Optimal Forks: Preprocessing Single-Source Shortest Path Instances with Interval Data

Niels Lindner
Zuse Institut Berlin, Germany

Pedro Maristany de las Casas[1]
Zuse Institut Berlin, Germany

Philine Schiewe
Department of Mathematics, Technische Universität Kaiserslautern, Germany

Abstract

We investigate preprocessing for single-source shortest path queries in digraphs, where arc costs are only known to lie in an interval. More precisely, we want to decide for each arc whether it is part of some shortest path tree for some realization of costs. We show that this problem is solvable in polynomial time by giving a combinatorial algorithm, using optimal structures that we call forks. Our algorithm turns out to be very efficient in practice, and is sometimes even superior in quality to a heuristic developed for the one-to-one shortest path problem in the context of passenger routing in public transport.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms

Keywords and phrases Preprocessing Shortest Path Problems, Interval Data, Graph Algorithms

Digital Object Identifier 10.4230/OASIcs.ATMOS.2021.7

Supplementary Material *Software (Source Code)*: `https://github.com/tfePedro/optimal-forks` archived at `swh:1:dir:f2f5d9db0b2228b799ca2d48dd9deff901f89725`

Funding *Pedro Maristany de las Casas*: German Federal Ministry of Education and Research (BMBF) (Research Campus MODAL 05M20ZBM).
Philine Schiewe: partially supported by DFG under grants SCHO 1140/8-2.

Acknowledgements We thank Lufthansa Systems GmbH & Co. KG. and in particular Marco Blanco for the data required to build the instance representing the airway network over Germany.

1 Introduction

The shortest path problem is fundamental to combinatorial optimization, and appears in various shapes in numerous applications, not only limited to the field of transportation. Although the classical shortest path problem is very efficiently solvable, it may still be computationally challenging due to the large size of the considered instances, or since frequent recomputations with different parameters are required. One example for the latter situation occurs in route planning in transportation networks, where arc costs depend on time [1]. E.g., travel times are affected by congestion in road networks, and aircraft flight routes depend on weather conditions. In public transportation networks, uncertain travel times plays a role not only during operations, e.g., in the case of delays, but also in the planning phase in the context of line planning, which is often performed before a timetable has been fixed [12].

[1] corresponding author

In these applications, although the travel times at a given time may be hard to predict, adequate lower and upper bounds are known. We therefore consider preprocessing of shortest path instances, where arc costs can be chosen arbitrarily within an interval. This applies not only to time-dependent shortest path problems, but to any situation where bounds on the arc costs are available, e.g., robust shortest path problems [8].

Our basic approach is to remove arcs when they can impossibly be on a shortest path. Ideally, for a given source s and target t, one would like to identify all the arcs that are not part of a shortest s-t-path. Several pruning heuristics have been developed for this purpose [9, 13], however, fast exact algorithms seem out of reach, as this problem is NP-complete [4, 6]. This is why we modify the problem as follows: We want to determine all arcs that cannot be on a shortest path tree rooted at a given source. A heuristic for detecting these arcs has been suggested in [2], but the complexity of the problem remained open.

We show that for a given source vertex s in a digraph on n vertices and m arcs, deciding whether an arc (w, v) can be part of a shortest path tree rooted at s is solvable in polynomial time, and construct an $\mathcal{O}(n(m + n \log n))$ algorithm. This result has been claimed in [7], but some proofs in this master thesis are incomplete or incorrect, the algorithm is more complicated than ours, and there are almost no computational results.

This paper is organized as follows: In Section 2, we reduce this single-source arc pruning problem to what we call the *s-v-w-scenario problem*, the latter serving as a basis of our considerations. Two mixed-integer programming formulations are presented in Section 3. We identify in Section 4 optimal substructures, called *forks*, which allow us to derive our combinatorial polynomial-time algorithm. Section 5 tests our single-source method on several real-world instances, and we compare our results to the one-to-one preprocessing heuristic developed for the purpose of integrated timetabling and passenger routing in [13]. We conclude the paper in Section 6.

2 Cost Scenarios and Weak Arcs

Let $G = (V, A)$ be a digraph. Let $\ell, u \in \mathbb{R}^A_{\geq 0}$ be lower resp. upper bounds for the arc costs, we assume that $\ell_a \leq u_a$ holds for every arc $a \in A$. A *cost scenario* is a vector $c \in \mathbb{R}^A_{\geq 0}$ that satisfies $\ell \leq c \leq u$. For a cost scenario c and vertices $s, t \in V$, we denote by $\Delta_{s,t}(c)$ the cost of a shortest s-t-path in G w.r.t. c. If p is a path in G containing the vertices v and w in this order, we denote by $p_{v,w}$ the v-w-subpath of p. We introduce at first the notion of weak arcs:

▶ **Definition 1** (cf. [6, 14]). *Let $s, t \in V$ be vertices in G.*

1. *An arc $a \in A$ is s-t-*weak *if there is a cost scenario c and a shortest s-t-path w.r.t. c containing a.*
2. *An arc $a \in A$ is s-*weak *if it is s-t-weak for some $t \in V$.*

The set of s-t-weak arcs defines the smallest subgraph of G that still contains all possible shortest s-t-paths w.r.t. all cost scenarios between ℓ and u. It is therefore desirable to characterize weak arcs algorithmically. However, there is the following negative result:

▶ **Theorem 2** ([3, 4, 6]). *Given a digraph G, lower and upper bounds ℓ and u, vertices $s, t \in V$, and an arc $a \in A$, it is strongly NP-complete to decide whether a is s-t-weak.*

In the single-source situation, there is an accessible characterization of weak arcs:

▶ **Lemma 3** ([2, Proposition 3]). *Let $s \in V$. An arc $a = (w, v) \in A$ is s-weak if and only if w is reachable from s and $\max\{\Delta_{s,v}(c) - \Delta_{s,w}(c) \mid \ell \leq c \leq u\} \geq \ell_a$.*

For fixed $s, v, w \in V$, we will therefore call $\Delta_{s,v}(c) - \Delta_{s,w}(c)$ the *value* of a cost scenario c.

▶ **Definition 4.** *Given a digraph G, lower and upper bounds ℓ and u, and vertices $s, v, w \in V$ such that v and w are reachable from s, the* s-v-w-scenario problem *is to find a cost scenario c of maximum value.*

The question whether an arc (w, v) can be removed from G without affecting shortest paths w.r.t. a cost scenario between ℓ and u can therefore be reduced to the solving s-v-w-scenario problem. An example for an optimal cost scenario can be seen in Figure 1.

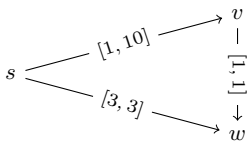

Figure 1 Digraph with interval data on arcs. The only optimal cost scenario is given by $c_{sv} = 10$, $c_{sw} = 3$, $c_{vw} = 1$, the value is 7.

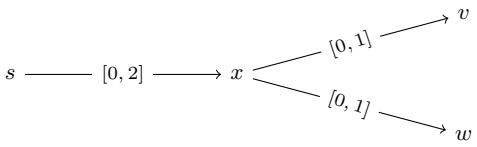

Figure 2 The shown s-v-w scenario instance is used in Example 7. It visualizes the statement in Lemma 6.

▶ **Definition 5.** *Let q be a path in G. The* cost scenario defined by q *is given by $c_a := \ell_a$ if $a \in q$ and $c_a := u_a$ otherwise.*

The following is a basic, but a rather cryptic self-referencing optimality condition. We refer to Appendix A for the proof.

▶ **Lemma 6** (cf. [6, Theorem 2.5])**.** *There is an optimal cost scenario c for the s-v-w-scenario problem defined by a shortest s-w-path w.r.t. c.*

▶ **Example 7.** Consider the graph in Figure 2. The cost scenario $c_{sx} = 1$, $c_{xv} = 1$, $c_{xw} = 0$ is optimal and has value 1. The shortest s-w-path w.r.t. c is $q = (s, x, w)$ with $c(q) = 1$. Using the notation from Lemma 6 and its proof, we can construct the cost scenario c^* with $c^*_{sx} = 0$, $c^*_{xv} = 1$, $c^*_{xw} = 0$ by setting the costs of all arcs along the shortest s-w-path $q = (s, x, w)$ w.r.t. c to their lower bound. Then c^* is the cost scenario induced by q, q is still a shortest path w.r.t. c^*, and c^* is an optimal solution to the s-v-w scenario problem with value 1.

3 Mixed-Integer Programming Formulations

In this section we provide two mixed-integer programs that solve the s-v-w-scenario problem. Resolving the maximum of a difference of minima and the linearization of shortest path costs require a few technical steps. The outcome is the program MIP_I:

$$\text{Maximize} \qquad \pi_v - \pi_s - \sum_{a \in A} y_a \tag{1a}$$

$$\text{s.t.} \qquad \pi_j - \pi_i \le c_{ij} \qquad (i,j) \in A \tag{1b}$$

$$\sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^-(i)} x_a = \begin{cases} 1 & i = s \\ -1 & i = w \\ 0 & \text{else} \end{cases} \qquad i \in V \tag{1c}$$

$$u_a(x_a - 1) + c_a \le y_a \le u_a x_a \qquad a \in A \tag{1d}$$

$$0 \le y_a \le c_a \qquad a \in A$$

$$\ell_a \le c_a \le u_a \qquad a \in A$$

$$x_a \in \{0, 1\} \qquad a \in A$$

$$\pi_i \in \mathbb{R} \qquad i \in V$$

▶ **Lemma 8.** $\mathrm{MIP_I}$ *solves the s-v-w-scenario problem.*

Proof. See Appendix B. ◀

Lemma 6 allows for a reduced mixed integer program $\mathrm{MIP_{II}}$ ([7]):

$$\text{Maximize} \qquad \pi_v - \pi_s - \sum_{a \in A} \ell_a x_a \tag{2a}$$

$$\text{s.t.} \qquad \pi_j - \pi_i \leq u_{ij} - (u_{ij} - \ell_{ij}) x_{ij} \qquad (i,j) \in A \tag{2b}$$

$$\sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^-(i)} x_a = \begin{cases} 1 & i = s \\ -1 & i = w \\ 0 & \text{else} \end{cases} \qquad i \in V \tag{2c}$$

$$x_a \in \{0,1\} \qquad a \in A$$

$$\pi_i \in \mathbb{R} \qquad i \in V$$

▶ **Lemma 9.** $\mathrm{MIP_{II}}$ *solves the s-v-w-scenario problem.*

Proof. See Appendix B. ◀

We want to remark that the proof of $\mathrm{MIP_{II}}$ in [7, Proposition 2.2] is incorrect: The author claims that in an optimal solution (π, x), x is always the incidence vector of a shortest s-w-path w.r.t. ℓ. However, as we will see in Remark 13, this is false.

4 Forks

We introduce forks as optimal combinatorial structures solving the s-v-w-scenario problem along with some properties in Section 4.1. Our algorithm is presented in Section 4.2.

4.1 The Theory of Forks

▶ **Definition 10.** *A* fork at $x \in V$ *is a pair* (p, q) *of paths in* G *such that*

1. q *is a shortest* s-w-*path w.r.t. the cost scenario* c *defined by* q*,*
2. p *is a shortest* s-v-*path w.r.t. the cost scenario* c*,*
3. p *and* q *both contain* x*,*
4. *the* s-x-*subpaths of* p *and* q *are identical,*
5. *the* x-v-*subpath of* p *and the* x-w-*subpath of* q *are arc-disjoint.*

We call c *the* cost scenario defined by the fork*, and we call* $\Delta_{s,v}(c) - \Delta_{s,w}(c)$ *the* value *of the fork.*

Figure 3 shows an example of a fork at a vertex $x \in V$ of the shown digraph. The following guarantees the existence of an optimal fork:

▶ **Lemma 11.** *For each shortest* s-w-*path* q *w.r.t. the cost scenario* c *defined by* q*, there is a fork* (p, q)*. In particular, there is an optimal solution* c^* *to the* s-v-w-*scenario problem such that* c^* *is defined by a fork.*

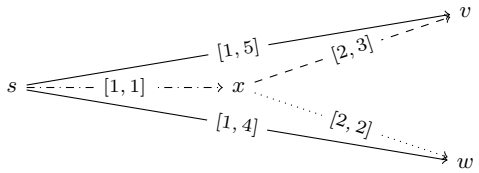

Figure 3 A digraph with a highlighted fork (p, q) at x. The dotted lines represent the s-w-path q. q is also a shortest s-w-path w.r.t. the cost scenario c defined by q. The s-v-path p is represented by the dashed lines and is minimal w.r.t. c. It shares its first arc (s, x) with q. After x, both paths diverge towards their target vertices.

Proof. Let q be a shortest s-w-path w.r.t. the cost scenario c defined by q. Let p be a shortest s-v-path w.r.t. c, and let x be the last common vertex of p and q. As the two s-x-subpaths of p and q are shortest w.r.t. c by subpath optimality, we can replace the s-x-subpath of p with the one of q and still guarantee that p is a shortest s-v-path w.r.t. c.

By Lemma 6, there is an optimal solution c defined by a shortest s-w-path q w.r.t. c. ◀

▶ **Lemma 12.** *Consider a fork (p, q) at x. Then $p_{x,v}$ is a shortest x-v-path w.r.t. u.*

Proof. Let p' be a shortest x-v-path w.r.t. u and assume that $u(p') < u(p_{x,v})$. Then

$$c(p') \leq u(p') < u(p_{x,v}) = c(p_{x,v}),$$

as $p_{x,v}$ uses only arcs $a \notin q$ with $c_a = u_a$, and this contradicts p containing a shortest x-v-path w.r.t. c. ◀

▶ Remark 13. It is in general not true that $q_{s,x}$ or $q_{x,w}$ are shortest paths w.r.t. ℓ. For the s-v-w-scenario instance in Figure 1, the only optimal cost scenario c^* with $c^*_{sv} = 10$, $c^*_{sw} = 3$, and $c^*_{vw} = 1$ is induced by a fork at $x = s$ built by the s-v-path $p = (s, v)$ and the s-w-path $q = (s, w)$. In this fork, the $q_{x,w}$-subpath is not a shortest x-w-path w.r.t. ℓ.

The value of a fork at x can be computed only by knowing x and $q_{x,w}$:

▶ **Lemma 14.** *The value of a (p, q) fork at x equals $\Delta_{x,v}(u) - \Delta_{x,w}(c) = \Delta_{x,v}(u) - \ell(q_{x,w})$.*

Proof. Let c be the cost scenario defined by q. Then

$$\begin{aligned}
\Delta_{s,v}(c) - \Delta_{s,w}(c) &= c(p_{s,v}) - c(q_{s,w}) \\
&= c(p_{s,x}) + c(p_{x,v}) - c(q_{s,x}) - c(q_{x,w}) \\
&= c(p_{x,v}) - c(q_{x,w}) \\
&= \Delta_{x,v}(c) - \Delta_{x,w}(c). \\
&= \Delta_{x,v}(u) - \ell(q_{x,w}).
\end{aligned}$$

◀

The following definition is essential for our algorithm:

▶ **Definition 15.** *Let $x \in V$. We call an s-w-path q* upper-bound-respecting at x *if*
(P1) *q contains x,*
(P2) *for all vertices j of $q_{s,x}$ holds $\ell(q_{j,x}) \leq \Delta_{j,v}(u) - \Delta_{x,v}(u)$,*
(P3) *for all vertices j of $q_{x,w}$ holds $\ell(q_{x,j}) \geq \Delta_{x,v}(u) - \Delta_{j,v}(u)$,*
(P4) *q is a shortest s-w-path for the cost scenario defined by q.*

▶ **Lemma 16.** *Let (p, q) be a fork at x. Then q is upper-bound-respecting.*

Proof. Let (p, q) be a fork at x defining the cost scenario c. Properties (P1) and (P4) are clear. For (P2), suppose that j comes before x on q. Then, as p contains a shortest j-v-path w.r.t. c via x, and by Lemma 12,

$$\Delta_{j,v}(u) \geq \Delta_{j,v}(c) = \Delta_{j,x}(c) + \Delta_{x,v}(c) = \Delta_{j,x}(c) + \Delta_{x,v}(u).$$

As $q_{j,x}$ is a shortest j-x-path w.r.t. c, we have $\Delta_{j,x}(c) = c(q_{j,x}) = \ell(q_{j,x})$. It remains to show (P3). Let j be a vertex of q after x. Since p contains a shortest x-v-path and by Lemma 12,

$$\Delta_{x,j}(c) + \Delta_{j,v}(u) \geq \Delta_{x,j}(c) + \Delta_{j,v}(c) \geq \Delta_{x,v}(c) = \Delta_{x,v}(u),$$

and we have $\Delta_{x,j}(c) = c(q_{x,j}) = \ell(q_{x,j})$. ◀

Recall that by Lemma 11, we know that there is an optimal cost scenario for the s-v-w-scenario problem that is defined by a fork. Our combinatorial algorithm will search for upper-bound-respecting paths to solve the s-v-w-scenario problem. The following lemma states that cost scenarios induced by upper-bound-respecting x-w-paths at x with minimal x-w-subpaths can only be better in value for the v-w-scenario problem than forks at x.

▶ **Lemma 17.** *Let c be a cost scenario defined by a fork (p, q) at x. Let q' be an upper-bound-respecting s-w-path at x with minimum $\ell(q')$, defining a cost scenario c'. Then*

$$\Delta_{s,v}(c') - \Delta_{s,w}(c') \geq \Delta_{s,v}(c) - \Delta_{s,w}(c).$$

Proof. First note that

$$\Delta_{s,w}(c') \leq \ell(q') = \ell(q'_{s,x}) + \ell(q'_{x,w}) \leq \ell(q'_{s,x}) + \ell(q_{x,w}),$$

as q is upper-bound-respecting by Lemma 16. Let p' be a shortest s-v-path w.r.t. c', let j denote the last common vertex of p' and q'.

Case 1: j is on $q'_{s,x}$. Then

$$\Delta_{s,v}(c') = c'(q'_{s,j}) + c'(p'_{j,v}) = \ell(q'_{s,j}) + u(p'_{j,v}) \geq \ell(q'_{s,j}) + \Delta_{j,v}(u).$$

Using that $\ell(q'_{s,x}) = \ell(q'_{s,j}) + \ell(q'_{j,x})$, we obtain from the condition (P2)

$$\ell(q'_{s,j}) = \ell(q'_{s,x}) - \ell(q'_{j,x}) \geq \ell(q'_{s,x}) + \Delta_{x,v}(u) - \Delta_{j,v}(u).$$

Inserting this,

$$\Delta_{s,v}(c') \geq \ell(q'_{s,x}) + \Delta_{x,v}(u),$$

so that, with the help of Lemma 14,

$$\Delta_{s,v}(c') - \Delta_{s,w}(c') \geq \Delta_{x,v}(u) - \ell(q'_{x,w}) \geq \Delta_{x,v}(u) - \ell(q_{x,w}) = \Delta_{s,v}(c) - \Delta_{s,w}(c).$$

Case 2: j is on $q'_{x,w}$. Then

$$\Delta_{s,v}(c') = \Delta_{s,x}(c') + \Delta_{x,j}(c') + \Delta_{j,v}(c') = \ell(q'_{s,x}) + \ell(q'_{x,j}) + u(p'_{j,v}) \geq \ell(q'_{s,x}) + \ell(q'_{x,j}) + \Delta_{j,v}(u).$$

By property (P3),

$$\ell(q'_{x,j}) \geq \Delta_{x,v}(u) - \Delta_{j,v}(u),$$

so that

$$\Delta_{s,v}(c') \geq \ell(q'_{s,x}) + \Delta_{x,v}(u),$$

and we find by Lemma 14

$$\Delta_{s,v}(c') - \Delta_{s,w}(c') \geq \Delta_{x,v}(u) - \ell(q_{x,w}) = \Delta_{s,v}(c) - \Delta_{s,w}(c).$$ ◀

4.2 Finding Optimal Cost Scenarios

Lemma 17 motivates Algorithm 1: Iterate over all vertices $x \in V$ and search for an upper-bound-respecting s-w-path q through x with minimum cost w.r.t. ℓ. If this cost equals $\Delta_{s,w}(c)$ for the cost scenario c defined by q, Lemma 17 ensures that the value of c is at least the value of any fork at x. Collecting the values of all those cost scenarios c for all x, we find an upper bound on the value of an optimal fork. But as there is an optimal fork by Lemma 11, we have solved the s-v-w-scenario problem:

▶ **Theorem 18.** *Algorithm 1 solves the s-v-w-scenario problem.*

An optimal fork can in principle be recovered by the procedure indicated in the proof of Lemma 11.

Algorithm 1 s-v-w-scenario solver.

Input : digraph $G = (V, A)$, arc cost bounds ℓ, $u \in \mathbb{R}^A_{\geq 0}$, vertices $s, v, w \in V$
Output : $\max\{\Delta_{s,v}(c) - \Delta_{s,w}(c) \mid \ell \leq c \leq u\}$ or $-\infty$

```
1  M ← −∞                                          /* maximum cost scenario value */
2  if v and w are reachable from s then
3      for x ∈ V do
4          q_{s,x} ← shortest s-x-path w.r.t. ℓ subject to (P2)
5          q_{x,w} ← shortest x-w-path w.r.t. ℓ subject to (P3)
6          if q_{s,x} ≠ NULL and q_{x,w} ≠ NULL then
7              q ← q_{s,x} + q_{x,w}                    /* concatenation of paths */
8              c ← cost scenario defined by q
9              if Δ_{s,w}(c) = ℓ(q) then
10                 M ← max{M, Δ_{s,v}(c) − Δ_{s,w}(c)}
11 return M
```

Implementation

Algorithm 1 returns the value M of an optimal cost-scenario for the s-v-w-scenario problem. M is initialized to $-\infty$ and can be immediately returned in case v or w are not reachable from s. For any other non-trivial input, an efficient implementation of Algorithm 1 requires a shortest-distance matrix Δ w.r.t. the upper bound costs u, to be able to check conditions (P2) and (P3) fast in the shortest path queries triggered in Lines 4 and 5. These lines are executed in the main loop of the algorithm for every $x \in V$ to find an upper-bound-respecting s-w-path at x in two stages. In Line 4, an s-x-subpath $q_{s,x}$ with minimum cost w.r.t. ℓ among the s-x-paths whose vertices fulfill (P2) is computed. Then, in Line 5 an x-w-subpath $q_{x,w}$ that is again minimal w.r.t. ℓ given that all its vertices fulfill (P3) is computed.

To compute both subpaths, we use a modified version of Dijkstra's algorithm w.r.t. the lower bounds ℓ. The query to compute $q_{s,x}$ is run from x to s on the reversed digraph $\overleftarrow{G}$ of G rather than from s to x. Then, while $q_{s,x}$ is being computed, let $q_{x,i}$ be a path extracted from the priority queue for some vertex $i \in V$. For any outgoing arc $(i, j) \in \overleftarrow{A}$ we build new x-j-subpaths $q_{x,j}$ but only those fulfilling $\ell(q_{x,j}) \leq \Delta_{j,v}(u) - \Delta_{x,v}(u)$, which is exactly condition (P2), are further considered in the elsewhere unaltered execution of Dijkstra's algorithm. This query runs from x to s because the left hand side of (P2) evaluates an j-x-subpath. If we would run Dijkstra's algorithm on the original digraph G, we would only

be able to evaluate s-j-subpaths for some $j \in V$. The computation of the $q_{x,w}$-subpath works very similarly. It uses the original digraph G and checks condition (P3) for every new path candidate. Note that if the shortest distance matrix Δ is known, the additional checks during the Dijkstra queries can be done in $\mathcal{O}(1)$ and thus do not have an impact on the overall complexity of the algorithm.

In case our modified Dijkstra find paths $q_{s,x}$ and $q_{x,w}$, their concatenation $q := q_{s,x} + q_{x,w}$ clearly fulfills conditions (P1)-(P3). If we then build the cost scenario c induced by q (Line 8) and run a one-to-all Dijkstra query w.r.t. c starting at s, we can check whether q also fulfills (P4) (Line 9). If it does, q is an upper-bound-respecting s-w-path through x. Thus, it qualifies to possibly update the return value M in case c yields a better value for the s-v-w-scenario problem than the best value known so far (Line 10).

▶ **Theorem 19.** *Algorithm 1 runs in $\mathcal{O}(n(n\log(n)+m))$.*

Proof. The computation of the shortest-distance matrix Δ can be done in $\mathcal{O}(n(n\log(n)+m))$ using Johnson's algorithm [5]. In every iteration of its main loop, the algorithm runs at most three Dijkstra queries (Line 4, Line 5, and Line 9). The distances $\Delta_{s,v}(c)$ (Line 10) and $\Delta_{s,w}(c)$ (Line 9) can be computed in the same Dijkstra query. Since n iterations are performed in total, this results in a running time of $\mathcal{O}(n(n\log(n)+m))$ for the main loop, if we assume that a Fibonacci heap is used. ◀

5 Computational Results

The aim of this section is twofold. First, in Section 5.2, we compute the sets of weak arcs for one-to-all shortest path instances with interval data using MIP_{II} and Algorithm 1, and show that the latter method is much faster. Secondly, in Section 5.3, we compare our method to an *arc-based pruning heuristic* introduced in [13] to show that our exact method is more effective.

The arc-based pruning from [13] works for the one-to-one shortest path problem with interval data. We assume that every graph has a set V_S of origin vertices and a set V_T of target vertices. Then, for $s \in V_S$ and $t \in V_T$, an arc $(v,w) \in A$ is guaranteed to not lie on any shortest s-t-path if

$$\Delta_{s,t}(u) < \Delta_{s,v}(\ell) + \ell_{v,w} + \Delta_{w,t}(\ell). \tag{3}$$

For a proof, see [13, Theorem 4]. Note that this criterion does not imply that an arc that does not fulfill (3) is s-t-weak. To check (3) algorithmically, two shortest path trees w.r.t. ℓ have to be computed: one rooted at s and one rooted at t on the reversed digraph of G. Then, for every vertex $v \in V$, the distances $\Delta_{s,v}(\ell)$ and $\Delta_{v,t}(\ell)$ are known and (3) can be used to discard irrelevant arcs from G (cf. [13, Algorithm 3]). Since this method discards irrelevant arcs for a fixed s-t-pair only, the set of remaining arcs is not directly comparable to the set of s-weak arcs. Therefore, we repeatedly apply the arc-based pruning from a fixed origin vertex $s \in V_S$ to all vertices $v \in V$. Then, for any s-v-pair, we get a set $A_{s,v}$ of arcs that do not lie on any shortest s-v-path. Consequently, only the arcs in $A_s := \bigcap_{v \in V} A_{s,v}$ are guaranteed to not lie on any shortest path starting at s. The complement of A_s is a superset of the set of s-weak arcs. We call the procedure of computing the sets A_s for all $s \in V_S$ the *V_S-V arc-based pruning*.

For realistic routing instances, it is often interesting to focus only on s-t-pairs for vertices $t \in V_T$. In contrast to the finding of s-weak arcs, the V_S-V arc-based pruning can easily be adapted to consider this setting. We get arcs $A_s^T := \bigcap_{t \in V^T} A_{s,t}$ that are guaranteed to

not lie on a shortest path from s to any target vertex $t \in V^T$. Thus, the complement of A_s^T contains all arcs that can lie on a shortest s-t path with $t \in V_T$. We call the computation of A_s^T the V_S-V_T *arc-based pruning*. In Section 5.3 we compare the sets of arcs returned by both arc-based pruning techniques with the sets of s-weak arcs.

5.1 Instance Description and Implementation Details

Table 3 in Appendix C shows an overview of the used instances and their size. The instances `toy`, `grid`, `lowersaxony`, and `athens` are taken from the open source software framework LinTim [11, 10] which contains algorithms and data sets for public transport planning. While `toy` and `grid` are artificial data sets, `lowersaxony` and `athens` represent the regional train system of Lower Saxony and the metro in Athens, respectively. For most instances, lower and upper bounds on the arcs are distinct, but for instance `grid-fix` the lower and upper bound coincide for a substantial amount of the arcs, as the duration of drive and wait activities is fixed. Instances `*-res` (stands for *res*tricted) and `*-all` vary in the set of transfer stations and thus in the number of arcs. The restriction of transfer stations is done according to [12] such that for instances with fixed drive and wait activities the optimal travel time is not impacted.

The instances `W1` to `W9` model subnetworks of the public transport network of the city of Wuppertal. As with `grid-fix`, drive and wait activities are fixed, moreover, the minimum transfer time is uniform across all stations.

We also consider the airway network above Germany. It is a layered directed graph, whose layers are connected via climb and descend arcs. A path using arcs in a higher layer represents a flight cruising at a higher altitude. A single arc has a copy in multiple layers and different costs in each of them. This discrete set of costs per arc allows us to derive lower and upper bounds on the arcs' costs. We then run all algorithms on a projection of the layered graph in which each arc appears only once, and climbing and descending arcs are ignored.

We use Gurobi 9.1.0 to solve the MIP_{II} models. Algorithm 1 (see supplementary material) and the arc-based prunings are implemented in C++ and compiled using gcc 7.5.0 and gcc 7.4.0, respectively. Gurobi and Algorithm 1 were run on a computer with an Intel Xeon CPU E5-2670 v2 @ 2.50GHz processor and 128GB of memory. The arc-based prunings are were run on a computer with an AMD Ryzen 5 PRO 2500U @ 2.00GHz processor and 16GB of memory.

5.2 Running Time: MIP_{II} vs. Algorithm 1

In every instance, we iterate over the origin vertices $s \in V_S$ and solve the resulting s-v-w-scenario problems for all arcs $(w, v) \in A$ using the MIP_{II} model and using Algorithm 1. For every origin vertex $s \in V_S$, we consider the average running time t_s needed to solve the s-v-w-scenario problems. Finally, in Table 1 we report the average of all t_s values, $s \in V_S$, for every considered graph and both solution approaches.

On the `W*` instances, we observe that the solutions calculated using Algorithm 1 are obtained orders of magnitude faster than using MIP_{II} models. The `W7` instance is the first for which not all origin vertices can be considered within the time limit of three days. In contrast, the running time of Algorithm 1 remains low even for the biggest instance `W9` since all s-v-w-scenario instances for a fixed origin vertex s can be solved within $3.2s$ in average. On the `toy-res` and `athens-res` instances, all MIPs could be solved but Algorithm 1 is around 4 orders of magnitude faster. On all other instances no optimal solution to the MIP_{II} models could be found due to memory restrictions or timeouts. The `air-germany` instance is the biggest instance and the running times of Algorithm 1 behave accordingly: on average, solving all s-v-w-scenarios for a fixed origin vertex takes $1551.28s$.

Table 1 Average time in seconds needed to compute the sets of s-weak arcs using Algorithm 1 and the MIP_{II} models. The *Solved* column reports the percentage of the origin vertices $s \in V_S$ for which all s-v-w-scenarios were solved by Gurobi. Algorithm 1 solved all instances. The computations stopped after 72 hours.

Instance	Algorithm 1	MIP_{II}	
Graph Name	Avg. time	Solved [%]	Avg. time
`toy-all`	0.1141	0.0	–
`toy-res`	0.0446	100.0	307.6409
`grid-all`	1.4550	0.0	–
`grid-res`	0.5730	0.0	–
`grid-fix-all`	1.4119	0.0	–
`grid-fix-res`	0.5417	0.0	–
`W1`	0.0019	100.0	4.8697
`W2`	0.0067	100.0	15.6175
`W3`	0.0211	100.0	68.7970
`W4`	0.2150	100.0	620.3328
`W5`	0.1804	100.0	698.2918
`W6`	0.6636	100.0	2466.2169
`W7`	1.1665	46.0	4015.982
`W8`	1.7622	37.2	4936.5306
`W9`	3.2349	15.7	9464.8684
`lowersaxony-all`	0.9811	0.0	–
`lowersaxony-res`	0.3155	0.0	–
`athens-all`	3.3776	0.0	–
`athens-res`	1.1595	100.0	3841.8138
`air-germany`	1551.2769	0.0	–

5.3 Effectiveness: Arc-based Pruning vs. Weak Arcs Solvers

We now compare the sets of s-weak arcs and the sets of remaining arcs after applying the V_S-V and the V_S-V_T arc-based pruning. In Table 2, we report the average cardinality of these sets after computing them for every $s \in V_S$ and $v \in V_T$ or $v \in V$ depending on the arc-based pruning variant. Figure 4 visualizes the same data. Recall that lower numbers are better since they imply that more arcs could be discarded. The remaining arcs after the V_S-V arc-based pruning are always a superset of the s-weak arcs and of the remaining arcs after the V_S-V_T arc-based pruning. There is no theoretical implication relating the size of the latter two sets.

On the `W*` instances, the difference between the set of s-weak arcs and the remaining arcs using the V_S-V arc-based pruning increases as the instances get bigger. For the `W9` instance, 30% of the arcs are s-weak and after the V_S-V arc-based pruning 51% are kept. On all `W*` instances, the V_S-V_T arc-based pruning discards the most arcs but without a clear correlation with the instances' size: for example, compared to the sets of s-weak arcs, it discards 10% more arcs for `W2` and for `W9` the advantage shrinks to only 1%.

On the synthetic and the remaining public transportation instances, we observe interesting results: There are not many target vertices and still the sets of arcs discarded by the V_S-V and V_S-V_T arc-based prunings are almost equal. Additionally, the sets of s-weak arcs turn out to be always smaller. This effect is particularly notable on the `grid-all` and `grid-fix-all` instances, where both arc-based prunings are equally effective and the s-weak arcs are 8% and 10% less, respectively. On the non-synthetic instances `lowersaxony-all` and `athens-all` the arc-based prunings again coincide and the sets of s-weak arcs contain 5% and 3% fewer arcs. Taking the average over the origin vertices, 80% of arcs are s-weak in `lowersaxony-all` and 68% in `athens-all`. Regarding the `air-germany` instance the sets of s-weak arcs contain

Table 2 Average number of remaining arcs for the s-v-w-scenario solver and the arc-based prunings. The averages are built among the number of weak arcs for every fixed vertex $s \in V_S$.

Graph Name	avg. weak arcs s-v-w-solver		avg. remaining arcs V_S-V_T-arc-pruning		avg. remaining arcs V_S-V-arc-pruning	
	tot.	rel.	tot.	rel.	tot.	rel.
toy-all	681	0.59	768	0.66	768	0.66
toy-res	498	0.48	526	0.50	542	0.52
grid-all	1953	0.75	2148	0.83	2148	0.83
grid-res	1464	0.62	1563	0.66	1572	0.67
grid-fix-all	1817	0.70	2088	0.80	2091	0.80
grid-fix-res	1355	0.58	1453	0.62	1504	0.64
W1	56	0.40	43	0.30	56	0.40
W2	95	0.38	69	0.28	104	0.42
W3	155	0.32	123	0.26	170	0.35
W4	359	0.34	289	0.27	451	0.43
W5	351	0.35	330	0.33	467	0.47
W6	587	0.35	520	0.31	846	0.51
W7	719	0.32	682	0.30	1127	0.50
W8	908	0.31	768	0.26	1350	0.46
W9	1177	0.30	1120	0.29	1984	0.51
lowersaxony-all	1410	0.80	1492	0.85	1492	0.85
lowersaxony-res	964	0.64	1004	0.67	1007	0.67
athens-all	2368	0.68	2482	0.71	2482	0.71
athens-res	1414	0.51	1454	0.52	1455	0.52
air-germany	13610	0.41	16504	0.49	27656	0.83

42% less arcs than the sets of remaining arcs after the V_S-V arc-based pruning. The V_S-V_T arc-based pruning works better than the latter but on average, the sets of remaining arc contain 8% more arcs than the sets of s-weak arcs. Figure 5 contains a plot showing the distribution of the size of the sets of remaining arcs per origin vertex for a representative instance of each type.

6 Conclusion

Deciding whether an arc is s-weak for some source vertex s can be done efficiently at the same complexity as a standard all-to-all shortest path query, using a series of minor modifications of Dijkstra's algorithm. It is hence no surprise that Algorithm 1 runs much faster than commercial solvers on the mixed-integer programming formulation MIP_{II}. In quality, our algorithm performs at least comparable to the arc-based pruning heuristic from [13], for some instances, it is even superior, although that heuristic has been developed for the one-to-one shortest path queries. We hence conclude that our s-v-w-scenario algorithm can serve as a powerful preprocessing tool for shortest path problems in a variety of application contexts.

Beyond testing our method on a larger variety of instances, e.g., road networks, and combining the algorithm with the V_S-V_T arc pruning in an iterative process, a natural question is to tackle the complexity of computing a subgraph of minimum size that contains at least one shortest path for each cost scenario, rather than containing all shortest paths. Another related problem is the detection of arcs that are part of a shortest path trees for all cost scenarios. These are called *strong arcs* in the robust optimization literature (e.g., [14]), and following [7], it seems that similar methods are available here.

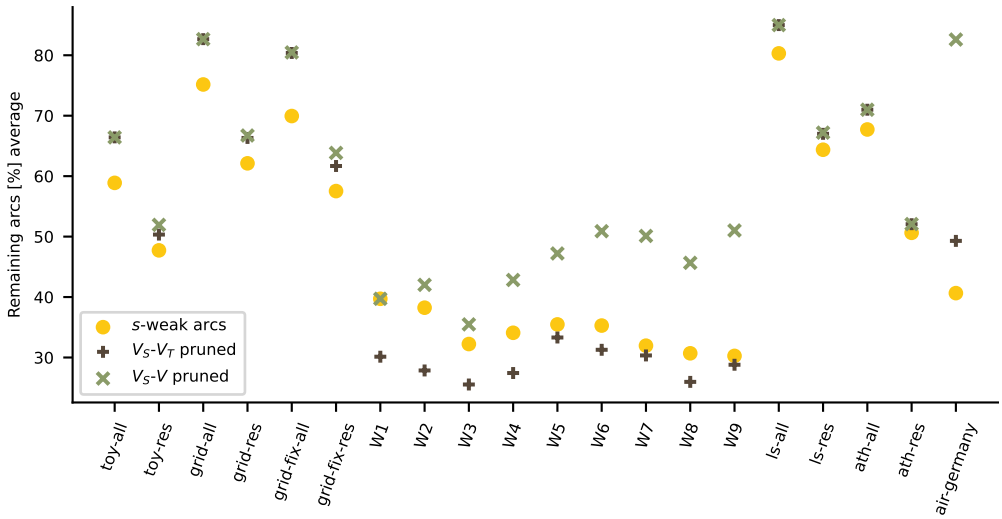

Figure 4 Average cardinality (as a percentage of the total number of arcs in each instance) of the sets of relevant arcs determined using Algorithm 1 and the V_S-V_T and V_S-V arc-based prunings.

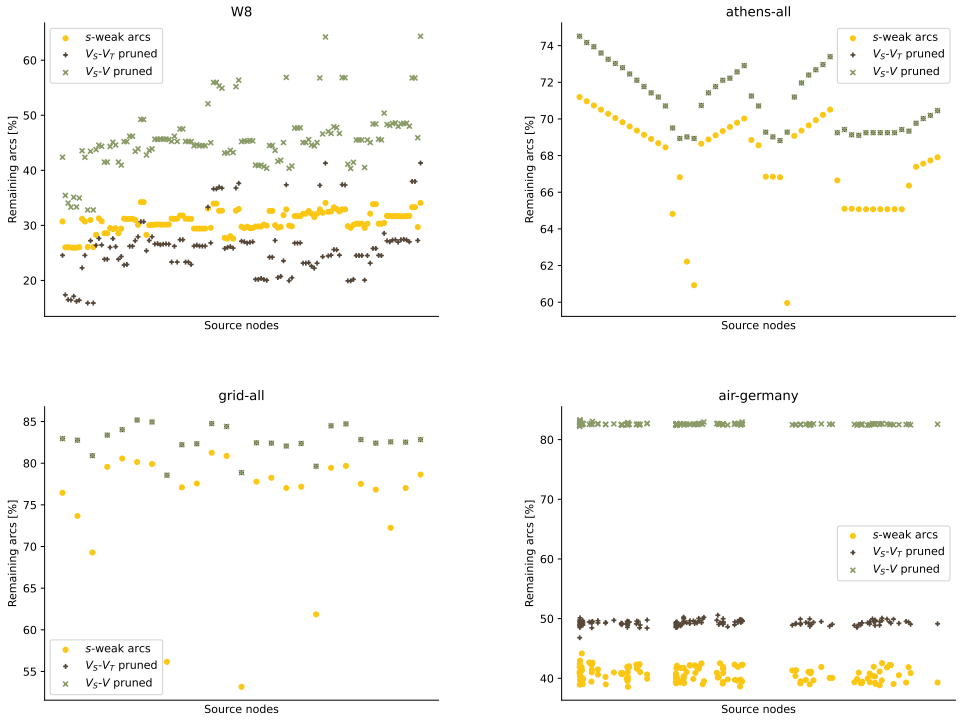

Figure 5 Remaining arcs per origin vertex.

References

1 H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route Planning in Transportation Networks. In *Algorithm Engineering*, 2016. doi:10.1007/978-3-319-49487-6_2.

2 D. Catanzaro, M. Labbé, and M. Salazar-Neumann. Reduction approaches for robust shortest path problems. *Computers & Operations Research*, 38(11):1610–1619, 2011. doi:10.1016/j.cor.2011.01.022.

3 S. Chanas and P. Zieliński. The computational complexity of the criticality problems in a network with interval activity times. *European Journal of Operational Research*, 136(3):541–550, February 2002. doi:10.1016/s0377-2217(01)00048-0.

4 S. Chanas and P. Zieliński. On the hardness of evaluating criticality of activities in a planar network with duration intervals. *Operations Research Letters*, 31(1):53–59, 2003. doi:10.1016/S0167-6377(02)00174-8.

5 D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, January 1977. doi:10.1145/321992.321993.

6 O. E. Karasan, M. Pinar, and H. Yaman. The Robust Shortest Path Problem with Interval Data. Technical report, Bilkent University, 2001.

7 F. Krellner. Shortest Paths with Interval Data and their Application in Timetabling. Master's thesis, Freie Universität Berlin, 2018.

8 R. Montemanni and L. M. Gambardella. An exact algorithm for the robust shortest path problem with interval data. *Computers & Operations Research*, 31(10):1667–1680, 2004. doi:10.1016/S0305-0548(03)00114-X.

9 A. Schienle, P. Maristany, and M. Blanco. A Priori Search Space Pruning in the Flight Planning Problem. In Valentina Cacchiani and Alberto Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASIcs)*, pages 8:1–8:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 2190-6807. doi:10.4230/OASIcs.ATMOS.2019.8.

10 A. Schiewe, S. Albert, P. Schiewe, A. Schöbel, and F. Spühler. LinTim - Integrated Optimization in Public Transportation. Homepage. https://lintim.net, 2020.

11 A. Schiewe, S. Albert, P. Schiewe, A. Schöbel, and F. Spühler. LinTim: An integrated environment for mathematical public transport optimization. Documentation for version 2020.12. Technical report, TU Kaiserslautern, 2020. URL: https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-62025.

12 P. Schiewe. *Integrated Optimization in Public Transport Planning*, volume 160 of *Optimization and Its Applications*. Springer, 2020. doi:10.1007/978-3-030-46270-3.

13 P. Schiewe and A. Schöbel. Periodic Timetabling with Integrated Routing: Toward Applicable Approaches. *Transportation Science*, 54(6):1714–1731, 2020. doi:10.1287/trsc.2019.0965.

14 H. Yaman, O. E. Karaşan, and M. Ç. Pınar. The robust spanning tree problem with interval data. *Operations Research Letters*, 29(1):31–40, 2001. doi:10.1016/S0167-6377(01)00078-5.

A Proof of Lemma 6

▶ **Lemma 6** (cf. [6, Theorem 2.5]). *There is an optimal cost scenario c for the s-v-w-scenario problem defined by a shortest s-w-path w.r.t. c.*

Proof. Consider an optimal solution c and a shortest s-w-path q w.r.t. c. Further consider the cost scenario c^* defined by q in the sense of Definition 5. We claim at first that q is also a shortest s-w-path w.r.t. c^*. Indeed, let q' be any s-w-path. Then, as q is shortest w.r.t. c,

$$\sum_{a \in q \cap q'} c_a + \sum_{a \in q \setminus q'} c_a = c(q) \leq c(q') = \sum_{a \in q \cap q'} c_a + \sum_{a \in q' \setminus q} c_a,$$

we find

$$\sum_{a\in q\setminus q'} c^*_a = \sum_{a\in q\setminus q'} \ell_a \le \sum_{a\in q\setminus q'} c_a \le \sum_{a\in q'\setminus q} c_a \le \sum_{a\in q'\setminus q} u_a = \sum_{a\in q'\setminus q} c^*_a,$$

and therefore

$$c^*(q) = \sum_{a\in q\cap q'} c^*_a + \sum_{a\in q\setminus q'} c^*_a \le \sum_{a\in q\cap q'} c^*_a + \sum_{a\in q'\setminus q} c^*_a = c^*(q').$$

In particular, the cost scenario c^* is defined by the shortest s-w-path q w.r.t c^*.

Now let p be a shortest s-v-path w.r.t. c^*. We then have

$$\begin{aligned}
\Delta_{s,v}(c^*) - \Delta_{s,w}(c^*) &= c^*(p) - c^*(q) \\
&= \sum_{a\in p\setminus q} c^*_a - \sum_{a\in q\setminus p} c^*_a \\
&= \sum_{a\in p\setminus q} u_a - \sum_{a\in q\setminus p} \ell_a \\
&\ge \sum_{a\in p\setminus q} c_a - \sum_{a\in q\setminus p} c_a \\
&= c(p) - c(q) \ge \Delta_{s,v}(c) - \Delta_{s,w}(c).
\end{aligned}$$

But as c was optimal, we must have that c^* is optimal as well. ◀

B Correctness Proofs for $\mathrm{MIP_I}$ and $\mathrm{MIP_{II}}$

▶ **Lemma 8.** $\mathrm{MIP_I}$ *solves the s-v-w-scenario problem.*

Proof. Let c^* be an optimal cost scenario and let q be a shortest s-w-path w.r.t. c^*. Set $x^*_a := 1$ for all arcs $a \in q$ and $x^*_a := 0$ otherwise. Then x^* satisfies the flow constraints (1c). For all vertices $i \in V$, set $\pi^*_i := \Delta_{s,i}(c^*)$. As $\Delta_{s,i}(c^*) + c^*_{ij} \ge \Delta_{s,j}(c^*)$ for all $(i,j) \in A$, π^* and c^* satisfy (1b). The coupling constraints (1d) yield a vector y^* with the property that $y^*_a = c^*_a$ for $a \in q$ and $y^*_a = 0$ otherwise, so that

$$\sum_{a\in A} y^*_a = \sum_{a\in A} c^*_a x^*_a = c^*(q) = \Delta_{s,w}(c^*).$$

We conclude that the objective value (1a) of this feasible solution is $\Delta_{s,v}(c^*) - \Delta_{s,w}(c^*)$, i.e., the value of c^*.

It remains to show that the optimal objective value of $\mathrm{MIP_I}$ is at most the value of c^*. To this end, let (c, π, x, y) be an optimal solution to $\mathrm{MIP_I}$. For given x, y, c, this optimal solution must satisfy $\pi_v - \pi_s = \Delta_{s,v}(c)$, as

$$\max\{\pi_v - \pi_s \mid \pi_j - \pi_i \le c_{ij} \text{ for all } (i,j) \in A, \pi_i \in \mathbb{R} \text{ for all } i \in V\}$$

is the dual linear programming formulation of the shortest s-v-path problem w.r.t. c. Moreover, as x indicates some s-w-path by (1c), and analyzing the coupling constraints (1d), we have $\sum_{a\in A} y_a = \sum_{a\in A} c_a x_a \ge \Delta_{s,w}(c)$. We conclude that the optimal value of $\mathrm{MIP_I}$ is at most $\Delta_{s,v}(c) - \Delta_{s,w}(c)$, and this is in turn at most the value of c^*. ◀

▶ **Lemma 9.** $\mathrm{MIP_{II}}$ *solves the s-v-w-scenario problem.*

Proof. Let c^* be an optimal cost scenario, we can complete c^* to an optimal solution (c^*, π^*, x^*, y^*) to MIP_I by the proof of Lemma 8. By Lemma 6, we can assume that x^* corresponds to a shortest s-w-path q w.r.t. c^*, and that c^* is defined by q. Clearly, x^* satisfies the flow constraints (2c). Moreover, for arcs $(i,j) \in A$ with $x^*_{ij} = 0$, we have $\pi^*_j - \pi^*_i \leq c^*_{ij} \leq u^*_{ij}$, so that (2b) holds. Otherwise, if $x^*_{ij} = 1$, we have that $c^*_{ij} = \ell_{ij}$, so that $\pi^*_j - \pi^*_i \leq \ell_{ij}$ and hence (2b) are satisfied. In particular, (π^*, x^*) is feasible for MIP_II, and we note that $\sum_{a \in A} \ell_a x^*_a = c^*(q^*) = \Delta_{s,w}(c^*)$, so that the objective value (2a) of (π^*, x^*) equals the $\Delta_{s,v}(c^*) - \Delta_{s,w}(c^*)$. This shows that the optimal objective value of MIP_II is at least the value of c^*.

Conversely, let (π, x) be an optimal solution to MIP_II. We obtain a feasible solution to MIP_I by defining $c_a := u_a - (u_a - \ell_a)x_a$ and $y_a := c_a x_a$ for all $a \in A$, and the objective value in MIP_I remains the same. Applying Lemma 8, the optimal value of MIP_II is at most the value of the optimal scenario c^*. ◀

C Instance Details

Table 3 Overview of the used instances.

Type	Name	Vertices	Arcs	Origins $\lvert V_S \rvert$	Targets $\lvert V_T \rvert$	$\ell_a = u_a$
Synthetic	`toy-all`	184	1156	8	8	188
	`toy-res`	184	1044	8	8	188
	`grid-all`	442	2598	25	25	392
	`grid-res`	442	2356	25	25	392
	`grid-fix-all`	442	2598	25	25	756
	`grid-fix-res`	442	2356	25	25	756
Public Transport	`W1`	56	142	28	28	142
	`W2`	88	248	36	36	236
	`W3`	122	480	52	52	436
	`W4`	254	1052	80	80	912
	`W5`	242	990	84	84	814
	`W6`	365	1663	102	102	1291
	`W7`	434	2249	111	111	1597
	`W8`	516	2957	129	129	2001
	`W9`	631	3889	140	140	2477
	`lowersaxony-all`	480	1756	34	34	412
	`lowersaxony-res`	480	1498	34	34	412
	`athens-all`	1066	3496	51	51	964
	`athens-res`	1066	2794	51	51	964
Air	`air-germany`	13896	26576	154	125	0