Bi-directional Search for Robust Routes in Time-dependent Bi-criteria Road Networks*

Matúš Mihalák[1] and Sandro Montanari[2]

1 Department of Knowledge Engineering, Maastricht University, The Netherlands

2 Department of Computer Science, ETH Zurich, Switzerland

Abstract

Based on time-dependent travel times for N past days, we consider the computation of robust routes according to the min-max relative regret criterion. For this method we seek a path minimizing its maximum weight in any one of the N days, normalized by the weight of an optimum for the respective day. In order to speed-up this computationally demanding approach, we observe that its output belongs to the Pareto front of the network with time-dependent multi-criteria edge weights. We adapt a well-known algorithm for computing Pareto fronts in time-dependent graphs and apply the bi-directional search technique to it. We also show how to parametrize this algorithm by a value K to compute a K-approximate Pareto front. An experimental evaluation for the cases $N = 2$ and $N = 3$ indicates a considerable speed-up of the bi-directional search over the uni-directional.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases shortest path, time-dependent, bi-criteria, bi-directional search, min-max relative regret

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.82

1 Introduction

The standard goal of *route planning* is the computation of a quickest route from a location s to a location t when departing at a time τ. Road networks are modeled as directed graphs with time-dependent edge weights representing travel times at a given period of time and a quickest route corresponds to a shortest s-t path in the time-dependent graph. The edge weights are usually an aggregation (e.g., average) of measured travel times of many individual cars over many similar time points. These aggregated values provide a good estimate of the expected travel time, yet only over a large amount of past days. They say little about deviations of the actual travel times and about per-day nuances of the traffic situations. In fact, a quickest path computed from such aggregated values can perform substantially bad on one particular day. In such situations a more appropriate goal is the computation of a *robust route* [22], that is a path offering guarantees on its travel time in the various situations we can encounter.

In this paper we consider the computation of robust routes in time-dependent road networks. Our focus is on speeding-up a particular method called *min-max relative regret*.

* This work was supported by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities and Sustainability), under grant agreement no. 288094 (project eCOMPASS) and by the Swiss National Science Foundation (SNF) under the grant number 200021 138117/1.

The method looks for a route minimizing the maximum *normalized* travel time in the worst-case scenario. In our case, there are N scenarios (N past days), and the normalization is done with respect to the travel time of a quickest route in the respective day.

In a straightforward implementation, we enumerate all s-t paths in order of increasing travel times, alternatively for each day, until the desired s-t path is found. This implementation is however computationally extremely demanding and impractical. Thus, we observe that its output lies on the Pareto front of the road network with appropriate *time-dependent* and *multi-criteria* edge weights induced by the N instances. We adapt a known algorithm of Hansen/Martins [19, 23] for computing the Pareto front of such a graph and apply the speed-up technique of *bi-directional* search to it. To the best of our knowledge, our paper is the first one to consider speeding-up the computation of shortest paths in a setting where every component of the edge weights is time-dependent *and* multi-criteria.

2 Quickest Paths and Min-max Optimization

We consider a directed graph $G = (V, E)$ with time-dependent edge weights $w : E \times T \to \mathbb{N}$ defined over a given time horizon T. In our road network application $w(e, \tau)$ expresses the travel time needed to traverse the edge $e \in E$ when the vehicle enters e at time $\tau \in T$. This information for an edge $e \in E$ is obtained from a piecewise-linear function $f_e : T \to \mathbb{N}$ with period of one week, represented succinctly by a number of breakpoints. In our data-set the week is divided into 2016 breakpoints, one every 5 minutes. We can therefore compute $w(e, \tau)$ for any $\tau \in T$ in time $O(1)$.

A *path* P is a sequence $\langle v_1, \ldots, v_k\rangle$ of vertices where $(v_i, v_{i+1}) \in E$ for $i = 1, ..., k-1$. We overload the function w to express the travel time of $P = \langle v_1, ..., v_k\rangle$ departing at $\tau \in T$ as

$$w(P,\tau) = \begin{cases} 0 & \text{if } k = 1 \\ w\left((v_1, v_2), \tau\right) + w\left(\langle v_2, \ldots, v_k\rangle, \tau + w\left((v_1, v_2), \tau\right)\right) & \text{otherwise.} \end{cases}$$

A *quickest path* for given source $s \in V$, target $t \in V$, and time $\tau \in T$, is an s-t-path P minimizing $w(P, \tau)$. Note that in the above definition we do not allow waiting at vertices; this is clearly not beneficial if waiting at a vertex on an s-t path does not result in arriving at t earlier, a property formally defined as follows.

▶ **Definition 1** (FIFO Property). A weight function $w : E \times T \to \mathbb{N}$ satisfies the *FIFO property* if for all $e \in E$ and all $\tau, \tau' \in T$ with $\tau \le \tau'$ it holds $\tau + w(e, \tau) \le \tau' + w(e, \tau')$.

If the edge weights satisfy the FIFO property, a quickest path can be computed efficiently using a generalization of Dijkstra's algorithm [11]. Since road networks are known to satisfy this property, we therefore focus on algorithms not waiting at nodes.

Given a departure time $\tau \in T$, vertices $s, t \in V$, and a finite discrete set of *scenarios*, the problem we consider is that of computing an optimum s-t path according to the *min-max relative regret* criterion. A scenario or instance I_i is specified by an edge weight function $w_i : E \times T \to \mathbb{N}$. The *relative regret* of an s-t path P in instance I_i is the ratio between its weight and the weight OPT_i of a quickest path in I_i. Given N different instances $I_1, \ldots, I_N$, we want to compute a path minimizing its maximum relative regret. In other words, we look for a path

$$\arg\min_P \max_i \left\{ \frac{w_i(P, \tau)}{OPT_i} \right\}. \tag{1}$$

The motivation for the study of this problem comes from its relation to robust optimization. For example, Buhmann et al. [4] define a general framework for robust optimization according to several different criteria. Among these criteria, the one denoted by the authors as "First intersection" corresponds to the min-max relative regret.

Problem (1) is similar in nature to the problems of optimizing according to the *min-max absolute* or the *min-max deviation* criteria [25], the difference lying only in the normalization factor. In the former we look for a path $\arg\min_P \max_i \{w_i(P,\tau)\}$, while in the latter the goal is $\arg\min_P \max_i \{w_i(P,\tau) - OPT_i\}$. All these problems are NP-hard even for 2 instances; hardness for the min-max absolute and the min-max deviation criteria was proven by Yu and Yang [25], while for the min-max relative regret it follows straightforwardly as a reduction from the constrained shortest path problem [13].

Since a worst-case efficient algorithm is unlikely to be found, we consider efficiency only from a practical perspective. Even though our theoretical results can be generalized for any number N of scenarios, for practical reasons we focus on the particular case $N = 2$. As additional motivation for considering a small number of scenarios, we observe that approaches based on the min-max or min-max (relative) regret criteria are of more interest in these cases. If the number of scenarios is large, approaches based on statistical analysis are typically more efficient and produce results of similar if not better quality. In Section 3 we prove that there always exists an optimum solution lying on the Pareto front of all s-t paths. In Section 5 we design a bi-directional algorithm for computing such a front that can be parametrized by a factor K in order to compute a K-approximate solution. In Section 6 we experimentally show that, on a road network of the Berlin and Brandenburg area, the speed-up of the bi-directional search over uni-directional is considerable. A preliminary evaluation of the algorithm for the case $N = 3$ seems to indicate that this speed-up scales with the number of instances. In Section 7 we propose a simple modification to the bi-directional algorithm exploiting the correlation between the travel times of the different instances.

3 Relation to Bi-criteria Quickest Paths

For the case of 2 scenarios, the instances I_1 and I_2 with edge-weight functions respectively w_1 and w_2 induce a bi-criteria weight function $w : E \times T \to \mathbb{N}^2$ defined as

$$w(e,\tau) = \begin{pmatrix} w_1(e,\tau) \\ w_2(e,\tau) \end{pmatrix}. \tag{2}$$

Again, we overload the definition of w to express the weight of a path P as

$$w(P,\tau) = w(P',\tau) + \begin{pmatrix} w_1(e,\tau + w_1(P',\tau)) \\ w_2(e,\tau + w_2(P',\tau)) \end{pmatrix}, \tag{3}$$

where P' is the path obtained from P without the last hop e. Given two s-t paths P and P', we say that P *dominates* P' if $w_i(P,\tau) \leq w_i(P',\tau)$ for all $i \in \{1,2\}$, with the inequality being strict for some i. If two paths have the same weight in both components, they are said to be *equivalent*. The *Pareto front* of a set of paths $\mathcal{P}$ is the subset of all paths in $\mathcal{P}$ that are not dominated by another path in $\mathcal{P}$. For the sake of readability, in the following we will assume that no two paths are equivalent. It is well known [1] that an optimum path for Problem (1) lies in the Pareto front $\mathcal{F}$ of all s-t paths departing at $\tau \in T$. The following theorem proves a slightly stronger statement.

▶ **Theorem 2.** *Let $\mathcal{F}_\rho$ be the Pareto front of all optimum paths of Problem (1). Then, $\mathcal{F}_\rho \subseteq \mathcal{F}$.*

Proof. Assume towards contradiction that there exists a path $P \in \mathcal{F}_\rho \setminus \mathcal{F}$. Then, there is a path $P' \notin \mathcal{F}_\rho$ dominating P. For $i \in \{1, 2\}$, we let

$$\rho_i' = \frac{w_i(P', \tau)}{OPT_i} \leq \frac{w_i(P, \tau)}{OPT_i} = \rho_i,$$

Note that the relative regret of an optimum path is $\rho^* = \max\{\rho_1, \rho_2\}$ and that $\max\{\rho_1', \rho_2'\} > \rho^*$. If $\max\{\rho_1', \rho_2'\} = \rho_i'$ for some $i \in \{1, 2\}$ we get a contradiction, because

$$\rho_i' \leq \frac{w_i(P, \tau)}{OPT_i} \leq \rho^* < \rho_i'. \qquad \blacktriangleleft$$

Theorem 2 implies that an optimum path for Problem (1) can be computed by enumerating all paths in $\mathcal{F}$ and picking one with smallest relative regret. Note that there may exist paths in $\mathcal{F}$ that are not optima, typically the quickest paths in either of the two instances. It is straightforward to prove Theorem 2 also for the min-max absolute and the min-max deviation criteria, implying that the bi-directional search algorithm proposed in the second half of this paper can be applied for those criteria as well. We further observe that the paths in $\mathcal{F}_\rho$ might not be extreme points of the convex hull of $\mathcal{F}$. This observation rules out the possibility of adopting known algorithms for the computation of such points [6, 12, 14].

▶ Remark. The definition in eq. (3) might appear unusual to a reader familiar with bi-criteria quickest path problems. In the literature it is more typically assumed that one of the two criteria of the weight of a path is its travel time while the other one is a cost depending on the travel time (for example, fuel consumption). Such a weight function can be written as

$$w(P, \tau) = w(P', \tau) + \begin{pmatrix} w_1(e, \tau + w_1(P', \tau)) \\ w_2(e, \tau + w_1(P', \tau)) \end{pmatrix}. \tag{4}$$

Note the difference in the time at which the second component is evaluated. Since our target application is robust routing, we however need to consider different travel times for the same path and hence use the definition in eq. (3). Under similar assumptions on the FIFO property of the edge weights, our results can be generalized for eq. (4) as well.

4 Related Work

We now consider the computation of an optimum path for Problem (1) by means of time-dependent multi-criteria optimization. Our aim is to apply the speed-up technique *bi-directional search* to an algorithm by Martins for computing Pareto fronts and experimentally investigate the improvements to its running time on road networks. In spite of its relevance, the literature about the problem is scant, and not many practical algorithms are known. The most closely related work is by Batz and Sanders [3] that consider the computation of shortest paths in a graph with multi-criteria edge weights where only one of the components is time-dependent. A great amount of work has been however invested by the community into the speed-up of routing algorithms in settings where edge weights are either only time-dependent [2, 7, 21] or only multi-criteria [8, 10].

Hansen [19] introduced several variants of bi-criteria shortest path problems and a pseudo-polynomial time algorithm computing the Pareto front of a graph with static non-negative bi-criteria edge weights. Martins [23] generalized this algorithm to static edge weights with more than two criteria. His algorithm keeps a priority queue of temporary labels Q and a set of permanent labels π_u for every vertex $u \in V$. Each label $(u, \boldsymbol{\omega})$ represents a path from s to u with weight $\boldsymbol{\omega} \in \mathbb{N}^k$ (for k criteria); we write $P \in \pi_u$ to indicate that the label

Listing 1 Time-dependent Martins' algorithm.

```
$\forall v \in V : \pi_v := \emptyset$
$Q$.insert($s, \binom{0}{0}$)
{Compute front}
while $Q \neq \emptyset$ do
 $(u, \boldsymbol{\omega}) := Q$.extract_min()
 for $e = (u, v) \in E$ do
  $\boldsymbol{\nu} := (v, \boldsymbol{\omega} + \binom{w_1(e, \tau+\omega_1)}{w_2(e, \tau+\omega_2)})$
  if $\neg\pi_v$.dominates($\boldsymbol{\nu}$) and $\neg\pi_t$.dominates($\boldsymbol{\nu}$) then $Q$.insert($\boldsymbol{\nu}$)
```

representing P is in π_u. At the beginning every π_u is empty, and a label $(s, \mathbf{0})$ is created and put into Q. At each iteration the algorithm extracts from Q the smallest label $(u, \boldsymbol{\omega})$ in lexicographical order and puts it into π_u. A new label $(v, \boldsymbol{\nu})$ is then generated for each vertex v that can be reached from u, with $\boldsymbol{\nu} = \boldsymbol{\omega} + w(u, v)$. If no label in π_v or π_t dominates the new one, it is inserted into Q and all labels corresponding to s-v paths that are dominated by $(v, \boldsymbol{\nu})$ are removed from Q. The algorithm ends when Q is empty; at this point, π_t contains labels representing all paths in the Pareto front $\mathcal{F}$. By storing labels in Q and in all π_v in lexicographical order, we can implement the operations of extract minimum, insertion, and dominance checking to run, for the bi-criteria case, in logarithmic time. For a number of criteria larger than 2 it is currently not known how to efficiently implement these operations.

Gräbener et al. [17] provide an experimental evaluation of a straightforward time-dependent extension of Martins' algorithm, shown in Listing 1, on some publicly accessible networks. The correctness of this extension crucially depends on the FIFO property. Hamacher et al. [18] consider the setting where the FIFO property does not hold, and provide algorithms computing the Pareto front for a given s-t pair as well as for the all-to-all variant.

Dijkstra's algorithm for finding shortest s-t paths in the static single-criteria case gradually grows a shortest-path tree from s. At any step, each vertex is in one of the following states: UNREACHED, SETTLED, or DISCOVERED. A vertex is UNREACHED if its distance from s is not known, it is SETTLED if its distance from s is known exactly, and it is DISCOVERED if only an upper bound on the distance is known. At every iteration the algorithm introduces a new edge in the shortest path tree and sets its tail vertex as SETTLED. The algorithm terminates when t is SETTLED. In the worst-case the tree contains all vertices, even though we are only interested in those on the shortest s-t path that is returned.

The idea behind the *bi-directional search* [15, 16] is to grow two trees rooted at s and t using Dijkstra's algorithm alternatively from s and from t. The execution from t, called *backward search*, uses the edges of the reverse graph, i.e., the graph containing the edges of the original one in reverse direction. As soon as a vertex v is SETTLED by both the forward and the backward search the algorithm terminates and a shortest s-t path is guaranteed to lie in the union of the so-far constructed shortest path trees (such a path might however not pass through v). Any alternation works correctly; a typical choice is to balance the number of iterations of the two searches.

For static multi-criteria edge weights one can apply the bi-directional search by replacing Dijkstra's algorithm with Martins'. Since the goal is to compute the whole Pareto front of s-t paths (and not only a single path), the stopping criterion is however different. Demeyer et al. [9] show that terminating the computation when the sum of the *point-wise minima* of the forward and backward queues is dominated by the front computed so far ensures that the Pareto front is found. The *point-wise minimum* of a queue Q, denoted as Q.p_min(), is the vector where each component is equal to the minimum among all labels in Q for the corresponding criterion.

When the edge weights are time-dependent, even in the single-criterion case, applying the bi-directional search is not straightforward anymore: the input consists of s, t, and the departure time τ. Thus, we can grow a tree from s starting at time τ, but we do not know the time τ' from which we shall start growing the tree from t – ideally, τ' is the earliest arrival time at t, but that is the number we wish to compute. A way to overcome this difficulty is to make the backward search static: for each edge (v, u) of the reverse graph $\overleftarrow{G} = (V, \overleftarrow{E})$, use a static weight defined as

$$\overleftarrow{w}((v,u)) = \min_{\tau \in T} \{w((u,v), \tau)\}. \tag{5}$$

Nannicini et al. [24] propose a bi-directional algorithm using the weights in eq. (5) working in three phases. In phase 1 the forward and backward search run alternatively until a vertex is DISCOVERED in both directions, resulting in an upper bound μ on the weight of a quickest path. In phase 2 both searches continue until the distances of all the DISCOVERED vertices in the backward queue are at least μ. In phase 3 only the forward search continues, with the constraint that only vertices that were SETTLED by the backward search are considered. In the following we show how to apply this idea to the time-dependent multi-criteria case.

5 Bi-directional Time-dependent Martins' Algorithm

A bi-directional algorithm for edge weights that are both time-dependent *and* bi-criteria can be designed by straightforwardly combining the ideas of Demeyer et al. and of Nannicini et al. This results in a three-phases search using Martins' algorithm both from s and from t, where the edge weights in the reverse graph are defined as in eq. (5) for both criteria. The termination condition of the backward search (i.e., the end of phase 2) is the stopping condition of Demeyer et al. As it turns out, however, this trivial algorithm can be improved considerably.

A critical observation to improve the straightforward algorithm is to note that in the backward direction our only interest is to identify vertices that might be on a Pareto optimal path. In other words, to determine whether or not the Pareto front of a given vertex contains at least one "promising" label. However, a label that is good for one criterion might not be good for the other one and we cannot know in advance which labels are promising. Our solution is to consider only the pointwise minima of the Pareto front π_v of each vertex v.

If the only purpose of the backward search is to compute pointwise minima, then Martins' algorithm is more than what is necessary. We can instead implement the backward search as two independent Dijkstra's runs on the reverse graph for each criterion. We modify the three phases of the bi-directional algorithm according to this observation as follows.

For phase 2, suppose we have found (in some way during phase 1) a number of non-necessarily Pareto-optimal s-t paths, and let M denote the Pareto front of these paths, while $\overrightarrow{Q}$ is the forward queue and $\overleftarrow{Q}_1, \overleftarrow{Q}_2$ are the backward queues. Suppose further that at some point during the computation, the weight of a path in M dominates

$$\beta := \overrightarrow{Q}.\text{p\_min}() + \begin{pmatrix} \overleftarrow{Q}_1.\min() \\ \overleftarrow{Q}_2.\min() \end{pmatrix}.$$

At this point, if a vertex v has not been SETTLED by both backward searches, then the weight of any s-t path through v is dominated by β and therefore by a path in M (we prove the correctness of this argument formally in the following). We can thus terminate phase 2 and the backward searches as soon as a path in M dominates β.

For phase 3 consider the situation where the forward search created a label $(v, \boldsymbol{\omega})$ to insert into $\overrightarrow{Q}$. Let the vector of distances computed by the backward searches for v be $v.\boldsymbol{d}$, the value of the second term of β (the minima of the backward queues) at the end of phase 2 be $\overleftarrow{\beta}$, and the minimum between $v.\boldsymbol{d}$ and $\overleftarrow{\beta}$ in each component be $\boldsymbol{\theta}$; that is, for $i \in \{1, 2\}$, we define $\theta_i := \min\{v.d_i, \overleftarrow{\beta_i}\}$. At the beginning of the computation $v.d_i$ is set to ∞ and at the end it holds that $v.d_i \leq \overleftarrow{\beta_i}$ if v has been SETTLED by the i-th backward search. If in phase 3 a path in M dominates $\boldsymbol{\omega} + \boldsymbol{\theta}$ then no path with the same s-v prefix as $\boldsymbol{\omega}$ can be optimal. We can thus discard all labels $(v, \boldsymbol{\omega})$ for which $\boldsymbol{\omega} + \boldsymbol{\theta}$ is dominated by a path in M.

According to the above phases, the purpose of phase 1 is the computation of a suitable tentative front M. Intuitively, a tentative front is good if the domination of β happens as early as possible, because less labels carry over to phase 3. We propose to terminate phase 1 as soon as a vertex v with $\pi_v \neq \emptyset$ is DISCOVERED by both backward searches and set M as the corresponding set of s-t paths passing through v. This strategy has the advantage of being efficient while at the same time being simple to implement. We note however that it is easy to come up with different strategies; it is an interesting open question to identify an optimum one.

To summarize, the three phases of the bi-directional algorithm are in detail as follows:

Phase 1 We let the forward and the backward search run alternatively. In the forward direction we use the time-dependent Martins' algorithm. In the backward direction we use two independent runs of Dijkstra's algorithm, one per criterion, using edge weights as in eq. (5). This phase ends as soon as a vertex v with $\pi_v \neq \emptyset$ is DISCOVERED by both backward searches. At the termination of the phase we let M be the Pareto front of the s-v paths in π_v concatenated with the v-t paths discovered in the backward direction.

Phase 2 Both the forward and the backward searches continue to run as in phase 1, until a path in M dominates β.

Phase 3 Only the forward search continues, with the constraint that labels $(v, \boldsymbol{\omega})$ for which $\boldsymbol{\omega} + \boldsymbol{\theta}$ is dominated by a path in M are ignored. This phase terminates when $\overrightarrow{Q}$ becomes empty.

The pseudocode of the algorithm under the name of BiTdMartins is illustrated by Listing 2. We use ϕ to denote the current phase and $\leftrightarrow$ to denote either the forward ($\leftrightarrow = \rightarrow$) or the backward search ($\leftrightarrow = \leftarrow$). The command $\leftrightarrow \in \{\rightarrow, \leftarrow\}$ selects the direction for the current iteration according to the alternation strategy; in our implementation we alternate between one iteration of the forward search and one iteration for each backward search. Note that to check the termination condition of phase 1 it is not necessary to search through all vertices. It is sufficient to check whether the condition holds only for the vertex extracted at the current iteration.

In the algorithm of Nannicini et al. [24] phase 2 terminates when the upper bound μ computed in phase 1 is at most the minimum of the backward queue. The authors proved that replacing this condition with one that, for a fixed parameter K, checks whether μ is at most K times the minimum of the backward queue results in an algorithm computing a K-approximate quickest path (i.e., a path with weight at most K times the weight of a quickest path). The following theorem shows that BiTdMartins satisfies a similar property. A corollary of this theorem, obtained by setting $K = 1$, implies correctness of the algorithm in the exact variant.

Listing 2 Algorithm BiTdMartins.

```
$M := \emptyset$, $\phi := 1$,  $\forall v \in V : \pi_v := \emptyset$, $v.\boldsymbol{d} := \binom{\infty}{\infty}$
$\overrightarrow{Q}$.insert($s, \binom{0}{0}$), $\overleftarrow{Q}_1$.insert($t, 0$), $\overleftarrow{Q}_2$.insert($t, 0$)
while $\overrightarrow{Q} \neq \emptyset$ do
 if $\phi = 3$ then $\leftrightarrow := \rightarrow$ else $\leftrightarrow \in \{\rightarrow, \leftarrow\}$
 {terminate phase 1}
 if $\phi = 1$ and $\exists v \in V : \pi_v \neq \emptyset$ and $v.d_1 \neq \infty$ and $v.d_2 \neq \infty$ then
   $M$.insert(< $s$-$t$ paths through $v$ >), $\phi := 2$
 {terminate phase 2}
 if $\phi = 2$ and $M$.dominates($\beta$) then $\phi := 3$
 {relax edges}
 if $\leftrightarrow = \rightarrow$ then
  $(u, \boldsymbol{\omega}) := \overrightarrow{Q}$.extract_min()
  for $e = (u, v) \in \overrightarrow{E}$ do
   $\boldsymbol{\nu} := \boldsymbol{\omega} + \binom{w_1(e, \tau + \omega_1)}{w_2(e, \tau + \omega_2)}$
   if $\phi = 3$ and $M$.dominates($\boldsymbol{\nu} + \boldsymbol{\theta}$) then continue
   if $\neg\pi_v$.dominates($\boldsymbol{\nu}$) and $\neg\pi_t$.dominates($\boldsymbol{\nu}$) then $\overrightarrow{Q}$.insert($v, \boldsymbol{\nu}$)
 else
  for $i \in \{1, 2\}$ do
   $u_i := \overleftarrow{Q}_i$.extract_min()
   for $e = (u_i, v) \in \overleftarrow{E}$ do
    if $u_i.d_i + \overleftarrow{w}_i(e) < v.d_i$ then $\overleftarrow{Q}_i$.insert($v, u_i.d_i + \overleftarrow{w}_i(e)$)
```

▶ **Definition 3.** Given $\tau \in T$ and $K \geq 1$, we say that a path P is a K-*approximation* of another path P' if $w(P, \tau)$ dominates or is equivalent to $K \cdot w(P', \tau)$. Given two sets of paths $\mathcal{P}$ and $\mathcal{P}'$, we say that $\mathcal{P}$ is a K-*approximation* of $\mathcal{P}'$ if every path in $\mathcal{P}'$ is K-approximated by a path in $\mathcal{P}$.

▶ **Theorem 4.** *Given $K \geq 1$, if we replace the condition to terminate phase 2 with M.dominates($K \cdot \beta$) and in phase 3 we discard all labels $(v, \boldsymbol{\nu})$ such that M.dominates($K \cdot (\boldsymbol{\nu} + \boldsymbol{\theta})$), then* BiTdMartins *computes a K-approximation of $\mathcal{F}$.*

Proof. Assume there exists a path $P \in \mathcal{F}$ not K-approximated by a path in π_t. That is, for every $P' \in \pi_t$ there is $i \in \{1, 2\}$ such that

$$K \cdot w_i(P, \tau) < w_i(P', \tau). \tag{6}$$

Let P_{sv} be the prefix of P from s to the first vertex v such that $P_{sv} \notin \pi_v$, and P_{vt} be the suffix of P from v to t. Since $P_{sv} \notin \pi_v$, there is a path in M dominating $K \cdot (w(P_{sv}, \tau) + \boldsymbol{\theta})$. Let P' be either this path, if it belongs to π_t, or a path in π_t dominating it otherwise. Note that for all $i \in \{1, 2\}$ it holds $\theta_i \leq \overleftarrow{w}_i(P_{vt})$ since, if v was settled by the i-th backward search, then $\theta_i = v.d_i = \overleftarrow{w}_i(P_{vt})$ while, if v was not settled by the i-th backward search, then $\theta_i = \overleftarrow{\beta}_i \leq \overleftarrow{w}(P_{vt})$. Supposing without loss of generality that eq. (6) holds for $i = 1$ we obtain a contradiction, because

$$K \cdot w_1(P, \tau) < w_1(P', \tau) \leq K \cdot (w_1(P_{sv}, \tau) + \theta_i) \leq K \cdot (w_1(P_{sv}, \tau) + \overleftarrow{w}_1(P_{vt})) \leq K \cdot w_1(P, \tau). \blacktriangleleft$$

Table 1 Run-time in milliseconds and average number of scanned labels.

			Labels		
	Max Rel Regret	Run-time (ms)	Phase 1	Phase 2	Phase 3
Dijkstra	1.0711	261	–	–	220,620
Naive	1.0074	–	–	–	8,420
Uni-dir	1.0074	3,105	–	–	1,419,524
Bi-dir	1.0074	1,888	178,855	189,336	449,560
$K = 1.02$	1.0085	1,570	178,855	177,133	402,970
$K = 1.04$	1.0108	1,427	178,855	165,209	356,479
$K = 1.06$	1.0156	1,286	178,855	153,549	311,216
$K = 1.08$	1.0232	1,150	178,855	142,190	268,139
$K = 1.10$	1.0337	1,028	178,855	131,160	228,646
$K = 1.20$	1.0724	712	178,855	80,678	93,660
$K = 1.40$	1.0830	338	178,855	16,854	10,420
$K = 1.60$	1.0856	275	178,855	1,858	2,407
$K = 1.80$	1.0865	269	178,855	270	1,787
$K = 2.00$	1.0868	269	178,855	81	1,692

▶ **Corollary 5.** BiTdMartins *computes the Pareto front $\mathcal{F}$.*

Note that the converse of Theorem 4 in general does not hold. There might be paths in π_t not approximating a Pareto optimal path.

6 Computational Results

The experimental evaluation was performed on one core of an Intel Xeon E5-2697v2 processor clocked at 2.7 GHz and 64 GB main memory. The code was written in C++ and compiled using GNU C++ compiler version 4.8.2 and optimization level 3.

6.1 Input Road Network

The input data consists of a road network of the area around Berlin and Brandenburg kindly provided by TomTom within the project eCOMPASS [5]. The largest strongly connected component of the graph consists of 443,365 vertices and 1,038,284 edges. The travel times of 750,544 edges are constant, while for the remaining 287,740 edges are given by a piecewise-linear function with period of one week.

To obtain two instances (edge weight functions) I_1 and I_2 we consider departure times τ_1 and τ_2 in two consecutive days. We select uniformly at random one of the 24 hours of a day and let τ_1 be the corresponding point in time on Tuesday and τ_2 be the same time in the following Wednesday. The edge weight functions are obtained by setting the beginnings of the time horizon (in other words the departure times) of I_1 and I_2 respectively at τ_1 and at τ_2. We select 10,000 pairs of vertices s and t uniformly at random.

6.2 Results

Table 1 shows a comparison of the algorithms considered in terms of quality (i.e., the maximum relative regret of the computed path) and efficiency, averaged among the performed 10,000 tests. The efficiency of an algorithm is measured in terms of CPU time and the number of labels scanned for each phase of the algorithm. The number of labels scanned, i.e.,

Table 2 Run-time and number of scanned labels for 3 instances.

			Labels		
	Max Rel Regret	Run-time (ms)	Phase 1	Phase 2	Phase 3
Uni-dir	1.0328	954,267	-	-	7,927,858
Bi-dir	1.0328	487,993	210,374	212,609	3,678,017
$K = 1.2$	1.0861	190,960	210,374	96,108	1,161,266
$K = 1.4$	1.1013	53,154	210,374	23,114	339,865
$K = 1.6$	1.1068	6,180	210,374	4,209	76,958
$K = 1.8$	1.1095	1,670	210,374	1,159	16,383
$K = 2.0$	1.1095	975	210,374	197	2,805

the overall number of labels extracted from the forward and from the backward queues, represents a machine-independent measure of efficiency. The counter of labels scanned for the bi-directional algorithm is increased by one for each iteration of the forward search, and by 0.5 for each iteration of one of the two backward searches.

The algorithms considered for comparison are: the unidirectional search using the time-dependent implementation of Martins' algorithm, the bi-directional search of BiTdMartins, the K-approximate BiTdMartins for different values of K, and the naive algorithm for the min-max relative regret problem. As additional reference, the table also shows information on the computation of a quickest path in I_1 using the time-dependent Dijkstra's algorithm.

The naive algorithm enumerates all s-t paths alternatively for I_1 and I_2 until an optimum path is found; it is implemented as a straightforward time-dependent generalization of an algorithm by Hershberger et al. [20] for the computation of the k-shortest paths. The row corresponding to this algorithm in Table 1 does not show the number of labels scanned. Instead, we display the average number of iterations before finding a path in the intersection. Since each iteration consists of several (a number linear in n) repetitions of the time-dependent Dijkstra's algorithm, we can have an idea on the number of labels scanned by looking at the first two rows of the table.

The improvements of the bi-directional search over the uni-directional is considerable both for the run-time and for the number of scanned labels. The efficiency further increases if we allow an approximation factor K greater than 1. It appears however that there is a limit to the speed-up that can be obtained via approximation. The reason for this limit is that large values of K greatly reduce the amount of time spent by the algorithm in phases 2 and 3, but do not decrease the time in phase 1. In particular, for some value of K, say K^*, the algorithm spends no time at all in phase 2 because the termination condition is met as soon as the phase begins. All values of K greater than K^* will therefore result in similar run-time and number of scanned labels.

6.3 Results for 3 Instances

Table 2 shows experimental results for the case of 3 instances (the third day being Thursday). Since in this case the operations of extract minimum, insertion and dominance checking necessary to implement Martins' algorithm cannot be implemented efficiently, all algorithms are as a result much slower than in the previous case. For this reason, the experimental evaluation is not as thorough, and only 240 pairs of s, t vertices were considered.

We can see that the speed-up of the bi-directional search over the uni-directional is still considerable, and an even more remarkable speed-up can be obtained via approximation. By setting an approximation factor of $K = 2.0$, computations that in the exact case require in average 25 minutes to terminate can be performed in less than one second.

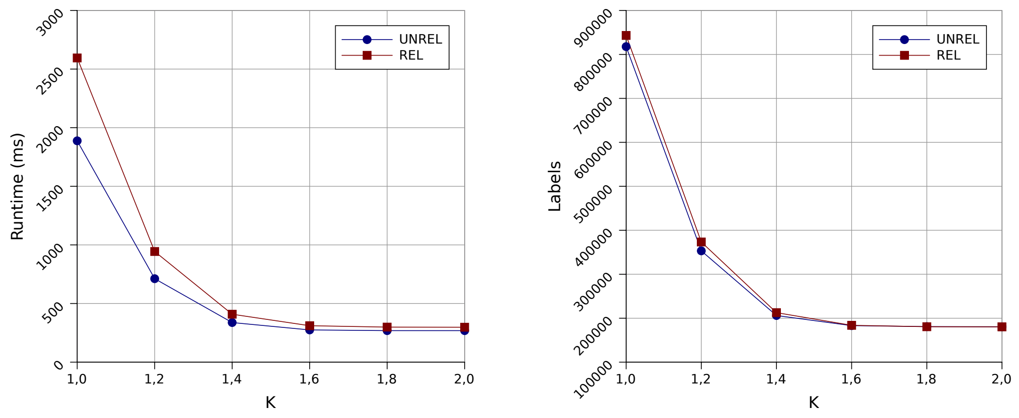

Figure 1 Comparison of bi-directional algorithms.

7 Single Backward Search

In our input data there is a strong correlation between the weights of a path in the different instances, since they represent travel times in very similar time periods. However, this correlation is not explicitly exploited by our algorithm. One might ask for a way to improve the efficiency of BiTdMartins by considering this feature more directly. For example, for the case of 2 instances we might get some improvement by replacing the two backward searches with a single one that, for each backward edge $e \in \overleftarrow{E}$, considers weights of the kind

$$\overleftarrow{w}(e) = \min_{i\in\{1,2\}} \{\overleftarrow{w_i}(e)\}\,.$$

The correctness of this algorithm and its approximated variants follows trivially from the previous proofs under the same assumptions as for BiTdMartin. The benefits of this new algorithm over the original one are however not trivial to estimate. On the one hand, if

$$\max_{e\in\overleftarrow{E}} \{|\overleftarrow{w_1}(e) - \overleftarrow{w_2}(e)|\} \tag{7}$$

is small, the modified backward search will settle almost the same vertices as before, with almost the same values, at the price of one execution of Dijkstra's algorithm instead of two. On the other hand, the lower bounds on the distances computed in the reverse graph are less accurate. As a result, the number of labels scanned by the modified algorithm is larger. The benefit of the modified algorithm is, in loose terms, inversely proportional to eq. (7).

Figure 1 shows a plot of the average run-time and the number of labels settled by the original bi-directional algorithm (unrel) and the modified one (rel) for different values K of approximation and for the same 10,000 s-t pairs. We can see that unrel is faster but it indeed settles more labels than rel. However, the difference between the two is very small and further decreases for increasing values of K until the point where the performance of the two algorithms is almost equal. It is an interesting open question to identify cases where the benefit of a single backward search takes over both the run-time and the number of labels.

8 Conclusions

We have considered the problem of computing an optimum path according to the min-max relative regret criterion and shown that there always exists one such path on the Pareto front of a multi-criteria weight function. We have therefore engineered a bi-directional algorithm for the computation of Pareto fronts in time-dependent multi-criteria graphs and experimentally demonstrated a considerable speed-up compared to the uni-directional variant.

We observe that the presented results appear of practical interest for the application of robust routing. A peculiarity of this application is that the considered criteria correspond to travel times for different days of the week. If the days considered are correlated like, for example, working days as opposed to working days and holidays, we expect this correlation to somehow appear in the travel times as well. As a result, the number of paths in the Pareto fronts is not too big; for our experiments, the average size of the fronts is 8 (although for the case of 3 instances this number increases to 40). It is an interesting open question to assess the practical efficiency of the proposed algorithms for multi-criteria edge weights inducing fronts of larger size, such as those considered by Delling and Wagner [8]. Furthermore, an assessment of the robustness of the routes computed using the min-max relative regret criterion on the Berlin and Brandenburg data-set is planned for a follow-up paper.

References

1 H. Aissi, C. Bazgan, and D. Vanderpooten. Min-max and min-max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427–438, 2009.

2 G. V. Batz, R. Geisberger, P. Sanders, and C. Vetter. Minimum time-dependent travel times with contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 18, 2013.

3 G. V. Batz and P. Sanders. Time-dependent route planning with generalized objective functions. In *ESA*, pages 169–180, 2012.

4 J. M. Buhmann, M. Mihalák, R. Šrámek, and P. Widmayer. Robust optimization in the presence of uncertainty. In *ITCS*, pages 505–514, 2013.

5 European Commission. eCOMPASS Project. `http://www.ecompass-project.eu/`, 2011-2014.

6 C. Daskalakis, I. Diakonikolas, and M. Yannakakis. How good is the chord algorithm? *CoRR*, abs/1309.7084, 2013.

7 D. Delling. Time-dependent SHARC-routing. *Algorithmica*, 60(1):60–94, 2011.

8 D. Delling and D. Wagner. Pareto paths with SHARC. In *SEA*, pages 125–136, 2009.

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

10 S. Erb, M. Kobitzsch, and P. Sanders. Parallel bi-objective shortest paths using weight-balanced B-trees with bulk updates. In *SEA*, pages 111–122, 2014.

11 L. Foschini, J. Hershberger, and S. Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014.

12 S. Funke and S. Storandt. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In *ALENEX*, pages 41–54, 2013.

13 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

14 R. Geisberger, M. Kobitzsch, and P. Sanders. Route planning with flexible objective functions. In *ALENEX*, pages 124–137, 2010.

15 A. V. Goldberg and C. Harrelson. Computing the shortest path: A^* search meets graph theory. In *SODA*, pages 156–165, 2005.

16 A. V. Goldberg and R. F. Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX*, pages 26–40, 2005.

17 T. Gräbener, A. Berro, and Y. Duthen. Time dependent multiobjective best path for multimodal urban routing. *Electronic Notes in Discrete Mathematics*, 36, 2010.

18 H. W. Hamacher, S. Ruzika, and S. A. Tjandra. Algorithms for time-dependent bicriteria shortest path problems. *Discrete Optimization*, 3(3):238–254, 2006.

19 P. Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making Theory and Application*, pages 109–127. Springer Berlin Heidelberg, 1980.

20 J. Hershberger, M. Maxel, and S. Suri. Finding the k shortest simple paths: A new algorithm and its implementation. *ACM Transactions on Algorithms*, 3(4), 2007.

21 S. C. Kontogiannis and C. D. Zaroliagis. Distance oracles for time-dependent networks. In *ICALP 2014*, pages 713–725, 2014.

22 P. Kouvelis and G. Yu. *Robust discrete optimization and its applications*, volume 14. Springer Science & Business Media, 2013.

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

24 G. Nannicini, D. Delling, D. Schultes, and L. Liberti. Bidirectional A^* search on time-dependent road networks. *Networks*, 59(2):240–251, 2012.

25 G. Yu and J. Yang. On the robust shortest path problem. *Computers & OR*, 25(6):457–468, 1998.

A Computing the Pareto Front

We provide as reference the proof of correctness for the time-dependent Martins' algorithm. The analysis of its run-time follows trivially from the one by Hansen [19].

▶ **Theorem 6.** *Let $G = (V, E)$ be a graph with edge weights $w : E \times T \to \mathbb{N}^2$ as in eq. (2). If $w_i : E \times T \to \mathbb{N}$ satisfies the FIFO property for every $i \in \{1, 2\}$, Listing 1 computes the Pareto front $\mathcal{F}$.*

Proof. The computed front π_t is not correct if there is a path in $\mathcal{F}$ that is not in π_t, there is a path in π_t that is not in $\mathcal{F}$, or both. We consider only the first case, since the remaining two follow from the fact that if π_t contains at least the paths in $\mathcal{F}$ then all other paths are dominated by those.

Suppose towards contradiction that there exists $P \in \mathcal{F}$ such that $P \notin \pi_t$. Consider the prefix P_{sv} of P from s to the first vertex v such that $P_{sv} \notin \pi_v$, and the suffix P_{vt} of P from v to t. We can express the weight of P as

$$w(P, \tau) = w(P_{sv}, \tau) + \begin{pmatrix} w_1(P_{vt}, \tau + w_1(P_{sv}, \tau)) \\ w_2(P_{vt}, \tau + w_2(P_{sv}, \tau)) \end{pmatrix}.$$

Since $P_{sv} \notin \pi_v$, there exists another path P'_{sv} dominating it, and we can obtain an s-t path P' (not necessarily simple) by concatenating P'_{sv} and P_{vt}. The weight of P' can be written as

$$w(P', \tau) = w(P'_{sv}, \tau) + \begin{pmatrix} w_1(P_{vt}, \tau + w_1(P'_{sv}, \tau)) \\ w_2(P_{vt}, \tau + w_2(P'_{sv}, \tau)) \end{pmatrix}.$$

Since P'_{sv} dominates P_{sv}, we know that, for every $i \in \{1, 2\}$, it holds that

$$w_i(P'_{sv}, \tau) \leq w_i(P_{sv}, \tau).$$

Since both w_1 and w_2 satisfy the FIFO property, we get that $w(P', \tau)$ dominates $w(P, \tau)$. This contradicts the assumption that $P \in \mathcal{F}$. ◀

▶ **Corollary 7.** *The run-time of Listing 1 is $O(nmW \cdot \log(nW))$, where*

$$W = \min_{i \in \{1,2\}} \left\{ \max_{e \in E, \tau \in T} w_i(e, \tau) \right\}.$$