Continuous and Discrete Time Label Setting Algorithms for the Time Dependent Bi-Criteria Shortest Path Problem

Aphrodite Veneti, Charalampos Konstantopoulos and Grammati Pantziou
Department of Informatics, University of Pireaus, Greece, aveneti@unipi.gr,
Department of Informatics, University of Pireaus, Greece, konstant@unipi.gr,
Department of Informatics, Technological Educational Institution of Athens, Greece, pantziou@teiath.gr

Abstract In this paper we address the bi-criteria shortest path problem in a network with time-dependent cost functions, a travel time constraint and zero waiting time at nodes. We propose a discrete time forward label setting algorithm which outperforms other approaches in the literature. We also propose a second, more general, forward label setting algorithm that can handle both discrete and continuous travel time functions. A speeding up technique for the proposed algorithms is also provided. The algorithms are compared with a backward algorithm, the best so far algorithm in the literature that solves the problem at hand and the experimental results confirm the higher efficiency of the proposed algorithms.

Keywords Multi-criteria optimization, Label setting algorithm, Time dependent networks, Resource-constrained shortest path

1. Introduction

The Multi-Objective Shortest Path problem (MOSP) is a fundamental multi-objective optimization problem with many applications. In a multi-objective optimization problem, there is no a single optimal solution, but a set of optimal solutions in a sense that a small improvement in one objective would deteriorate the others. So, the main task is to seek the trade-off among the different objective functions and find the so-called Pareto set of non-dominated solutions $P$. Specifically, the set $P$ is the set of feasible solutions whose vector of the objective values is not dominated by any other solution. In a multi-objective minimization problem (maximization problem) a solution $p$ dominates another solution $q$ if and only if $f_i(p) \leq f_i(q)$ (respectively, $f_i(p) \geq f_i(q), \forall 1 \leq i \leq d$), with a strict inequality for at least one $i \in \{1, \ldots, d\}$, where $f_i$ is the cost function of the $i$-th objective and $d$ is the number of objectives of the problem at hand. The MOSP problem is NP-complete and \#P-complete. It is also an intractable problem, due to the exponential size of the Pareto set, even in the case of two objectives. Although the size of the Pareto set may be exponential in theory, however, this worst case usually does not arise in practice. Thus, finally, the decision maker can choose a solution fulfilling his/her criteria from a relatively small set of Pareto solutions.

Advances in transportation path planning have triggered the interest in the dynamic MOSP problem, where the topology or the arc costs of the network are changing over time. In the time-dependent shortest path problem, we deal with time-varying arc cost values. In [4], Chabini distinguishes among different types of the time dependent shortest path problem. Some of the categorization criteria are: discrete versus continuous representation
of time, whether there is a single departure time or a time window, and also whether waiting at the nodes of the network is allowed or not. An important property of the network which much differentiates the complexity of the time-dependent algorithms is the FIFO property [4] [13] [20]. A network is said to fulfill the FIFO property if all of its arcs fulfill that property i.e., for each arc \((i,j)\) of the network, earlier departure from \(i\) always leads to earlier arrival at \(j\), that is, the arrival events at \(j\) are in the same chronological order as the departure events at \(i\). More explicitly, it may be defined in the following mathematical form:

\[
\forall (i,j,t) \quad t + c(i,j,t) \leq (t+1) + c(i,j,t+1)
\]

where cost function \(c(i,j,t)\) denotes the cost for traversing arc \((i,j)\) at time instance \(t\) and has integer-valued domain and positive integer-valued range. When the FIFO property holds, waiting at the nodes of the network is pointless since leaving immediately from each node is always a beneficial practice leading to optimal solution paths. Computing shortest paths in FIFO networks is a polynomially solvable problem [13]. On the other hand, when the FIFO property does not hold, optimal solutions may require waiting at certain nodes of the network. Therefore, in non-FIFO networks the complexity of the time-dependent shortest path problem depends on the waiting policy at nodes. If waiting is allowed, the problems is polynomially solvable otherwise, the problem is NP-hard [19].

In this paper we address the time-dependent bi-criteria shortest path problem in non-FIFO networks with fixed departure time, no waiting at nodes and a constraint on the total travel time. The problem finds application in ship routing where frequent stops in the middle of the sea or alternating ship speed frequently is not a common practice [21], [16], [15]. To the best of our knowledge, only a few algorithmic approaches for the problem have been proposed in the literature [14], [10], [11]. Also, despite of the significance of the problem and its applications, there is no recent research work on the topic. In contradiction with this fact, various methods have been recently proposed facing relevant problems such as the time-dependent itinerary planning problem [24], [1], [2], [9] and the dynamic path planning under uncertainty [12], [18], [22].

The existing algorithmic approaches for tackling the multi-objective time-dependent shortest path problem [14], [10], [11] are either forward or backward dynamic programming approaches which are not efficient when waiting at nodes is forbidden. In this paper, we propose new forward label setting algorithms for solving the time-dependent bi-criteria shortest path problem with an upper bound on the total travel time, no waiting at nodes and fixed departure time from the source node. Forward label-setting algorithms seem to be more suitable for addressing the above problem than backward algorithms. Specifically, in a forward label-setting algorithm, the computation of the time arriving at a node is an easier task than in a backward algorithm; a forward algorithm starts from a single label at the source node corresponding to the fixed departure time and then is able to accurately determine the arrival time at each node of the network, during its execution. On the other hand, in a backward algorithm, the execution starts from the destination node and since the possible arrival time instances at the destination are not known in advance, we have to start with a large number of labels, each corresponding to a different possible arrival time. As a result, a much larger set of labels is created in the nodes of the network during the algorithm execution. Therefore, we employ a forward label-setting approach to tackle the aforementioned problem. Specifically, we propose a new discrete time forward label setting algorithm as well as a more general forward label setting algorithm that can handle both discrete and continuous travel time values. The number of the labels being processed during the algorithm execution is kept at a reasonable level, reducing so the computational overhead. Moreover, our algorithms can be easily extended to handle multiple departure time instances at the source node, as well. The algorithms are compared against the backward algorithm of Hamacher et al’s [11], the best so far algorithm in the literature that solves the
problem at hand, and the experimental results confirm the higher efficiency of the proposed algorithms.

The rest of the paper is organized as follows. The related work is presented in Section 2. In Section 3, we formally define the problem. Assuming that the travel time function is discrete-valued, we then propose a new discrete time forward label setting algorithm. Next, we give a label setting algorithm that can handle both discrete and continuous travel time values. Efficiency of the proposed algorithms is confirmed through a series of experiments presented in Section 4, where the proposed algorithms are compared against the best so far approach in the literature for the problem at hand, which is a backward label setting algorithm. Finally, the conclusions are presented in Section 5.

2. Related Work

Various single objective shortest path algorithms have been proposed, handling time-dependent arcs costs. We can differentiate between algorithms where time is considered to be integer-valued and algorithms where time is regarded as a continuous real variable.

Cooke’s and Halsey’s algorithm [5] is the first to deal with time dependency. The algorithm is based on Bellman’s optimality principle in order to retrieve all-to-all fastest paths in a FIFO network where functions have discrete ranges and domains. Since the FIFO property holds, waiting at the nodes is not needed. The authors propose a backward dynamic programming formulation of the problem for finding the shortest route between any two nodes and for any departure time at the source node. However, the approach is not that efficient when working with a departure time window at the start node in place of the single departure time.

Dreyfus [8] also modified Dijkstra’s static shortest path algorithm in order to calculate the fastest path between two nodes for a given departure time. His algorithm can work with discrete and continuous delay functions and also handle both waiting and no waiting at nodes. Hence the proposed method is suitable for both FIFO and non-FIFO dynamic networks. Later, Kaufman and Smith [13] formally proved that if FIFO property holds, there is a modified label setting algorithm with the same time complexity as the static shortest path algorithm that solves the time dependent shortest path problem.

Chabini’s [4] approach deals with a slightly different time-dependent shortest path problem. Specifically, he makes the assumption that after a finite time interval, there is no time-dependency and hence the computation of fast paths boils down to a static shortest path problem. This is a strong assumption and facilitates the development of a backward, label setting, discrete time algorithm which computes shortest paths for all possible departure time instances in both FIFO and non-FIFO networks. Ding et al. [7] proposed a continuous time algorithm based on the Dijkstra algorithm for calculating shortest paths for a given departure time window. Their algorithm is suitable for all networks with or without the FIFO property.

The idea of bounded waiting at the nodes of a non-FIFO network was first introduced by Cai et al. [3]. The proposed approach deals with the time-dependent minimum cost path whose total travel time is below a certain bound and for a fixed departure time at the source node. They proposed three variations of a discrete time algorithm that deal with three different waiting polices at nodes, namely unrestricted waiting, forbidden waiting and bounded waiting.

In [20], Orda et al. propose a set of algorithms for finding the fastest path in time-dependent networks, with arbitrary travel time functions, for a fixed departure time instance as well as for all possible departure time instances. They first distinguish three different solution strategies: unlimited waiting at nodes, no waiting at nodes and waiting only at the source node. An important result in [20] is that if the delay function is continuous or piecewise continuous, there is a simple shortest path with unlimited waiting at the source
node and no waiting at the other nodes of the path which has the same total travel time as an optimal path with unlimited waiting at all nodes. More specifically, they proved that every shortest topological path with unlimited waiting at nodes is also a shortest topological path with waiting only at the source node, having the same total travel time but different arrival times at intermediate nodes. They also pointed out that if waiting at nodes is forbidden, shortest paths may contain loops. Furthermore, finding loopless paths in time-dependent networks where waiting is restricted is a NP-complete problem and this claim was proved by a reduction from the Hamiltonian Path problem [19].

In [19], Orda et al deal with a slightly different problem where the total travel time is upper-bounded and the objective is to find a minimum cost path. Their algorithm can handle both waiting and no waiting policies at nodes, and also works with either a single departure time or departure time window at the source node. In many non-FIFO networks, where also there is an upper bound on the waiting time permitted at nodes, it may not be feasible to find finite optimal paths and Orda et al formulated a criterion guaranteeing the existence of finite ones. Finally, Ziliaskopoulos and Mahmassani [23] proposed a label correcting method for computing the shortest paths from all nodes to a single destination for all departure times, without the FIFO property.

Although research has intensively focused on multi-objective optimization, relatively few works have addressed the problem of multiple criteria optimization in dynamic, time-dependent networks. In [14], Kostreva and Wiecek modified the dynamic programming approach of Cooke and Halsey [5] to deal with the multi-objective time-dependent path planning and developed a discrete time backward dynamic programming approach where the cost functions are positive. Under the assumption of continuous time and monotonically increasing cost functions (ensuring so the FIFO property for every cost function), forward principle of optimality in time-dependent Pareto optimal paths is proved to hold and a forward dynamic programming approach was also developed.

Getachew et al. [10] proposed a recursive algorithm with alternating forward and backward phases. Their algorithm does not require time discretization or the FIFO assumption. They only assume that there exist lower and upper bounds in the values of the cost functions. At each round of the algorithm, a backward dynamic programming phase finds Pareto optimal paths using the lower bounds of edge costs and then a forward evaluation of paths runs.

Hamacher et al. [11] extended the Dijkstra algorithm and developed a backward label setting algorithm solving the time-dependent shortest path problem with non-negative cost functions, fixed departure time and arbitrary arrival time at the destination node. Although the backward principle of optimality holds, Hamacher’s algorithm does not essentially take advantage of this fact for reducing the number of generated labels during the algorithm execution. Eventually, it generates a lot of labels during its execution, namely a set of non-dominated labels for each node and for each time instance. However, the proposed algorithm appears to be more efficient than the algorithm presented in [14].

From the discussion above, it is now clear that there are algorithms that deal with the time-dependent bi-criteria shortest path problem. However, these algorithms do not efficiently handle the no waiting policy. In this paper, we propose a discrete time forward label setting algorithm which solves the problem at hand more efficiently than the existing approaches. Additionally, we propose a more general label setting algorithm that can work with continuous as well as discrete time values. Finally, our algorithms can handle either a single departure time or a finite set of departure time instances at the source node.

3. The Time-Dependent Bi-Objective Shortest Path Problem with Zero Waiting Time

In this section, we first formally define the bi-objective, time-dependent shortest path problem with zero waiting time at nodes, fixed departure time from the source node and a
Formally, a path objectives can be improved in value without degrading the value of the other objective. A solution is termed as a non dominated solution if none of the functions may be conflicting, a single solution that simultaneously optimizes each objective may not exist and therefore, the problem actually is to find the set of Pareto optimal or non dominated solutions. The frozen arc model \[20\] is also assumed where the arc cost is determined at the problem. The frozen arc model \[20\] is also assumed where the arc cost is determined at the arrival event (coincident with the arrival time at the tail of the arc and does not change during its traversal.

Let \( G = (V, A) \) be a directed graph, where \( V \) is the set of nodes and \( A \) is the set of arcs. Let \(|V| = n \) and \(|A| = m \). Each arc \((i, j) \in A\) is associated with three attributes \( c_1(i, j, t) \), \( c_2(i, j, t) \) and \( pt(i, j, t) \), whose values are assumed to be non-negative and may change over time; \( c_1(i, j, t) \) and \( c_2(i, j, t) \) denote the two costs for traversing \((i, j)\) and \( pt(i, j, t) \) denotes the travel time for traversing \((i, j)\), departing from node \( i \) at time instance \( t \). The two costs are the objectives to be minimized, while the travel time is the “resource” constrained in the problem. The frozen arc model \[20\] is also assumed where the arc cost is determined at the arrival time at the tail of the arc and does not change during its traversal.

Let \( C_1(P) \), \( C_2(P) \) denote the total cost of a path \( P \) according to the first and the second criterion respectively, and \( PT(P) \) denote the total travel time along \( P \). Given a start node \( s \in V \), a destination node \( d \in V \), a departure time \( t_{\text{start}} \), an upper bound \( T \) on the maximum permissible total travel time, with no waiting at nodes, the problem is to find a shortest path \( P \) from \( s \) to \( d \) departing at \( t_{\text{start}} \) from \( s \) that minimizes the two objectives \( C_1(P) \) and \( C_2(P) \) without violating the travel time constraint i.e., \( PT(P) \leq T \). Since the objective functions may be conflicting, a single solution that simultaneously optimizes each objective may not exist and therefore, the problem actually is to find the set of Pareto optimal or non dominated solutions. A solution is termed as a non dominated solution if none of the objectives can be improved in value without degrading the value of the other objective. Formally, a path \( p \) is said to dominate another path \( p' \) (\( p \prec p' \)) if \((C_1(p) < C_1(p')) \) and \((C_2(p) \leq C_2(p')) \) or \((C_1(p) \leq C_1(p')) \) and \((C_2(p) < C_2(p')) \).

The main drawback of a backward approach for solving the time-dependent bi-objective shortest path problem is the large set of labels that are initialized at the destination node and then expanded, especially when the upper bound on the total travel time is relatively large. Forward algorithms overcome this problem, as the departure time instances at the source node are usually few in number. Furthermore, since waiting at nodes is not allowed, each arrival event at an intermediate node creates only a single departure event (coincident with the arrival event) and thus the number of departure time instances at nodes is also relatively low. In contrast, in a backward algorithm, due to the large number of labels initially created at the destination node, there are much more possible departure time instances at the nodes of the network and accordingly higher number of generated labels at each node. For this reason, we propose a forward label setting algorithm (Algorithm \[1\]) for finding the set of optimal Pareto paths in time-dependent non-FIFO networks, with no waiting at nodes.

Algorithm \[1\] keeps a set of labels \( l_i(t) \) where \( i \) is a node and \( t \) is a time instance. Each label \( l_i(t) = ((C_1, C_2); j; \text{prev} \_\text{ptr}) \) corresponds to a path from the source node \( s \) to the node \( i \) arriving at \( i \) at time \( t \), and consists of the following components:

- A pair of reals \( C_1, C_2 \) representing the total cost of the estimated path from \( s \) to \( i \) with respect to the two criteria \( c_1, c_2 \).
- An integer \( j \) corresponding to predecessor of \( i \) on the path from \( s \) to \( i \).
- A pointer \( \text{prev} \_\text{ptr} \) pointing to the Pareto optimal label of the predecessor whose extension gave this label.

It is important to note that the FIFO property does not hold for cost functions, that is leaving as much as earlier from a node does not necessarily reduce the cost of outgoing edge. Thus, there is no way to know in advance the ideal time of leaving a node for achieving the optimal cost path to the destination node. As a result, for each node and for each arrival (and hence departure) time instance at this node, we should keep all the non-dominated
labels referring to that particular arrival time. In contrast, in the corresponding static bi-criteria shortest path problem there is no need to partition the labels at each node according to their arrival time and only a single set of non-dominated labels can be kept at each node.

Algorithm 1 iterates over all integer time instances in the interval $[t_{\text{start}}, T + t_{\text{start}}]$, where $t_{\text{start}}$ is the departure time from the source node and $T$ is the maximum permissible total travel time. During the execution, the algorithm maintains two groups of label lists at each node and for each time instance, namely, the permanent and temporary lists. The labels of the temporary list of a node for a given time instance are all transferred one by one to the corresponding permanent list when the algorithm iteration proceeds to that particular instance. However, just before the transfer above, each of these labels is extended, thus creating a new label. Each of these new labels is kept only if it does not violate the total travel time constraint (line 19) and also if it is not dominated by another label belonging to the same node with the same arrival time (lines 22-38). Also, labels already in the temporary list for that arrival instance are compared against this newly generated label and discarded when they are dominated by the new label. The order according to which the labels of the current time instance are extended is not actually important, since all labels of that instance should be extended first before proceeding to the next iteration step. Thus, the label to be extended (pivot label) each time can be selected randomly (line 8). By the end of the execution, all Pareto-optimal solution paths will have been stored as permanent labels of the destination node $d$. Notice also that for the node $d$, no label extension takes place (line 14) since the algorithm does not need to go further down the destination node due to the fact that the cost functions are non-negative valued and hence going further only adds to the total cost of the path. Finally, in Algorithm 1, there is a single departure time instance from the source node but it is trivial to handle the case of multiple possible departure time instances too.

3.2. The Forward Label Setting Algorithm for Both Continuous and Discrete Time

In this subsection we propose a “generalized” forward label setting algorithm (Algorithm 2) which can handle both discrete and continuous time values. It is based on Martin’s algorithm [17] which solves the multi-objective shortest path problem, returning all Pareto-optimal paths from a single source node to all the other nodes of the network.

A common practice of discrete time algorithms [4, 3, 11] is to exhaustively iterate over all possible time instances. However, depending on the granularity of time discretization, this approach may lead to a number of needless iterations because many of the discrete time instances considered in different iterations do not happen as actual arrival times at nodes. By exploiting this fact, we propose Algorithm 2 which considers only the “legitimate” arrival time instances, resulting in considerable reduction in the number of iterations performed. This reduction heavily depends on the number of different values that the travel time function $pt$ can possibly get. Moreover, Algorithm 2 does not require discretized passage time function values, avoiding thus the possible cumulative error caused by the discretization.

The label $l_i = \langle PT; (C_1, C_2); j; \text{prev_ptr} \rangle$ used in Algorithm 2 corresponds to a path from the source node $s$ to the node $i$ and consists of the following components:

- A real value $PT$ corresponding to the total travel time of the path from $s$ to $i$.
- A pair of reals $C_1$, $C_2$ representing the total cost of the path from $s$ to $i$ with respect to the two objectives $c_1$ and $c_2$.
- An integer $j$ corresponding to the predecessor of $i$ on the path from $s$ to $i$.
- A pointer $\text{prev_ptr}$ pointing to the Pareto optimal label of the predecessor whose extension gave this label.

At each iteration of the algorithm, a temporary label is selected. Specifically, this label is the lexicographically minimum label (line 10) considering first the total travel time $PT$ and
Algorithm 1 The Discrete Time Forward Label Setting Algorithm

Require: $G = (V, A)$, and $C$, the cost matrix for all arcs $(i, j) \in A$

Ensure: All Pareto optimal paths from the node $s$ to the node $d$

1: $s$: the start node
2: $d$: the destination node
3: $t_{\text{start}}$: the departure time
4: $T$: the maximum permissible total travel time
5: $pt(i, j, t)$: the travel time for traversing the arc $(i, j)$ departing from the node $i$ at time instance $t$
6: $l_i(t)$: an arrival time instance at the node $j$
7: $L_{\text{temp}}(i, t)$: the list of temporary labels of node $i$ at time instance $t$
8: $L_{\text{perm}}(i, t)$: the list of permanent labels of node $i$ at time instance $t$
9: $\text{card}(l_i(t), L_{\text{perm}}(i, t))$: the position of $l_i(t)$ in the list of permanent labels of node $i$
10: $L^p_i(t)$: the $p$th component of a label $l_i(t)$
11: /* Initialization of temporary and permanent labels of every node and for all $t \in \{t_{\text{start}}, t_{\text{start}} + 1, \ldots, T + t_{\text{start}}\}$ */
2: $L_{\text{temp}}(i, t) \leftarrow \emptyset$, $\forall i \in V, \forall t \in \{t_{\text{start}}, t_{\text{start}} + 1, \ldots, T + t_{\text{start}}\}$
3: $L_{\text{temp}}(s, t_{\text{start}}) \leftarrow \{[0, 0]; \text{null}; \text{null}]\}
4: 5: \textbf{for} $t_{\text{ar}} = t_{\text{start}} \text{ to } T + t_{\text{start}} \text{ do}
6: \quad \textbf{while} ($\bigcup_{i \in V} L_{\text{temp}}(i, t_{\text{ar}}) \neq \emptyset$) do
7: \quad \quad Select a pivot label $l_{i^*}(t_{\text{ar}})$ from $\bigcup_{i \in V} L_{\text{temp}}(i, t_{\text{ar}})$
8: \quad \quad /* Remove $l_{i^*}(t_{\text{ar}})$ from $L_{\text{temp}}(i, t_{\text{ar}})$ and add it to $L_{\text{perm}}(i^*, t_{\text{ar}})$ */
9: \quad \quad $L_{\text{temp}}(i, t_{\text{ar}}) \leftarrow L_{\text{temp}}(i, t_{\text{ar}}) \setminus \{l_{i^*}(t_{\text{ar}})\}$
10: \quad \quad $L_{\text{perm}}(i^*, t_{\text{ar}}) \leftarrow L_{\text{perm}}(i^*, t_{\text{ar}}) \cup \{l_{i^*}(t_{\text{ar}})\}$
11: \quad /* Store the position of label $l_{i^*}(t_{\text{ar}})$ in the list $L_{\text{perm}}(i^*, t_{\text{ar}})$ */
12: \quad \quad $h \leftarrow \text{card}(l_{i^*}(t_{\text{ar}}), L_{\text{perm}}(i^*, t_{\text{ar}}))$
13: \quad \quad \textbf{if} $i^* \neq d$ \textbf{then}
14: \quad \quad \quad /* Label all the successors of $i^*$ */
15: \quad \quad \quad \textbf{for} all $(i^*, j) \in E$ \textbf{do}
16: \quad \quad \quad \quad $t_{\text{ar}}^j \leftarrow t_{\text{ar}} + pt(i^*, j, t_{\text{ar}})$
17: \quad \quad \quad /* Verification that the constraint is fulfilled */
18: \quad \quad \quad \textbf{if} $t_{\text{ar}}^j - t_{\text{start}} \leq T$ \textbf{then}
19: \quad \quad \quad \quad $l_j(t_{\text{ar}}^j) \leftarrow ((l_{i^*}(t_{\text{ar}}) + c_1(i^*, j, t_{\text{ar}}), t_{\text{ar}}^j + c_2(i^*, j, t_{\text{ar}})); i^*; h)$
20: \quad \quad \quad /* Verify that there is no label $l_j(t_{\text{ar}}^j)$ of node $j$ at time instance $t_{\text{ar}}^j$ dominating label $l_j(t_{\text{ar}})$ */
21: \quad \quad \quad \textbf{if} $\forall l_j(t_{\text{ar}}^j) \in \{L_{\text{temp}}(i^*, t_{\text{ar}}) \cup L_{\text{perm}}(i^*, t_{\text{ar}})\}$ and $l_j(t_{\text{ar}}^j) < l_j(t_{\text{ar}})$ \textbf{then}
22: \quad \quad \quad /* Store the label $l_j(t_{\text{ar}}^j)$ of node $j$ at time instance $t_{\text{ar}}^j$ as temporary */
23: \quad \quad \quad $L_{\text{temp}}(i^*, t_{\text{ar}}) \leftarrow L_{\text{temp}}(i^*, t_{\text{ar}}) \cup \{l_j(t_{\text{ar}}^j)\}$
24: \quad \quad \quad /* Delete all temporary labels of node $j$ at time instance $t_{\text{ar}}^j$ dominated by $l_j(t_{\text{ar}})$ */
25: \quad \quad \quad $L_{\text{temp}}(i^*, t_{\text{ar}}) \leftarrow L_{\text{temp}}(i^*, t_{\text{ar}}) \setminus \{l_j(t_{\text{ar}}^j) \in L_{\text{temp}}(i^*, t_{\text{ar}}) \text{ and } l_j(t_{\text{ar}}^j) < l_j(t_{\text{ar}})\}$
26: \quad \quad \quad \textbf{end if}
27: \quad \quad \quad \textbf{end for}
28: \quad \quad \quad \textbf{end if}
29: \quad \quad \quad \textbf{end for}
30: \quad \quad \textbf{end while}
31: \quad \textbf{end for}
then the costs $C_1$ and $C_2$. Thus, all temporary labels are visited in chronological order and this in turn ensures that the currently visited label can be safely transferred to the permanent labels of the node which it belongs to since with respect to the first component (the total travel time), this label will be better than all temporary labels of this node visited at later iterations. In Algorithm 1 for ensuring that the pivot label is actually a permanent label, all possible time instances should be considered exhaustively in chronological order. Now, this is avoided in this algorithm. Also, similarly to Algorithm 1, the currently selected label is extended, generating new labels at the neighbors of the current node. Although, Algorithm 2 employs a single temporary and permanent list at each node, the newly generated label is checked for dominance only with the labels of the node referring to the same arrival time as in Algorithm 1. Notice also that each time a label is inserted into the list of permanent labels of the destination node $d$, the corresponding path is a Pareto-optimal solution and thus at the end of the execution, all Pareto-optimal solutions can be found in this list. Finally, although Algorithm 2 assumes a single departure time at the source node, it can be trivially extended to handle multiple possible departure time instances.

3.3. Speeding up Techniques

Even in the static multi-objective shortest path problems, label setting algorithms might be slow, especially in large networks. In the time-dependent multi-objective shortest path problem, the computation time is further increased due to the fact that for each node and arrival at that node, a set of Pareto-optimal labels should be kept and expanded during the algorithm execution. For reducing the computational overhead, we employ a technique proposed by Demeyer et al. for decreasing the number of generated labels. Specifically, the technique always checks whether a newly generated label after a label extension is dominated by a label belonging to the destination node and in that case the label is discarded since it is not going to give a Pareto-optimal solution.

4. Computational results

In order to compare the efficiency of Algorithms 1 and 2 a number of experiments were conducted. The algorithms were compared against the backward algorithm of Hamacher et al’s which was slightly adjusted to the problem at hand. A Fibonacci heap was also employed in Algorithm 2 for keeping the set of generated labels lexicographically ordered.

The algorithms were tested on the same set of randomly generated graphs. The size of the random graphs in the experiments was set to 500, 750, 1000 and 1500 nodes while the number of ingoing and outgoing arcs was 12, 14, 16, 18 and 20 in different experiments. All the nodes in the graph had the same number of ingoing and outgoing arcs. The single departure time instance from the source node $t_{\text{start}}$ was set to 0. The time-dependent edge costs were randomly selected from the set $\{0, 1, \ldots, 10\}$ for each time instance $t \in \{0, 1, \ldots, T\}$. The maximum permissible total travel time $T$ was set equal to 100 and 500. The CPU time was averaged over 50 different experiments.

Algorithms were implemented in C++. The experiments were performed on a system equipped with an Intel(R) Core(TM)2 Duo CPU P7350 at 2.00GHz processor and 3 GB RAM. The computational results are presented in Tables 1 and 2.

Furthermore, the algorithms were tested on a set of raster grids. Path planning in unstructured environments (e.g. robot path planning or ship routing) make use of a grid structure. The grid resolution was set equal to 1 km, while the size of the raster grids in the experiments was set to 500 and 1000 nodes. The number of ingoing and outgoing arcs of each node was 4, 8 and 16 in different experiments. All the nodes in the graph had the same number of ingoing and outgoing arcs. The single departure time instance from the source node $t_{\text{start}}$ was set to 0. The time-dependent edge costs were randomly selected from the set $\{0, 1, \ldots, 10\}$ for each time instance $t \in \{0, 1, \ldots, T\}$, while the passage time costs was set proportionally...
Algorithm 2 The Generalized Forward Label Setting Algorithm

Require: $G = (V,A)$, and $C$, the cost matrix for all arcs $(i,j) \in A$
Ensure: All Pareto optimal paths from source node $s$ to destination node $d$

1: $s$: start node  
$d$: the destination node  
$t_{start}$: the departure time  
$T$: the maximum permissible total travel time  
$p_t(i,j,t)$: the travel time for traversing the arc $(i,j)$ departing from the node $i$ at time $t$  
$l_i = (PT; (C_1, C_2); j; prev_{-}ptr)$: a label of the node $i$ corresponding to a path from $s$ to $i$  
$L_{temp}_i$: the list of temporary labels of node $i$  
$L_{perm}_i$: the list of permanent labels of node $i$  
card$(l_i, L_{perm}_i)$: the position of label $l_i$ in the list of permanent labels of node $i$  
l$_i^j$: the $j^{th}$ component of the label $l_i$

2: /* All temporary labels across all nodes are lexicographically ordered according first to the passage time $PT$ and then according to $C_1$ and $C_2$ using a priority queue. */

3: /* Initialization */
4: $L_{temp}_i, L_{perm}_i \leftarrow \emptyset, \forall i \in V$
5: $L_{temp}_s \leftarrow \{(0; (0,0); null; null)\}$
6: 
7: /* Iteration */
8: while $(\cup_{i \in V} L_{temp}_i \neq \emptyset)$ do
9: /* Find the minimum lexicographic label */
10: $l_i^* \leftarrow \text{minlex} \{\cup_{i \in V} L_{temp}_i\}$
11: $L_{temp}_i \leftarrow L_{temp}_i \setminus \{l_i^*\}$
12: /* Move the selected label from the temporary list to the permanent list of node $i^*$ */
13: $L_{perm}_i^* \leftarrow L_{perm}_i \cup \{l_i^*\}$
14: /* Store the position of label $l_i^*$ in the list $L_{perm}_i^*$ */
15: $h \leftarrow \text{card}(l_i^*, L_{perm}_i^*)$
16: /* Label all the successors of $i^*$ */
17: if $i^* \neq d$ then
18: /* for all $j \in V$ with $(i^*,j) \in A$ do */
19: /* Compute $l_j^*$, the current label of node $j$ */
20: $l_j^* \leftarrow (l_{i^*}^j + p_t(i^*,j,l_{i^*}^j + t_{start}); (l_{i^*}^j + C_1(i^*,j,l_{i^*}^j + t_{start}), l_{i^*}^j + C_2(i^*,j,l_{i^*}^j + t_{start})); i^*; h)$
21: /* Verify that the time constraint is fulfilled */
22: if $l_j^* \leq T$ then
23: /* Verify that label $l_j^*$ is not dominated by any other label $l'_j$ of node $j$ at the same arrival time */
24: if $\forall l'_j \in \{L_{temp}_j \cup L_{perm}_j\}$ and $l'_j + t_{start} = l_j^* + t_{start}$ and $l'_j < l_j^*$ then
25: /* Store the label $l_j$ of node $j$ as temporary */
26: $L_{temp}_j \leftarrow L_{temp}_j \cup \{l_j\}$
27: /* Delete all temporary labels of node $j$ dominated by $l_j$ at the same time */
28: $L_{temp}_j \leftarrow L_{temp}_j \setminus \{l'_j \in L_{temp}_j \text{ and } l'_j + t_{start} = l_j + t_{start} \text{ and } l_j \prec l'_j\}$
29: end if
30: end if
31: end for
32: end if
33: end while
Table 1. Computational results - departure time $t_{\text{start}} = 0$, maximum travel time $T = 100$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m/n$</th>
<th>Algorithm 1</th>
<th>Algorithm 2</th>
<th>Hamacher’s Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>5.25</td>
<td>23.42</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1.24</td>
<td>9.82</td>
<td>36.4</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>16</td>
<td>22.5</td>
<td>50.86</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>33.42</td>
<td>61.36</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>6.2</td>
<td>42.5</td>
<td>83.2</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1.54</td>
<td>6.4</td>
<td>42.6</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2.24</td>
<td>9.98</td>
<td>63.44</td>
<td></td>
</tr>
<tr>
<td>750</td>
<td>16</td>
<td>23.98</td>
<td>78.9</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>6.94</td>
<td>36.5</td>
<td>93.76</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>7.98</td>
<td>45.36</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2.96</td>
<td>9</td>
<td>53.24</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>4.12</td>
<td>13.82</td>
<td>69.98</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>16</td>
<td>24.56</td>
<td>90.5</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>7.68</td>
<td>39.5</td>
<td>104.76</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>9.2</td>
<td>55.4</td>
<td>135.7</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>8.78</td>
<td>19.4</td>
<td>98.56</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>10.46</td>
<td>29.82</td>
<td>123.22</td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>16</td>
<td>36</td>
<td>141.4</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>18.60</td>
<td>57.74</td>
<td>179.34</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>20.38</td>
<td>67.92</td>
<td>200.86</td>
<td></td>
</tr>
</tbody>
</table>

to the length of the edge. The maximum permissible total travel time $T$ was set equal to 100. The CPU time was averaged over 20 different experiments. The computational results of the experiments conducted in grid structures are presented in Table 3.

In all experiments conducted, Algorithm 1 outperforms both Algorithm 2 and Hamacher et al’s algorithm while its execution time is slightly affected by the number of the nodes and arcs of the random graph and the travel time bound $T$. Algorithm 2 is slower than the discrete time Algorithm 1 but on the other hand, it is a more general algorithm that handles both discrete and continuous travel time instances. The performance of Algorithm 2 seems to be affected by the size of the random graph and its topology. In contrast, the constraint on the total travel time does not seriously affect the algorithm performance. Algorithm 2 is more efficient than Hamacher et al’s algorithm. This fact confirms our claim that a forward algorithm can solve this particular problem more efficiently than a backward algorithm. Hamacher et al’s algorithm iterates through all possible time instances and extends a large set of labels proportional to the maximum permissible total travel time. Therefore, it seems that a backward algorithm lacks the ability of finding Pareto-optimal paths fast when a long total travel time is permitted. In contrast, by considering only actual arrival instances at nodes, Algorithm 2 saves many iterations, especially when the arc travel time function takes on only a small number of different values.

5. Conclusions

In this paper, we proposed algorithms for the time-dependent bi-objective shortest path problem, namely, the problem of point-to-point shortest path with no waiting at nodes, fixed departure time at the source node and a constraint on the total travel time. A new discrete time forward label setting algorithm was proposed which is more efficient than a backward label setting algorithm existing in the literature. Dropping the assumption of discrete travel
time, another forward label setting algorithm was also proposed. Both forward algorithms solve the aforementioned problem much more efficiently, especially when the constraint on the maximum permissible total travel time is relatively loose.

Acknowledgement

This work was carried out in the framework of the project "AMINESS: Analysis of Marine Information for Environmentally Safe Shipping" which was co-financed by the European Fund for Regional Development and from Greek National funds through the operational programs "Competitiveness and Entrepreneurship" and "Regions in Transition" of the National Strategic Reference Framework - Action: "COOPERATION 2011 Partnerships of Production and Research Institutions in Focused Research and Technology Sectors".

References


