortest path problem, such as the goal-directed (A*) algorithm of [42]. We opted to investigate the bidirectional speedup technique, as, with a well defined stop condition, this technique is able to guarantee still finding the optimal path for single objective shortest path problems. The basic idea of a bidirectional shortest path algorithm is that the search procedure is divided into two separate procedures: a forward search starting from the origin node and a backward search starting from the destination node.

There are two factors that influence the efficiency of bidirectional shortest path algorithms, namely the alternation between the forward and the backward search and the stop condition, of which the former is less critical than the latter. One can determine to alternate equally between the two searches or to pick the search direction with the smallest label. In this research we opted to alternate equally, but this can easily be changed.

The stop condition determines when it can be guaranteed that the optimal path has been found, and thus when both searches can be interrupted. For the single objective shortest path search, it has been proven [31] that the shortest path is found once the cost of the shortest path found so far (i.e. the minimum sum of the forward and the backward label of the same node) is smaller than or equal to the sum of the minimum

labels of the forward and the backward temporary set. It should be noted that the performance of the bidirectional shortest path algorithm (with the stop condition as mentioned before) is dependent upon the network configuration. Examples exist of networks in which the two searches proceed in different directions, resulting in two nearly completely unidirectional search trees. Predicting the average speedup factor based on qualitative or quantitative measures of the network is difficult. Nevertheless, for most planar and evenly distributed networks (i.e. networks in which the link costs are of the same magnitude), like the networks we are using in our research, a speedup of around 2 can be achieved.

The bidirectional multiple objective shortest path algorithm starts two search processes simultaneously: one from the origin and one from the destination. These search processes are similar to the search process of the unidirectional algorithm, in which new labels are constructed for the neighbors of the owner of the investigated label. These new labels are then added to the (forward or backward) label set of the specific nodes, if not dominated by any of the existing labels. Moreover, labels, which are dominated by this new label, are removed from this set. It should be noticed that nodes now contain two (Pareto optimal) label sets: one for the forward search and one for the backward search. When a new label is added to one of the label sets of a node and the other label set is not empty (i.e. this node has been visited in the other direction), this new label is combined with all labels of the other label set in order to form paths between the origin and the destination. These paths are then added to the temporary Pareto optimal solution set. It should be noted that a path is represented by a cost vector, together with a description of the path itself (i.e. a list of nodes/links). However, in order to save memory space, in the implementation only the cost vector together with the forward and the backward label, that were combined, are stored in the solution set. As each label contains a reference to its predecessor label, the path can be constructed from these two stored labels.

When all Pareto optimal paths are found both searches should be aborted. Therefore, a stop condition needs to be defined which guarantees that all Pareto optimal paths are found. For this, we make use of the pointwise minima (defined as in section 2.3) $[min_i(T_F)]$ and $[min_i(T_B)]$ of the forward and the backward temporary set respectively. A new label will be constructed which contains the pointwise sum of these minima $[min_i(T_F) + min_i(T_B)]$ and this label will be compared to the paths of the solution set ($R \in L_{result}$). In the next section, it will be proven that the following stop condition guarantees that all feasible Pareto optimal paths have been found.

Stop Condition 2 (bidirectional s-t algorithm)
The forward and the backward search can be aborted when $[min_i(T_F) + min_i(T_B)]$ *is dominated by any of the result paths* $R \in L_{result}$.

Below, pseudo code of the bidirectional multiple objective algorithm is given. The code which has not been used in algorithm 1 is explained in table 2.

Algorithm 2

```
T[forward] := EMPTYSET;
T[backward] := EMPTYSET;
```

Table 2 Explanation of the pseudo code used in algorithm 2.

`T[<direction>]`	the set of temporary labels in a specific direction
`L[result]`	the set of Pareto optimal (result) paths
`<node>.addLabel(<label>,<direction>)`	adds a label to a node in a specific direction while removing all dominated labels in that direction
`<node>.labels(<direction>)`	returns the set of labels of a node in a specific direction
`getDirection()`	determines in which direction the iteration of the algorithm will take place
`d'`	the opposite direction of d
`combine(<label>, <set of labels>)`	combines the label with each of the elements of the set to form a set of paths
L_{result}`.addPaths(<set of paths>)`	adds all elements of the set to the result set, while removing the dominated paths

```
L_result := EMPTYSET;

originLabel := [origin,[0,...,0],NULL];
origin.addLabel(originLabel, forward);
T[forward].add(originLabel);

destinationLabel := [destination,[0,...,0],NULL];
destination.addLabel(destinationLabel, backward);
T[backward].add(destinationLabel);

while([min_i(T[forward])+min_i(T[backward])]
                    is not dominated by any R in L_result){
     d := getDirection();
     label := T[d].removeMin();
     owner := label.owner();
     neighbours := owner.neighbors();
     forall(n in neighbors){
           link := getLinkBetween(owner, n);
           newLabel := [n,[label[i]+link.cost[i]], label];
           if(newLabel not dominated by any of n.labels(d)){
             n.addLabel(newLabel, d);
             T[d].add(newLabel);
             if(n.Labels(d') is not empty){
               results := combine(newLabel, n.labels(d'));
               L_result.addResults(results);
             }
           }
     }
}
```

First, both the forward and the backward temporary sets are initialized with an empty set. Similarly, the solution set L_{result} is initialized. The temporary sets are or-

dered according to the lexicographic ordering relationship, similar to the unidirectional algorithm. The solution set is a Pareto optimal set, which means that paths can be added only if they are not dominated by any of the existing paths and that existing paths which are dominated by the newly added path have to be removed from the set. Subsequently, a null label, i.e. a label containing the null vector and a null reference, is assigned to both the origin and the destination. The origin label is added to the forward temporary set, while the destination label is added to the backward set.

After initialization, the following steps are repeated until the stop condition (stop condition 2) is met. First, a direction is determined. Multiple possibilities exist to do this, like for example choosing the direction with the largest/smallest temporary set or alternating between the forward and the backward search. We opted for the latter one in order not to favor one of the directions. Then, the smallest label (according to a lexicographic ordering) is taken from the temporary set of the specific direction and investigated. This means that all neighbors of the owner of this label are determined and new labels are constructed from the investigated label and the cost vectors of the links between this owner and its neighbors. Every new label then is added to the specific (forward/backward) label set of the specific neighbor, if it is not dominated by any of the existing labels of this set. When added successfully, all existing labels which are dominated by the new label are removed from the set and the new label is added to the temporary set of the specific direction. Moreover, if this neighbor contains labels in the other direction d' (i.e. the label set of the other direction is not empty), this new label is combined with each of the labels of the label set of the other direction in order to form new paths. These new paths are then added to the solution set. It should be noted that this solution set is Pareto optimal, meaning that if the new paths are dominated by one of the existing paths, they will not be added, and that, if they are added, all existing paths dominated by them will be eliminated.

This is repeated until the stop condition is met, which means that the label containing the pointwise sum of the pointwise minima of the temporary sets is dominated by at least one of the resulting paths. These minima of the temporary sets are updated every time a new label is added to the set and every time a label is removed from it (when this label is investigated or when a temporary label is removed from a label set of a node).

This algorithm guarantees to find all Pareto optimal paths between the origin and the destination, as will be shown in the next section.

3.2 Proof of the optimality

In this section we will prove the correctness of following theorem.

Theorem 1 (optimality bidirectional MOSP algorithm)
All Pareto optimal paths are found once the stop condition holds for the bidirectional multiple objective shortest path algorithm.

Proof The bidirectional algorithm searches the network simultaneously from the origin and the destination, similar to the search process of the unidirectional algorithm. When a node contains both forward and backward labels, these labels are combined

to form paths, which are then added to the solution set, if Pareto optimal. Once the stop condition holds, all forward labels X^F with

$$\exists i : x_i^F < min_i(T_F)$$

and all backward labels X^B with

$$\exists i : x_i^B < min_i(T_B)$$

have been investigated, i.e. they are permanent labels.
This means that for all forward labels which have not been investigated yet holds that

$$\forall i : x_i^F \geq min_i(T_F) \tag{1}$$

and for all uninvestigated backward labels

$$\forall i : x_i^B \geq min_i(T_B) \tag{2}$$

The stop condition says that the pointwise sum of the pointwise minima of the temporary sets is dominated by at least one path $R = [[r_i], P_R]$ of the Pareto optimal solution set L_{result}, or

$$(\forall i : min_i(T_F) + min_i(T_b) \geq r_i) \wedge (\exists i : min_i(T_F) + min_i(T_b) > r_i) \tag{3}$$

Let us now assume that there still exists a path $C = [[c_i], P_C]$, which has not been found yet, but should be an element of the solution set. This means that this path is not dominated by any of the result paths $R = [[r_i], P_R], R \in L_{result}$. According to the dominance definition, this implies that

$$\exists i : c_i < r_i \tag{4}$$

From the operation of bidirectional algorithm we know that this path C is constructed by combining two labels of the same node, namely C^F and C^B, thus

$$\forall i : c_i = c_i^F + c_i^B \tag{5}$$

Combining equations (3), (4) and (5) results in

$$\exists i : c_i^F + c_i^B < r_i \leq min_i(T_F) + min_i(T_B) \tag{6}$$

or

$$\exists i : c_i^F + c_i^B < min_i(T_F) + min_i(T_B) \tag{7}$$

We will now demonstrate that this path C does not exist or that all paths for which condition (7) holds, have already been found by the algorithm. As path C has not been found yet, this means that either $\forall i : c_i^F \geq min_i(T_F)$ or $\forall i : c_i^B \geq min_i(T_B)$ or both. This latter case is in contradiction with comparison (7) and thus can never lead to a Pareto optimal path. Two possible situations remain, namely $\forall i : c_i^F \geq min_i(T_F) \wedge (\exists i : c_i^B < min_i(T_B) \vee \forall i : c_i^B = min_i(T_B))$ and $(\exists i : c_i^F < min_i(T_F) \vee \forall i : c_i^F = min_i(T_F)) \wedge \forall i : c_i^B \geq min_i(T_B)$. Due to the symmetric character of the bidirectional algorithm we will

only discuss one of these cases. The other one then can be deduced in the same way. Let us assume that

$$(\exists i : c_i^F < min_i(T_F) \vee \forall i : c_i^F = min_i(T_F)) \wedge \forall i : c_i^B \geq min_i(T_B)$$

This means that the backward label has not been investigated yet, while the forward label is already made permanent. The investigated forward label implies that all neighboring labels are already present in the network (some of them permanent, others not). If the path C has not been found yet by the algorithm, this means that the label C^B is not yet present in the network.

We will now look at the predecessor label C'^B of this backward label C^B. Let n'_C be the owner of this label C'^B, while n_C denotes the owner of the labels C^F and C^B. Since C^B and C'^B are neighboring labels, the nodes n_C and n'_C are connected. Moreover, C^F is a permanent label, which implies that its neighboring labels are already present in the network. From this, we can deduce that n'_C contains a forward label C'^F which has C^F as predecessor label. All 'labels' can be divided into three subsets (areas) for both the forward and the backward search: labels which have been investigated (i.e. permanent labels), labels which are added to a node but have not been investigated yet (i.e. temporary labels) and labels which have not been found yet by the algorithm. We know that C'^F is already present in the network, which means that it is either permanent or temporary. Since label C^B has not been found yet by the algorithm, label C'^B has not been investigated yet, which means that is either a temporary label or a label which has not been found yet. This leads to four cases for C'^F and C'^B as illustrated in figure 1.

Case 1: C'^F is a permanent label and C'^B is a temporary label. This path has been found by the algorithm as both the forward and the backward label are present in the network.

Case 2: C'^F is permanent and C'^B is not an element of the backward search space (i.e has not been found yet). This means that

$$(\exists i : c_i'^F < min_i(T_F) \vee \forall i : c_i'^F = min_i(T_F)) \wedge \forall i : c_i'^B \geq min_i(T_B)$$

which can be reduced to the initial situation by translating C'^F and C'^B to C^F and C^B respectively.

Case 3: Both C'^F and C'^B are temporary labels. Similar to case 1, this path has been found since both the forward and the backward label are present in the network.

Case 4: C'^F is an element of the forward temporary set and C'^B has not been found yet. As the label C'^F is an element of the forward temporary set, this means that $\forall i : c_i'^F \geq min_i(T_F)$. Moreover, the label C'^B is situated outside the backward search area, which means that $\forall i : c_i'^B \geq min_i(T_B)$. Combining these labels will result in a path for which equation (7) does not hold and which thus cannot be an element of the Pareto optimal set.

From this we can conclude that there exists no path which has not been found yet, but which should be an element of the Pareto optimal solution set. Once the stop condition holds all Pareto optimal paths are found. □

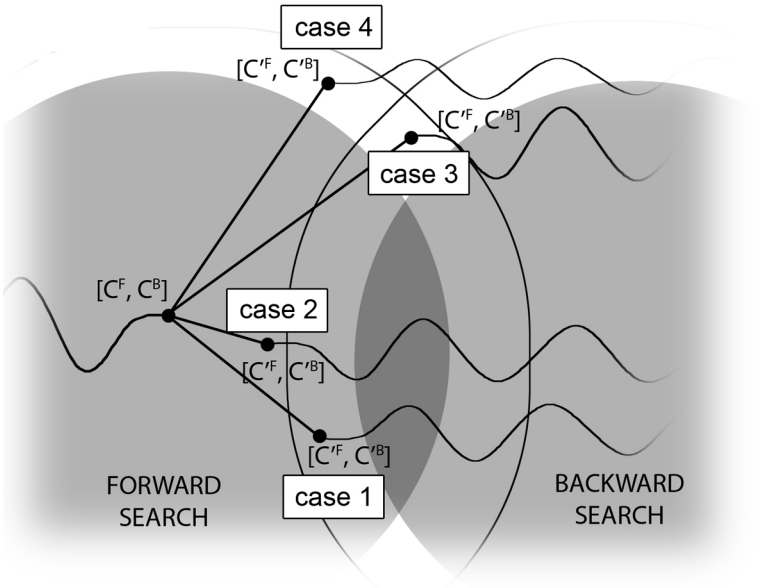

Fig. 1 Part of the forward and the backward search area. The gray areas contain all permanent labels, while the rings around these areas contain the labels of the temporary sets.

4 Experimental results

In this section the experimental results are presented. First, an overview is given of the setup in which the experiments were executed, together with a description of the networks to which the algorithms were applied. Subsequently, in the next subsection, it is shown for which networks applying the stop condition of section 2.3 in the unidirectional s-t algorithm is really advantageous. Moreover, an analysis is made of the execution time in one specific network and it is investigated how the number of objectives influences the speedup factor. In the last subsection, we will look at the bidirectional algorithm, more specifically at the speedup factors in different network configurations, the execution time in one specific network and the impact of an increasing number of objectives. Finally, the cardinality of the Pareto optimal solution set is discussed briefly.

4.1 Setup of the experiments

All algorithms presented in this paper were implemented in Java (version 1.6.0-18). We opted to represent the temporary set by a (lexicographically ordered) priority queue. Moreover, the label sets of the nodes were implemented as has hsets. All experiments were executed on a machine with the following configuration: Intel (R) Core(TM) 2 Duo CPU P8600, 2.40 GHz and 4 GB of RAM.

All graphs that are used in this research are assumed to be directed. All undirected links were replaced by two inverse directed links. There are 4 network configurations that are used: transportation, square grid, random and complete networks.

A number of transportation networks (Transport X) were provided by a Belgian company who is a major industrial player in the field of traffic information. In this research, we made use of a network representing the Belgian road network (B). Moreover, the three transportation networks that were used in [36] and [37] were retrieved. These networks are road networks of the US, more specifically those of Washington DC (DC), Rhode Island (RI) and New Jersey (NJ). Next to the length of the links (i.e. streets, or part of streets), travel time information was made available. For the experiments, the length (in meters) was utilized as first objective and the travel time (in milliseconds) as second objective. Since we do not have more data at hand, for the other objectives random values between 0 and 1000 were used.

In a square grid network (Square Grid N) nodes are placed on a square grid ($N \cdot N$) and all the direct (horizontal and vertical) neighbors are connected to each other. This means that each node has at most 4 neighbors. A grid network of this type is often denoted with the term 'Manhattan network'. The cost vectors assigned to the links consist of n (assumed that there are n objectives) completely uncorrelated random values between 0 and 1000.

A random network (Random N(d)) contains N nodes and ($d \cdot N$) links. These nodes are randomly connected to each other by the specified number of links, where we made sure that the networks are connected. This means that the network needs to contain at least (N-1) links. Again, (completely uncorrelated) random values between 0 and 1000 were used to represent the different costs of the links. Random networks have the advantage that the number of nodes and the number of links can be changed easily. In this research we will use the random networks to determine trends in the speedup factors in function of the number of nodes in the networks and the average node degree in the networks.

Finally, a complete (or full dense) network (Complete N) is a network with N nodes in which each node is connected to each of the other nodes. Here too, we made use of n completely uncorrelated random values (between 0 and 1000) to represent the cost vectors of the links.

An overview of all networks that were used in the experiments is given in table 3.

All experiments were executed for 1000 randomly selected origin-destination pairs and the presented results are averages over these 1000 multiple objective shortest path calculations. For both presented speedup techniques, speedup factors are reported, with a speedup factor (SUF) of algorithm A over algorithm B is defined as:

$$SUF(A/B) = \frac{avg.\ execution\ time\ algorithm\ B}{avg.\ execution\ time\ algorithm\ A}$$

Table 3 Overview of all network configurations used for the experiments.

network configuration	# nodes	# links	average outdegree
Transport (DC)	9 559	29 765	3.1
Transport (B)	23 080	50 364	2.2
Transport (RI)	53 658	138 083	2.6
Transport (NJ)	330 386	869 471	2.6
Complete 100	100	9 900	99
Complete 300	300	89 700	299
Square Grid 30	900	3 480	3.86
Square Grid 100	10 000	39 600	3.96
Random 10000(3)	10 000	30 000	3
Random 20000(3)	20 000	60 000	3
Random 100000(3)	100 000	300 000	3
Random 10000(4)	10 000	40 000	4
Random 10000(5)	10 000	50 000	5
Random 10000(6)	10 000	60 000	6

A speedup factor higher than 1 means that algorithm A is faster than algorithm B. For clarity issues, we will denote the unidirectional MOSP algorithm without the stop condition with U, the unidirectional MOSP algorithm with the stop condition with US and the bidirectional MOSP algorithm (with the stop condition) with B.

4.2 The unidirectional MOSP algorithm

In this section we will demonstrate that using the stop condition (see section 2.3) indeed speeds up a number of unidirectional shortest path calculations in some network configurations. In the first part a comparison will be made of the speedup factors for the different network configurations. Subsequently, we will look at one specific configuration and see how the calculation time varies with a varying 'distance' between the origin and the destination.

Table 4 shows the average execution time of the unidirectional MOSP algorithm without the stop condition and the speedup achieved by using stop condition 1 (SUF(US/U)) for a number of network configurations, and this for both 2 and 3 objectives. One can see that the speedup factors for the transportation networks are much higher than those for the other networks. The results for the three US network (DC, RI and NJ) are comparable to the results of [36]. For the square grid networks, on the other hand, slightly higher speedup factors are perceived. Applying the stop condition in a complete network does not result in smaller execution times. At all times, the whole network needs to be investigated and the algorithm with the stop condition is even slightly slower than the algorithm without the stop condition, due to the overhead of checking the stop condition. Looking at the random networks, two trends are perceived. When the number of nodes in the network increases, the speedup factor increases as well. The larger the network is, the higher the chance that the origin and the destination node are 'closer' to each other and that only a smaller part of the network needs to be investigated. If the number of nodes in the network is kept constant and the average out degree of the nodes is increased, the speedup factor

Table 4 The execution times of the unidirectional MOSP algorithm without the stop condition and the speedup factors of the unidirectional algorithm with the stop condition over the one without the stop condition (SUF(US/U)) in different network configurations.

	2 objectives		3 objectives	
network configuration	**CPU time (ms)**	**SUF(US/U)**	**CPU time (ms)**	**SUF(US/U)**
Transport (DC)	127	2.35	962	2.55
Transport (B)	443	3.35	47 657	3.41
Transport (RI)	1 914	2.39	156 494	2.86
Transport (NJ)	18 191	2.96	1 784 817	3.25
Complete 100	40	0.99	229	0.98
Complete 300	839	0.99	7 837	0.99
Square Grid 30	136	1.33	16 312	1.44
Square Grid 100	978	1.59	103 435	1.62
Random 10000(3)	96	1.25	102	1.41
Random 20000(3)	547	1.58	717	1.96
Random 100000(3)	2 155	2.00	6 124	2.69
Random 10000(4)	199	1.15	674	1.19
Random 10000(5)	314	1.02	1 225	1.01
Random 10000(6)	498	0.99	2 068	0.99

decreases. For the highest out degrees, this is similar to the results of the complete networks.

It is known that for the algorithm of Dijkstra interrupting the search process as soon as the label of the destination node is permanent is only advantageous (with respect to the calculation time) if the origin and the destination node are situated not too far from each other. The calculation time increases monotonously in function of the distance (i.e. shortest path length) between the origin and the destination node. Now, we want to investigate the relationship between the calculation time of the unidirectional MOSP algorithm and the 'distance' between the origin and the destination node. Since, for multiple objective shortest path algorithms, there is no single value that indicates the 'distance' between two nodes, an estimation needs to be used. We will assume the minimum value of the first objective as an approximation of how far apart two nodes are from each other.

As the speedup factors for the transportation networks are the highest, and the concept of 'distance' is most significant for these networks, we opted to examine this for transportation networks only. Here we made use of the Transport (DC) network. Figure 2 shows the execution times of the unidirectional algorithm with and without stop condition in function of the 'distance' (i.e. minimum value of the first objective) between the origin and the destination node, for both 2 and 3 objectives. Moreover, the minimum and the maximum values of the execution times are indicated. The execution times of the algorithm without the stop condition are constant as each time the whole network is investigated. As expected, when the stop condition is applied smaller execution times are perceived for the 'shorter' paths and for the 'longer' paths the execution times tilt to those of the algorithm without the stop condition. The bars in the figures show that there is more variability in the execution times for the paths with medium 'length'. The curve for 3 objectives reaches its upper limit sooner than the one for 2 objectives. When dealing with 3 objectives, more of the network needs to be investigated, before the search can be aborted.

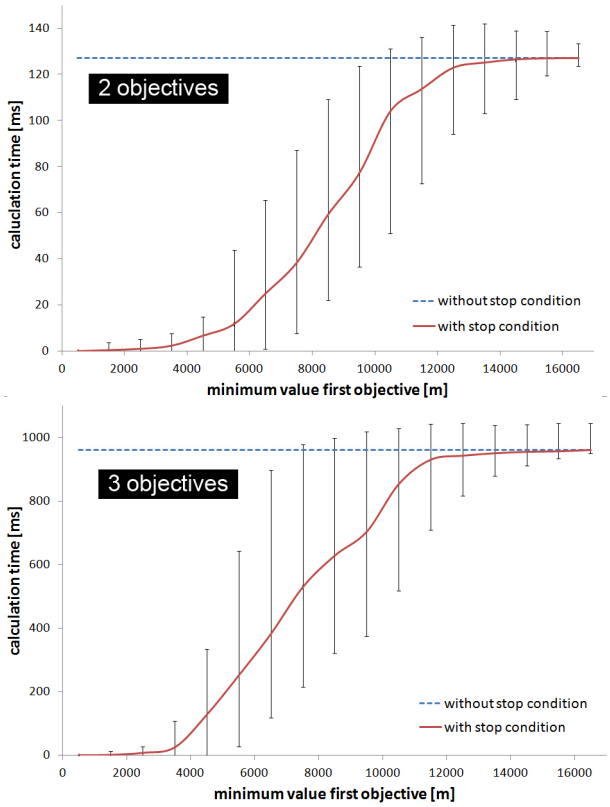

Fig. 2 Calculation time of multiple objective shortest path algorithm with and without stop condition in the Transport (DC) network with 2 objectives (top) and 3 objectives (bottom).

In table 4 results are shown for both two and three objectives. The speedup factors of the unidirectional algorithm with the stop condition over the one without the stop condition are slightly higher for 3 objectives than for 2 objectives. We now will investigate, for one particular case (the Transport (DC) network), if this trend continues with higher number of objectives. Results are shown in table 5. It can be seen that indeed the speedup factor increases as the number of objectives increases. Nevertheless, these increases are relatively small.

4.3 The bidirectional MOSP algorithm

In this section we will demonstrate how searching the network bidirectionally in most cases indeed speeds up the calculations. Besides examining the speedup factors for the different network configurations, we will see how the speed up factor reacts to an increasing number of objectives in transportation networks. Moreover, it will be indicated for which paths searching bidirectionally is really advantageous. Next to this, we will have a quick look at the cardinality of the solution set.

Table 5 Speedup factors (SUF) of the unidirectional MOSP algorithm for different number of objectives in the Transport(DC) network.

# obj.	SUF(US/U)
2	2.35
3	2.55
4	2.69
5	2.76
6	2.87
7	2.97
8	3.05

Table 6 The average execution times of the unidirectional MOSP algorithm with the stop condition and the speedup factors of the bidirectional algorithm over the unidirectional algorithm with the stop condition (SUF(B/US)) in different network configurations.

	2 objectives		3 objectives	
network configuration	**CPU time (ms)**	**SUF(B/US)**	**CPU time (ms)**	**SUF(B/US)**
Transport (DC)	54	5.76	377	12.47
Transport (B)	132	5.91	13 976	14.46
Transport (RI)	801	7.77	54 718	16.16
Transport (NJ)	6 146	10.08	549 174	21.43
Complete 100	40	0.60	305	0.50
Complete 300	847	0.58	7 916	0.37
Square Grid 30	102	1.49	11 328	1.57
Square Grid 100	615	1.79	63 849	1.96
Random 10000(3)	77	3.29	72	3.54
Random 20000(3)	346	3.99	66	3.99
Random 100000(3)	1 078	4.52	2 277	4.67
Random 10000(4)	173	2.75	566	2.51
Random 10000(5)	308	2.01	1 213	1.30
Random 10000(6)	503	1.04	2 89	0.53

Table 6 shows, next to the average cardinality of the solution set, the speedup factors of the bidirectional MOSP algorithm over the unidirectional algorithm with the stop condition, for both 2 and 3 objectives. Clearly noticeable is the fact that the speedup factors in the transportation networks are higher than those in the other network configurations. Searching the network bidirectionally is certainly not a good choice when dealing with complete networks. Here, the bidirectional algorithm is even slower than its unidirectional counterpart. For square grid networks, only relatively small speedups are achieved. Looking at the random networks, some trends can be identified. When the number of nodes in the network increases, the speedup factor increases too. If, on the other hand, the number of nodes is kept constant, and the average out degree of the nodes is increased, the speedup factor decreases. These trends are similar to the ones observed for the speedup factor of the unidirectional algorithm with the stop condition over the one without the stop condition.

Similar to the results of section 4.2 we looked at the calculation time of the bidirectional MOSP algorithm in the Transport (DC) network in function of the 'distance' (i.e. minimum value of the first objective) between the origin and the destination node. This is shown in figure 3, together with the calculation time of the unidirectional algorithm with the stop condition, which is the same as in figure 2. One

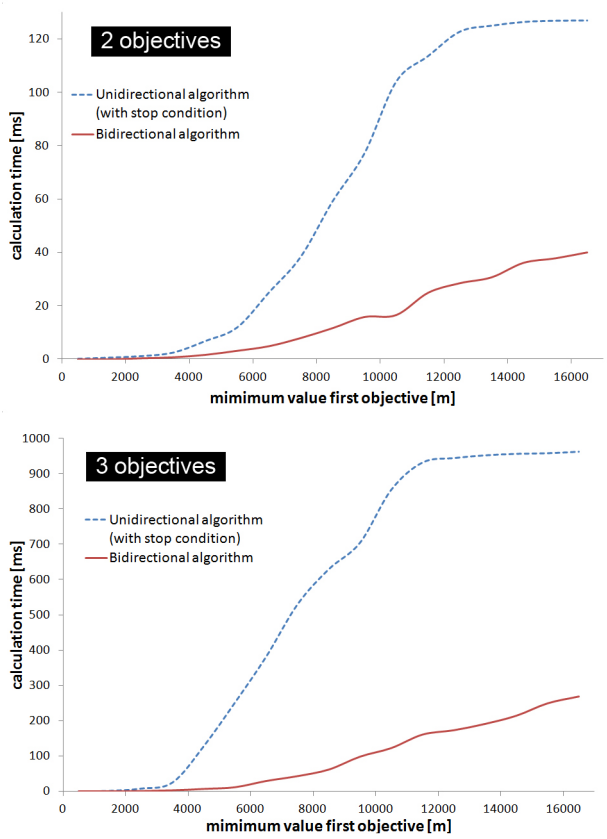

Fig. 3 Comparison of unidirectional and bidirectional algorithm in function of the number of hops in the shortest path.

can see that the calculation times for the bidirectional algorithm are indeed much smaller. The curves of the bidirectional algorithm increase monotonously, seemingly without reaching an upper limit, like the curves of the unidirectional algorithms do. This means that the highest speedups are achieved for the medium length paths. For the shortest paths the curves are nearly flat for both algorithms, resulting in smaller speedup factors. When the 'distance' between the origin and the destination node increases, the calculation time of the unidirectional algorithm increases faster than that of the bidirectional algorithm. This results in higher speedup factors. For the longest paths the unidirectional algorithm has reached its upper limit, while the calculation time of the bidirectional algorithm still increases, resulting in smaller speedup factors. So, the highest speedup factors are calculated for the medium length paths.

Subsequently, we investigated the impact of the number of objectives on the speedup factor. In table 6 results are shown for both 2 and 3 objectives, where it can

Table 7 Speedup factors (SUF) of the bidirectional MOSP algorithm over the unidirectional algorithm for different number of objectives in the Transport(DC) network.

# obj.	SUF(US/U)
2	5.76
3	12.47
4	12.88
5	13.20
6	13.24
7	14.27
8	15.50

Table 8 Average cardinality solution set for different network configurations.

	avg. # L_{result}	
	2 objectives	3 objectives
Transport (DC)	14.97	32.67
Transport (B)	21.72	48.03
Transport (RI)	30.40	87.98
Transport (NJ)	83.23	151.15
Complete 100	12.83	58.79
Complete 300	17.98	120.29
Square Grid 30	23.52	242.49
Square Grid 100	173.64	502.37
Random 10000(3)	3.86	3.67
Random 20000(3)	3.94	7.87
Random 100000(3)	4.27	9.46
Random 10000(4)	4.89	10.98
Random 10000(5)	5.71	13.84
Random 10000(6)	6.56	17.11

be seen that the speedup factors are in most cases higher when there are 3 objectives. For the complete networks, where it is not advantageous to search bidirectionally, adding an objective is worse for the execution time of the bidirectional algorithm. For the random networks with a higher average out degree of the nodes, this is also the case. We now focus on the transportation networks and calculated the average speedup of the bidirectional algorithm over the unidirectional algorithm (with stop condition) for a different number of objectives. Results are shown in table 7. It is shown that indeed the speedup factor increases with an increasing number of objectives. There is a difference in the increase between two and three objectives and that between a higher number of objectives. This can be explained by the fact that for the first two objectives actual distances and travel time information was used, while for the other objectives random values were selected.

Besides this, we investigated the average cardinality of the solution set. Looking at table 8 one can see that this cardinality is highly dependent on the network configuration with the highest cardinalities for the square grid networks and the lowest ones for the random networks. Moreover, the average cardinality is dependent on both the number of nodes in the network and the average out degree of the nodes. When the number of nodes in the network increases, the cardinality of the solution set increases. Similarly, when the average out degree of the nodes increases, the cardinality of the solution set does also. This is as expected.

5 Conclusion

In this article the main goal was to speed up the multiple objective shortest path (MOSP) algorithm as presented by Martins [24], while still maintaining the guarantee to find all Pareto optimal paths. The improvements prove to be very useful, especially in large transportation networks.

First, a stop criterion for the unidirectional algorithm was introduced which finishes the search process as soon as the destination node contains all its labels, which can be translated to the Pareto optimal paths. It has been proven that, once this stop condition holds, all Pareto optimal paths are found. Experiments have shown that a remarkable speedup can be achieved in transportation networks and that this speedup is the largest if the origin and the destination node are 'close' to each other. Moreover, the speedup factor increases when dealing with more objectives.

Secondly, a bidirectional MOSP algorithm was introduced, which searches the network from the origin and the destination simultaneously. It is proven that this algorithm always returns the complete Pareto optimal set of paths. In the experiments, it can be seen that the speedup achieved by searching the network bidirectionally is dependent upon the network configuration, the number of nodes in the network, the average out degree of the nodes, the relative position of the origin and the destination node, and the number of objectives.

From the experiments it can be concluded that the speedup measures presented in this article are particularly well suited for (large) transportation networks. The highest speedup is achieved by searching the network bidirectionally.

References

1. 9th DIMACS Implementation Challenge: Shortest Paths, http://www.dis.uniroma1.it/∼challenge9 (2006)
2. Azevedo JA., Costa MEOS., Madeira JJERS., Martins EQV., An algorithm for the ranking of shortest paths, European Journal of Operational Research, 69, 97-106 (1993)
3. Bellman R., On a Routing Problem, Quarterly of Applied Mathematics, 16, 87-90 (1958)
4. Bornstein C., Maculan N., Pascoal MMB., Pinto L., Multiobjective combinatorial optimization problems with a cost and several bottleneck objective functions: an algorithm with reoptimization, Computers & Operations Research, 39, 1969-1976 (2012)
5. Brumbaugh-Smith J., Shier D., An empirical investigation of some bicriterion shortest path algorithms, European Journal of Operational Research, 43, 216-224 (1989)
6. Clímaco JNC., Martins EQV., A bicriterion shortest path algorithm, European Journal of Operational Research, 11, 399-404 (1982)
7. Clímaco JNC., Pascoal MMB., Multicriteria path and tree problems - Discussion on exact algorithms and applications, International Transactions in Operations Research, 19, 63-98 (2012)
8. de Lima Pinto L., Bornstein CT., Maculan N., The tricriterion shortest path problem with at least two bottleneck objective functions, European Journal of Operational Research, 198, 387-391 (2009)
9. Demeyer S., Audenaert P., Slock B., Pickavet M., Demeester P. Multimodal transport planning in a dynamic environment, Conference on Intelligent Public Transport Systems, Amsterdam, 155-167 (2008)
10. Dijkstra EW., A Note on Two Problems in Connexion with Graphs, Numerische Mathematik, 1, 269-271 (1959)
11. Disser Y., Müller-Hannemann M., Schnee M., Multi-Criteria Shortest Paths in Time-Dependent Train Networks, Experimental Algorithms, 7th International Workshop, WEA 2008, Provincetown, MA, USA, 2008, 347-361 (2008)

12. Ehrgott M., Gandibleux X., A survey and annotated bibliography of multiobjective combinatorial optimization, OR Spektrum, 22, 425-460 (2000)
13. Ehrgott M., Gandibleux X., Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys, International Series in Operations Research and Management Science, 52, Springer (2002)
14. Eppstein D., Finding the k shortest paths. SIAM Journal on Computing, 28, 652-673 (1998)
15. Fu L., Sun D., Rilett LR., Heuristic shortest path algorithms for transportation applications: State of the art, Computers and Operations Research, 33, 3324-3343 (2006)
16. Gandibleux X., Beugnies F., Randriamasy S., Martins' algorithm revisited for multi-objective shortest path problems with MaxMin cost function, 4OR - a Quarterly Journal for Operations Research, 4, 47-59 (2006)
17. Garroppo RG., Giordano S., Tavanti L., A survey on multi-constrained optimal path computation: Exact and approximate algorithms, Computer Networks, 54, 3081-3107 (2010)
18. Geisberger R., Sanders P., Schultes D., Delling D., Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks, in: McGeoch (Ed.): Workshop on Experimental Algorithms, LNCS 5038, Springer-Verlag, Berlin/Heidelberg, 319-333 (2008)
19. Goldberg A., Kaplan H.,Werneck R., Reach for A*: Efficient point-to-point shortest path algorithms, in: Workshop on Algorithm Engineering & Experiments, Miami, 129-143 (2006)
20. Guerriero F., Musmanno R., Label Correcting Methods to Solve Multicriteria Shortest Path Problems, Journal of Optimization Theory and Applications, 111, 589-613 (2001)
21. Hansen P., Bicriterion path problems, in: Fandel G, Gal T (Eds), Multiple Criteria Decision Making: Theory and Applications, Lecture Notes in Economics and in Mathematical Systems 177, Springer: Heidelberg, 109-127 (1980)
22. Jiménez V., Marzal A., Computing the K shortest paths: a new algorithm and experimental comparison, in: Vitter JS. Zaroliagis CD. (eds) Proceedings of the 3rd International Workshop on Algorithm Engineering, LNCS 1668, Springer-Verlag, Berlin/Heidelberg, 15-29 (1999)
23. Jiménez V., Marzal A., A lazy version of Eppstein's k shortest path algorithm, in: Jansen K., Margraf M., Matrolli M., Rolim J. (eds) Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms, LNCS 2647, Springer-Verlag, Berlin/Heidelberg, 179-191 (2003)
24. Martins EQV., On a multicriteria shortest path problem, European Journal of Operational Research, 16, 236-245 (1984)
25. Martins EQV., An algorithm for ranking paths that may contain cycles, European Journal of Operational Research, 18, 123-130 (1984)
26. Martins EQV., Paixão JM., Rosa MS., Santos JLE., Ranking multiobjective shortest paths, Pré-publicações do Departamento de Matemática 07-11, Universidade de Coimbra (2007)
27. Martins EQV., Pascoal MMB., Santos JLE., Deviation algorithms for ranking shortest paths. The International Journal of Foundations of Computer Science, 10, 247-263 (1999)
28. Martins EQV., Pascoal MMB., Santos JLE., Labeling algorithms for ranking shortest paths. Technical Report 001 CISUC (2000)
29. Martins EQV., Pascoal MMB., Santos JLE., A new improvement for a K shortest path algorithm. Investigação Operational, 21, 47-60 (2001)
30. Martins EQV., Santos JLE., The Labeling Algorithm for the Multiobjective Shortest Path Problem, CISUC Technical Report TR 99/005, University of Coimbra, Portugal (1999)
31. Nicholson JAT., Finding the shortest route between two points in a network. Computer Journal, 9, 275-280 (1966)
32. Pangilinan JMA., Janssens GK., Evolutionary Algorithms for the Multiobjective Shortest Path Problem, International Journal of Applied Science, Engineering and Technology, 4, 205-210 (2007)
33. Paixão JM., Santos JL., Labelling methods for the general case of the multi-objective shortest path problem - a computational study, Pré-publicações do Departamento de Matemática 07-42, Universidade de Coimbra (2007)
34. Paixão JM., Santos JL., A new ranking path algorithm for the multiobjective shortest path problem, Pré-publicações do Departamento de Matemática 08-27, Universidade de Coimbra (2008)
35. Pinto L., Pascoal MMB., On algorithms for tricriteria shortest path problems with two bottleneck objective functions, Computers & Operations Research, 37, 1774-1779 (2010)
36. Raith A., Speed-up of labelling algorithms for biobjective shortest path problems, Proceedings of the 45th Annual Conference of the ORSNZ, Auckland, New Zealand, 313-322 (2010)
37. Raith A., Ehrgott M., A comparison of solution strategies for biobjective shortest path problems, Computers & Operations Research, 36, 1299-1331 (2009)
38. Sastry V., Janakiraman T., Mohideen S., New algorithms for multi-objective shortest path problem, Opsearch, 40, 278-298 (2003)

39. Serafini P., Some considerations about computational complexity for multiobjective combinatorial problems, Recent advances and historical development of vector optimization, 294, 222-232 (1986)
40. Skriver AJV., A classification of bicriterion shortest path (BSP) algorithms, Asia Pacific Journal of Operational Research, 17, 192-212 (2000)
41. Skriver AJV., Andersen K., A label correcting approach for solving bicriterion shortest-path problems, Computers & Operations Research, 27, 507-524 (2000)
42. Stewart BS., White CC., Multiobjective A*, Journal of the Association for Computing Machinery, 38, 775-814 (1991)
43. Vincke P., Problemes multicritères, Cahiers Centre Etudes Recherche Operationnelle, 16, 425-439 (1974)