Optimal Upgrading Schemes for Effective Shortest Paths in Networks

Eduardo Álvarez-Miranda[1], Martin Luipersbeck[2] and Markus Sinnl[2]

[1] Department of Industrial Engineering, Universidad de Talca, Curicó, Chile
ealvarez@utalca.cl
[2] Department Statistics and Operations Research, University of Vienna, Vienna, Austria {martin.luipersbeck,markus.sinnl}@univie.ac.at

Abstract. In this paper, a generalization of a recently proposed optimal path problem concerning decisions for improving connectivity is considered (see Dilkina et al., 2011). Each node in the given network is associated with a connection delay which can be reduced by implementing upgrading actions. For each upgrading action a cost must be paid, and the sum must satisfy a budget constraint. Given a fixed budget, the goal is to choose a set of upgrading actions such that the total delay of establishing paths among predefined node pairs is minimized. This model has applications in areas like multicast communication planning and wildlife reserve design.
A novel formulation is provided along with an ad-hoc branch-and-cut and a stabilized Benders decomposition algorithm. These strategies exploit connections of the considered problem with other well-known network design problems. Computational results on a large set of instances show the efficacy of the proposed preprocessing methods and optimization algorithms with respect to existing alternatives for the problem. Complementary, the scalability of the models and the corresponding algorithms is investigated with the aim of answering questions raised by (Dilkina et al., 2011).

1 Introduction and Motivation

Finding optimal paths in networks is a fundamental task in a plethora of decision making contexts involving traffic in some form. The basic variant consists of finding a minimum distance path between two predefined points (nodes), source and target. The decision on which connections (or edges) must be chosen to connect source and target is typically taken by considering the sum of the lengths or weights of these edges (which is generally minimized), while respecting some other set of topological and/or operative requirements (the reader is referred to Vassilevska, 2008; Fournier, 2010, and the references therein, which propose models and algorithms for different optimal path problems).

Although edge-weighted networks offer a broad range of modeling possibilities in several applications, there exist problems in which decisions must be taken based on the set of nodes traversed by a given path. Hence, the performance

of a path depends on node weights rather than edge weights. For instance, in a multicast communication setting a backbone server broadcasts a signal to many subscribers; the layout of such communication network should be such that delays (which express when the signal traverses a node) between the server and all subscribers must be minimal or bounded. In such a problem, the decision maker seeks for an arrangement of nodes, technologies, and connections such that a positive function of node delays is minimized or it fulfills a Quality-of-Service (QoS) requirement.

The Upgrading Shortest Path problem (USP), originally proposed by (Dilkina et al., 2011), fits within the above mentioned context. In this problem, an *upgrading* action can be taken, at a certain cost, in order to decrease the delay induced by nodes. The optimization problem corresponds to (i) finding an upgrade strategy that induces a minimum network delay while respecting a total upgrade cost budget, or (ii) finding an upgrade strategy that yields a minimum total upgrade cost while ensuring that the overall delay is not greater than a given QoS bound. Moreover, and as stated in (Dilkina et al., 2011), the USP can be regarded as a decision aid tool for the design of wildlife reserves (see, e.g., Dilkina and Gomes, 2010).

The problem of finding optimal upgrading schemes for improving network effectiveness has been addressed before. In the seminal work (Paik and Sahni, 1995) several variants of network upgrading problems are proposed. For these variants delays can be caused both along edges and across nodes, so the upgrade decisions involve both components. The complexity of these problems is provided, showing that they range from polynomially solvable problems up to NP-hard problems. Later on, in (Krumke et al., 1999), a problem closely related to the USP is proposed. If a node is upgraded at a given cost, the weight of all incident edges is decreased. The goal is to find an upgrade strategy so as to reduce the total weight of a corresponding minimum spanning tree in the graph. Only complexity results are provided. In (Campbell et al., 2006) a set of arc-based upgrading problems is proposed; in all cases the aim is to find an upgrading strategy so that a min-max type of objective is optimized. Complexity results are provided as well as heuristic approaches.

From an algorithmic point of view, (Dilkina et al., 2011) first prove that the USP is NP-hard. Additionally, the authors provided a mixed integer linear programming (MIP) formulation for the USP and designed two greedy algorithms. The performance of these algorithms is contrasted with results obtained by solving the formulation using a stand-alone MIP solver. The MIP model and the heuristics are able to tackle synthetic grid instances with up to 20×20 nodes and a medium size real-world instance taken from a wildlife planning application. Although interesting, the obtained results reveal the need of developing more sophisticated exact tools able to solve larger instances while still providing reasonable guarantees of optimality.

Contribution and Paper Outline The aim of this paper is to provide different exact algorithmic tools to solve the budget constrained variant of the generalized counterpart of the USP. Generalized means that one can choose

among several upgrading actions at each node. This generalization is mentioned in (Dilkina et al., 2011) as an interesting topic for further work. Experimental results on a large set of benchmark instances show that the proposed methods are capable of outperforming the results provided by the compact formulation presented in (Dilkina et al., 2011) and, moreover, are capable of solving larger instances.

The paper is organized as follows. A formal definition of the problem and a formulation based on node separators, along with a corresponding exact algorithm, are presented in Section 2. A decomposable formulation along with a Benders decomposition scheme is provided in Section 3. Computational results on different data sets are reported in Section 4. Finally, concluding remarks are drawn in Section 5.

2 Cut-based Formulation

In this Section a formal definition of the problem is first presented. Afterwards, a formulation based on connectivity cuts is given along with a B&C scheme for tackling it.

Problem Definition Let $G = (V, E)$ be an undirected graph, where V is the set of nodes and E is the set of edges. Set $P \subseteq V \times V$ corresponds to the set of node pairs, say $p = (s, t)$, that must be connected by paths. Let $d_v \geq 0$, be the delay of node v, $v \in V$; likewise, for each upgrading level $l \in L$ and node $v \in V$, let $d_v^l \geq 0$ be the *reduced* delay of node v, if the node is upgraded to level l. Complementary, $c_v^l \geq 0$ corresponds to the cost of *upgrading* node $v \in V$ to level $l \in L$. Finally, let $B \geq 0$ be the total cost budget.

An *upgrading scheme* S is a partition $V^0 \cup V^1 \cup \ldots \cup V^l$ of the node set V, with the meaning that a node $i \in V^l$ is updated to level l, a node $i \in V^0$ is not updated. An upgrading scheme S is feasible, if the cost of the upgrading actions induced by S do not exceed B. Let $\mathcal{S}$ denote the family of all upgrading schemes. Let $D_p(S)$ be the delay of the shortest path connecting $p = \{s, t\}$ under upgrading scheme S. Using this notation, the problem can be formulated as follows

$$\min \left\{ \frac{1}{|P|} \sum_{p \in P} D_p(S) \;\; | \sum_{l \in L} \sum_{v \in V^l} c_v^l \leq B, S \in S \right\}.$$

In other words, we look for a feasible upgrading scheme that induces a minimum average path delay. This definition corresponds to the budget-constrained variant of the general USP. Note that even in the case where $|L| = 1$, the problem has been proven to be NP-hard (Dilkina et al., 2011). In the following, the constant term $\frac{1}{|P|}$ will be neglected for ease of exposition.

2.1 Node separators and MIP Formulation

Let $\mathbf{x} \in \{0, 1\}^{|V| \times |L|}$ be a vector of binary variables such that $x_v^l = 1$ if node $v \in V$ is upgraded to level $l \in L$, and $x_v^l = 0$ otherwise. Likewise, let $\mathbf{y} \in \{0, 1\}^{|V| \times |P|}$

be a vector of binary variables such that $y_{pv} = 1$ if node $v \in V$ is part of the path connecting the pair $p = (s,t) \in P$, and $y_{pv} = 0$ otherwise. Complementary, let $\mathbf{z} \in \{0,1\}^{|V|\times|P|\times|L|}$ be a vector of binary variables such that $z^l_{pv} = 1$ if the node $v \in V$, upgraded to level $l \in L$, is part of the path connecting the pair $p = (s,t) \in P$, and $z^l_{pv} = 0$ otherwise. For an arbitrary set of nodes, say $S \subseteq V$, for any pair $p \in P$ and for a given $l \in L$, the notation $\mathbf{y}_p(S) = \sum_{v\in S} y_{vp}$ and $\mathbf{z}^l_p(S) = \sum_{v\in S} z^l_{vp}$ will be used. The following definition is required.

Definition 1. ***(Node separator)*** *For a given pair $(s,t) \in P$, a subset of nodes $N \subseteq V \setminus \{s,t\}$ is called (s,t)* node separator *if and only if after eliminating N from V there is no (s,t) path in G. A separator N is* minimal *if $N \setminus \{i\}$ is not a (s,t) separator, for any $i \in N$. Let $\mathcal{N}(s,t)$ denote the family of all (s,t) separators.*

With these elements, a feasible set of paths along with an upgrading scheme must fulfill the following set of constraints,

$$\mathbf{y}_p(N) + \sum_{l\in L} \mathbf{z}^l_p(N) \geq 1, \qquad \forall N \in \mathcal{N}(s,t), p = (s,t) \in P \tag{C.1}$$

$$z^l_{pv} \leq x^l_v, \qquad \forall l \in L,\ \forall v \in V \setminus \{s,t\},\ \forall p = (s,t) \in P \tag{C.2}$$

$$\sum_{l\in L}\sum_{v\in V} c^l_v x^l_v \leq B. \tag{C.3}$$

Constraints (C.1) ensure that for every pair $p = (s,t)$ in P, there is a path comprised by a combination of normal nodes or upgraded nodes. Constraint (C.2) imposes that an upgraded node can be used ($z^l_{pv} = 1$), if and only if it has been actually upgraded ($x^l_v = 1$). Finally, constraint (C.3) imposes that any feasible upgrading scheme must meet the budget limitation. Hence, one can formulate the budget constrained USP as follows,

$$\text{(NODE)} \qquad \min \sum_{(s,t)=p\in P} \sum_{v\in V|_{v\neq s,t}} \left(d_v y_{pv} + \sum_{l\in L} d^l_v z^l_{pv} \right) \tag{1}$$

$$\text{s.t.} \qquad \text{(C.1)-(C.3)} \tag{2}$$

$$(\mathbf{x},\mathbf{y},\mathbf{z}) \in \{0,1\}^{|V|\times|L|+|V|\times|P|+|V|\times|P|\times|L|}. \tag{3}$$

Note that although this formulation contains an exponential number of constraints (C.1), it can be solved efficiently by a branch-and-cut (B&C) algorithm in which these constraints are added on-the-fly.

2.2 Branch-and-Cut Algorithm

The main ingredient of the B&C approach is its separation scheme, in which violated constraints of type (C.1) are identified during the exploration of the branch-and-bound tree. Moreover, two primal heuristic procedures are also discussed in this Section.

Separation Schemes Each time (NODE) is solved, the current LP solution $(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}})$ is used to compute a set of violated inequalities (C.1). To perform

separation, the following transformation of G into a bi-directed graph $G'_A = (V', A')$ is needed (see also (Álvarez-Miranda et al., 2013; Fischetti et al., 2014) for similar transformations). This graph is obtained by first bi-directing G (i.e., each edge is replaced by two anti-parallel arcs), and then splitting each node $i \in V$ into an arc (i_1, i_2). In other words, a graph $G'_A = (V', A')$ is created such that $V' = \{i_1 \mid i \in V\} \cup \{i_2 \mid i \in V\}$, $A' = \{(i_2, j_1) \mid (i, j) \in A\} \cup \{(i_1, i_2) \mid i \in V\}$. To separate inequalities (C.1) for a path $(s, t) = p \in P$, arc capacities in G'_A are defined as follows:

$$cap_{uv} = \begin{cases} \tilde{y}_{pi} + \sum_{l \in L} \tilde{z}^l_{pi}, & \text{if } u = i_1, v = i_2, i \in V, i \neq s, t \\ \infty, & \text{otherwise.} \end{cases}$$

Next, the maximum flow/minimum cut between s_2 and t_1 in G'_A is calculated. Note that due to the choice of arc capacities, a minimum (s_2, t_1) cut in G'_A solely contains split-arcs, and thus corresponds to an (s, t) node separator in G. If the computed maximum flow is smaller than one, the associated inequality of type (C.1) is violated, and subsequently added to (NODE).

Alternatively, if the current LP solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}})$ is integer at a given node of the search tree, the following more efficient separation scheme runs in linear time for each $p \in P$:

Let $\tilde{G}^p = (\tilde{V}^p, \tilde{E}^p)$ be the subgraph induced by $\tilde{V}^p = \{v \in V \mid \tilde{y}_{pv} + \sum_{l \in L} \tilde{z}^l_{pv} = 1\}$. If $\tilde{G}^p$ contains a path between s and t, no violated inequality exists. Otherwise, $\tilde{G}^p$ contains at least two disconnected components H^p_s and H^p_t, such that $s \in H^p_s$ and $t \in H^p_t$.

Let $\bar{H}^p_t$ be the set of neighboring nodes of H^p_t in G, i.e., $\bar{H}^p_t = \{v \in V \setminus H^p_t \mid \exists \{u, v\} \in E \text{ and } u \in H^p_t\}$. A minimal separator between s and t can be found as follows: (i) delete from G all edges induced by $H^p_t \cup \bar{H}^p_t$; (ii) apply a BFS from s, and let $R(s)$ be the set of all the reached nodes; finally, (iii) the set $\mathcal{N}_{s,t} = R(s) \cap \bar{H}^p_t$ defines a minimal (s, t) node separator, and the corresponding cut of type (C.1) is added to the model.

Primal Heuristic In order to accelerate the convergence of the method, a simple, but effective, LP-based procedure has been designed with the aim of using the current LP values for the construction of feasible (and eventually incumbent) solutions.

Let $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}})$ be the current LP solution; the primal heuristic works as follows,

Step 1: For every $v \in V$, find $\ell_v = \arg\max_{l \in L}(\tilde{x}^l_v)$, and calculate $\tilde{d}_v = (1 - \tilde{x}^{\ell_v}_v) d_v + \tilde{x}^{\ell_v}_v d^{\ell_v}_v$.

Step 2: Compute the shortest path (SP) for every $(s, t) = p \in P$ using the delay values calculated in **Step 1**; let $\tilde{Y} = \left\{\tilde{Y}^1, \tilde{Y}^2, \ldots, \tilde{Y}^{|P|}\right\}$ be such paths.

Step 3: For every $v \in V$ and $l \in L$, compute $\gamma^l_v = \frac{d_v - d^l_v}{c^l_v} \sum_{p \in P} \left|\tilde{Y}^p \cap v\right|$. Afterwards, use a knapsack-like heuristic to pack as many upgrades into the paths as possible (which defines $\mathbf{x}$), considering the order given by the values $\boldsymbol{\gamma}$.

In this procedure, the values of the $\mathbf{x}$ variables are set in Step 3; hence, the values of the $(\mathbf{y}, \mathbf{z})$ variables can be straightforwardly calculated from the paths obtained in Step 2.

Local Branching Along with the above mentioned construction heuristic, a state-of-the-art procedure for generating primal solutions using an MIP-solver as black-box, known as Local Branching is implemented (see (Fischetti and Lodi, 2003); the technique is also known as Limited Discrepancy Search (Harvey and Ginsberg, 1995) within the constraint programming community).

Roughly speaking, for a given (feasible) upgrade scheme $\tilde{\mathbf{x}}$, Let $S = \{(l, v) \mid \tilde{x}_v^l = 1,\ \forall v \in V, \forall l \in L\}$ be a set of pairs such that each element denotes if a node is upgraded using a certain upgrade type in the current incumbent solution. The goal is to find an improved *neighboring* solution containing at least $|S| - k$ upgrades from the current incumbent solution. This is achieved by solving the current model via branch-and-bound (B&B) after adding the following so-called asymmetric local branching constraint $\sum_{(l,v)\in S}(1 - x_v^l) \leq k$.

Initially, $k := 10$ and a B&B node limit of 10000 and time limit of 10 seconds are imposed. If within the current neighborhood no improving solution is found for the given node and time limit, k is increased by 5. The procedure is repeated as long as $k \leq 20$. As soon as an improving solution has been found, the B&B is restarted, and k is reset to its initial value. In each B&B node the proposed primal heuristic is executed. Only integer solutions are cut off, i.e., inequalities are only separated when the current LP solution is integral. The cuts separated are gathered in a cut pool and added to the model for subsequent iterations.

At the beginning of the resolution process, the previously described primal heuristic is used to produce a starting solution, using the original delay values.

3 Benders-based Formulation

3.1 Decomposable Formulation

The USP problem embodies the typical structure of a two-stage like problem; in a first stage one would decide over the upgrading scheme, and on a second stage one would define the corresponding shortest paths. Therefore, the USP becomes a natural candidate to be solved via Benders Decomposition: the master problem decides over the values of vector $\mathbf{x}$ (upgrading decisions); this solution is then used as parameter when solving the corresponding slave problems (shortest paths) whose solutions are mapped back in the master in the form of so-called (optimality) *Benders* cuts.

Considering the definition of variables presented before, the master problem is given by

$$\text{(BF)}\ \min \left\{ \sum_{p\in P} \theta_p \mid \boldsymbol{\theta} \geq \boldsymbol{\Phi}(\mathbf{x}, P),\ \text{(C.3)},\ \mathbf{x} \in \{0,1\}^{|V|\times|L|} \text{ and } \boldsymbol{\theta} \in \mathbb{R}_{\geq 0}^{|P|} \right\}, \qquad \text{(MP)}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{|P|}_{\geq 0}$ corresponds to a set of $|P|$ auxiliary variables, where each of them serves as a surrogate of the lower-bound, given by $\Phi(\mathbf{x}, p)$, of corresponding p-th path.

Recall the graph transformation described before for the separation of connectivity cuts for the linear case. For an optimal solution, say $\mathbf{x}^*$, of (MP), and a given path $(s,t) = p \in P$, the underlying slave problem corresponds to

$$\text{(BF-Sub)} \quad \Phi(\mathbf{x}^*, p) = \min \sum_{v \in V|_{v \neq s,t}} \left(d_v \bar{y}_v + \sum_{l \in L} d_v^l \bar{z}_v^l \right) \tag{SP.1}$$

$$\text{s.t} \qquad \bar{z}_v^l \leq x_v^{l*} \qquad \forall l \in L,\ \forall v \in V \tag{SP.2}$$

$$\sum_{e \in \delta^-(s^-)} f_{pe} = 0 \ \text{ and } \ \sum_{e \in \delta^+(s^+)} f_{pe} = 1 \tag{SP.3}$$

$$\sum_{e \in \delta^-(t^-)} f_{pe} = 1 \ \text{ and } \ \sum_{e \in \delta^+(t^+)} f_{pe} = 0 \tag{SP.4}$$

$$\sum_{e \in \delta^-(v^-)} f_{pe} = \bar{y}_v + \sum_{l \in L} \bar{z}_v^l \ \text{ and } \ \sum_{e \in \delta^+(v^+)} f_{pe} = \bar{y}_v + \sum_{l \in L} \bar{z}_v^l,\ \forall v \in V \setminus \{s,t\} \tag{SP.5}$$

$$(\bar{\mathbf{y}}, \bar{\mathbf{z}}) \in \{0,1\}^{|V|+|V|\times|L|} \ \text{ and } \mathbf{f} \in [0,1]^{|A'|}, \tag{SP.6}$$

where $\mathbf{f} \in [0,1]^{|A'|}$ is a set of flow variables that enable to model an s,t-path on G' and associate the corresponding delay values in the objective function.

The algorithmic scheme designed on the basis of this decomposable formulation will be outlined in detail in §3.2.

3.2 Benders Decomposition

In the following, the generation of Benders cuts and the details of the implemented stabilization procedure are described. Note that in this paper the Benders decomposition has been implemented within a B&C framework.

Benders Cuts: Fractional and Integer Case Due to the structure of the above presented formulation, it holds that the slave problem is always feasible for any master solution; therefore the generated cuts are then regarded as *optimality* cuts.

The separation of Benders cuts depends on whether the current master solution $\tilde{\mathbf{x}}$ is integer or not. If $\tilde{\mathbf{x}}$ is fractional, the corresponding slave problem (SP.1)-(SP.6) is solved as a linear problem and the dual multipliers are then used to build the cut.

Note that this decomposition scheme falls within the general scheme for solving fixed-charged (uncapacitated) network design problems (see Magnanti et al., 1986; Costa, 2005, for further details). In particular, for integer $\tilde{\mathbf{x}}$, the subproblem for a $p \in P$ reduces to a shortest-path problem in the graph induced by the upgrades selected in $\tilde{\mathbf{x}}$. Let $SP_p(\tilde{\mathbf{x}})$ be the value of a shortest-path in this graph and for each $l \in L$, let $S^l = \{v \in V \mid \tilde{x}_v^l = 1\}$. If for a given $p \in P$ it holds that

$\tilde{\theta}_p < SP_p(\tilde{\mathbf{x}}, p)$, the following inequality cuts off the current integer point,

$$\theta_p \geq SP_p(\tilde{\mathbf{x}}, p) - \sum_{l \in L} \sum_{v \notin S^l} (d_v - d_v^l) x_v^l. \tag{CC}$$

The validity of (CC) can be explained as follows. Clearly, removing any node from S^l cannot improve the value of the shortest path. Moreover, adding a node v to some S^l may improve the value $SP_p(\tilde{\mathbf{x}}, p)$ obtained with the currently selected updates, but the improvement is bounded by $(d_v - d_v^l)$.

Stabilization Benders decomposition frequently exhibits a strong tailing-off effect, i.e., cutting planes get significantly less effective as the lower bound increases. A possible strategy to address this issue is to include some form of stabilization into the performed separation scheme. In the proposed implementation, a simple stabilization procedure similar to the in-out-method (see Ben-Ameur and Neto, 2007; Fischetti and Salvagnin, 2010) is applied at the root node. Instead of performing separation for the (optimal) master LP solution $\tilde{\mathbf{x}}$, a separation point $\mathbf{x}_{sep}$ is computed as linear combination between a stabilization point $\bar{\mathbf{x}}$ and the optimal LP solution $\tilde{\mathbf{x}}$.

For $\bar{\mathbf{x}}$, the vector $\mathbf{1}^n$ is used. The separation point $\mathbf{x}_{sep}$ is computed as $\mathbf{x}_{sep} := \gamma\ \bar{\mathbf{x}} + (1 - \gamma)\ \tilde{\mathbf{x}}$, for some $\gamma \in (0.1, 1]$. In each cutting plane iteration, separation points are iteratively generated until the generated point is violated. The parameter γ is chosen in the form of a binary-search, approaching $\tilde{\mathbf{x}}$ with each iteration. If no violated point is found within five iterations, the stabilization procedure is terminated and separation is performed for $\tilde{\mathbf{x}}$.

When performing separation based on $\tilde{\mathbf{x}}$, adding a small ε to $\tilde{\mathbf{x}}$ improves the strength of cuts. However, during the final cutting plane iterations, this approach may lead to numerical difficulties, so the ε is removed once the lower bound increase between iterations is below a fixed threshold. If the removal of ε does not decrease tailing-off, the cut-loop is terminated and branching is performed. After the root node, separation of optimality cuts is performed without stabilization and the number of cutting plane iterations is limited to five per B&B node.

Primal Heuristic As for the B&C approach, a scheme for generating primal (master) feasible solutions is embedded into the Benders decomposition. This scheme is basically equivalent to the one designed for the B&C: the master (optimal) solution $\tilde{\mathbf{x}}$ is used for computing a vector of delays $\tilde{\mathbf{d}}$ (Step 1), which are then used to compute a new feasible vector $\check{\mathbf{x}}$ along with values $\check{\theta}^p$ (the value of the corresponding p-th shortest path). The pair $(\check{\mathbf{x}}, \check{\boldsymbol{\theta}})$ is therefore a candidate of a new incumbent solution.

4 Computational Results

Experimental Setting The algorithmic schemes described in §2 and §3 have been implemented in C++ using the CPLEX 12.6 Concert framework. All experiments have been performed on an Intel Xeon CPU with 2.5 GHz and 20

cores (only one core is used per run). A fixed memory limit of 16 GB and a time limit of 1800 seconds have been imposed. The budget B has been set to βB_{max}, where B_{max} corresponds to the total budget necessary to achieve the shortest delay possible. Three different budget configurations have been tested, i.e., $\beta \in \{0.1, 0.25, 0.5\}$.

Benchmark Instances Two types of instances are considered. The first type corresponds to `N×N` grid graphs (see Dilkina et al., 2011). The second type are random instances generated by the following scheme: Given the number of nodes, arcs are placed randomly between nodes until a specified density $\alpha = |E|/|V|$ is reached and the graph is connected. For each type, set P is defined by randomly selecting $|P|$ pairs of nodes.

For both types of instances, the upgrade costs $\mathbf{c}$ have been chosen uniformly at random from the range $[50, 1000]$. For defining the delay of upgraded nodes, the following schemes have been considered (Dilkina et al., 2011): (i) **Scaled** – each upgraded delay value is set to $d'_v = cd_v$, $\forall v \in V$, where $c \in [0, 1]$ *scales* d_v. For experiments, c has been set to 0.1, 0.5 and 0.9. (ii) **Constant** – each upgraded delay value $d'_v = 50$. (iii) **Tiered** – for $d_v \in (500, 1000]$, $d'_v = 500$, for $d_v \in (100, 500]$, $d'_v = 75$, and for $d_v \in [50, 100]$, $d'_v = 50$. Thus in total five upgrading schemes are considered: *Scaled=0.1*, *Scaled=0.5*, *Scaled=0.9*, *Constant*, *Tiered*.

One grid instance has been generated for every combination between upgrading schemes, graph size `N` $\in \{20, 30\}$ and number of paths $|P| \in \{5, 10, 20, 40\}$ (40 grid instances). Similarly, one dense instance has been generated for every combination between upgrading schemes, number of nodes $|V| \in \{1000, 2000\}$ and graph density $\alpha \in \{4, 8, 6, 32\}$, with a fixed number of paths $|P| = 20$ (40 dense instances).

4.1 Algorithmic Performance

First, experiments are reported which measure the average effect of all implemented algorithmic components separately, i.e., the stabilization procedure, primal heuristics and preprocessing. Afterwards, a detailed comparison of the algorithmic strategies is given. These strategies include B&C algorithms based on the proposed Benders formulation (BF) and cut formulation based on node separators (NODE). As a third strategy the multi-commodity flow formulation (MCF) proposed in (Dilkina et al., 2011) is considered.

Table 1 shows the average influence of the implemented stabilization procedure. For each considered budget slack β, all instances have been run once with and without stabilization. Columns $t'_R(s)$ and $t_R(s)$ display the average root relaxation solution time for formulation (BF), with and without stabilization, respectively. The column $g_R(\%)$ lists the average root gap to the best known solution. Results show that the speedup increases with the value of β, reaching one order of magnitude for $\beta = 0.5$. The gap of the root relaxation is on average already close to the optimum, suggesting that both (MCF) and its Benders reformulation (BF) achieve high-quality bounds.

Table 1. Comparison of the average root relaxation solution time for formulation (BF) with (t_R) and without stabilization (t'_R).

β	t_R(s)	t'_R(s)	g_R(%)
0.1	21.49	19.88	1.59
0.25	58.19	44.62	3.64
0.5	573.48	23.43	0.69

Table 2 compares the influence of primal heuristics. For each considered budget slack β, all instances have been run once with and without primal heuristics. The columns compare running time and gap for each configuration. Average results are reported only for formulations (NODE) and (BF). For formulation (MCF), the implemented primal heuristics did not manage to outperform the default CPLEX heuristics, and were thus switched off for (MCF) in all subsequent runs. The results show that the primal heuristics play a crucial role for formulations (BF) and (NODE), where less information is available for the LP solver to exploit than for (MCF). Note that since the implemented local branching heuristic is potentially very time-consuming, as the exploration of neighborhoods involves the solution of LPs, in our implementation it is only applied once after solving the root relaxation.

Table 2. Average influence of primal heuristics on running time and gap.

	$\beta = 0.1$		$\beta = 0.25$		$\beta = 0.5$	
	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)
W/O HEUR.	535	5.51	600	8.93	470	4.47
HEUR.	391	2.68	451	4.70	365	2.04

As a preprocessing step the same procedure as proposed in (Dilkina et al., 2011) is implemented, which can be directly incorporated into (BF) and (NODE). On average, the percentage of fixed node variables is 2.82% (constant), 4.64% (scaled=0.1), 25.28% (scaled=0.5), 57.24% (scaled=0.9) and 24.55% (tiered). The results show that the preprocessing is not very effective for the delay types which were established as difficult in (Dilkina et al., 2011).

Tables 3 and 4 compare the algorithms' performance for $|L| = 1$. For each setting both the running time (in seconds, columns "t(s)") and optimality gap (in percent, columns "g(%)") are reported. All results are partitioned based on budget slack β and delay structure (scaled, tiered, constant). If a run exceeds its time limit, the corresponding time column contains *TL*. If for a run the formulation's root relaxation could not be solved within the time limit, the gap column contains "-". For each configuration the best results are marked in bold.

Table 3. Test results on grid graphs.

	$\beta = 0.1$						$\beta = 0.25$						$\beta = 0.5$					
	MCF		BF		NODE		MCF		BF		NODE		MCF		BF		NODE	
$\|P\|$	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)
N=20 × 20, *Constant*																		
5	**4**	**0.0**	**4**	**0.0**	26	0.0	**16**	**0.0**	159	0.0	143	0.0	**6**	**0.0**	229	0.0	28	0.0
10	**5**	**0.0**	**5**	**0.0**	21	0.0	**30**	**0.0**	76	0.0	63	0.0	**1**	**0.0**	7	0.0	12	0.0
20	271	0.0	**266**	**0.0**	*TL*	0.1	**741**	**0.0**	*TL*	5.8	*TL*	17.1	**146**	**0.0**	254	0.0	549	0.0
40	***TL***	**10.6**	*TL*	18.5	*TL*	24.0	***TL***	**15.0**	*TL*	22.3	*TL*	41.3	**693**	**0.0**	*TL*	0.8	*TL*	5.5
N=20 × 20, *Scaled=0.1*																		
5	**2**	**0.0**	4	0.0	20	0.0	**14**	**0.0**	72	0.0	187	0.0	**10**	**0.0**	61	0.0	44	0.0
10	**10**	**0.0**	12	0.0	59	0.0	**52**	**0.0**	178	0.0	1363	0.0	**20**	**0.0**	90	0.0	78	0.0
20	184	0.0	**112**	**0.0**	1279	0.0	**1662**	**0.0**	*TL*	0.5	*TL*	22.1	**197**	**0.0**	530	0.0	1692	0.0
40	*TL*	5.7	***TL***	**4.1**	*TL*	24.2	*TL*	18.1	***TL***	**15.8**	*TL*	43.7	***TL***	**1.6**	*TL*	6.1	*TL*	31.3
N=20 × 20, *Scaled=0.5*																		
5	**2**	**0.0**	**2**	**0.0**	9	0.0	3	0.0	**2**	**0.0**	16	0.0	**2**	**0.0**	**2**	**0.0**	7	0.0
10	2	0.0	**1**	**0.0**	18	0.0	**2**	**0.0**	3	0.0	22	0.0	**1**	**0.0**	**1**	**0.0**	15	0.0
20	9	0.0	**5**	**0.0**	93	0.0	**6**	**0.0**	**6**	**0.0**	74	0.0	29	0.0	**28**	**0.0**	117	0.0
40	310	0.0	**42**	**0.0**	*TL*	0.0	328	0.0	**124**	**0.0**	1736	0.0	215	0.0	**197**	**0.0**	1485	0.0
N=20 × 20, *Scaled=0.9*																		
5	**1**	**0.0**	**1**	**0.0**	**1**	**0.0**	**1**	**0.0**	**1**	**0.0**	2	0.0	**0**	**0.0**	**0**	**0.0**	2	0.0
10	**1**	**0.0**	**1**	**0.0**	3	0.0	**1**	**0.0**	**1**	**0.0**	2	0.0	**1**	**0.0**	**1**	**0.0**	2	0.0
20	2	0.0	**1**	**0.0**	10	0.0	2	0.0	**1**	**0.0**	10	0.0	**3**	**0.0**	**3**	**0.0**	8	0.0
40	5	0.0	**2**	**0.0**	53	0.0	6	0.0	**3**	**0.0**	45	0.0	5	0.0	**4**	**0.0**	31	0.0
N=20 × 20, *Tiered*																		
5	**1**	**0.0**	**1**	**0.0**	7	0.0	**2**	**0.0**	**2**	**0.0**	10	0.0	**2**	**0.0**	**2**	**0.0**	10	0.0
10	2	0.0	**1**	**0.0**	7	0.0	**1**	**0.0**	**1**	**0.0**	7	0.0	**1**	**0.0**	**1**	**0.0**	13	0.0
20	86	0.0	**27**	**0.0**	417	0.0	20	0.0	**12**	**0.0**	154	0.0	26	0.0	**17**	**0.0**	106	0.0
40	333	0.0	**72**	**0.0**	*TL*	1.1	372	0.0	**54**	**0.0**	*TL*	0.1	29	0.0	**20**	**0.0**	438	0.0
N=30 × 30, *Constant*																		
5	**16**	**0.0**	24	0.0	59	0.0	**33**	**0.0**	203	0.0	74	0.0	**62**	**0.0**	*TL*	3.2	138	0.0
10	**663**	**0.0**	*TL*	8.6	*TL*	17.0	**1492**	**0.0**	*TL*	33.5	*TL*	36.1	**192**	**0.0**	*TL*	17.1	857	0.0
20	***TL***	**1.8**	*TL*	11.1	*TL*	24.2	*TL*	66.1	***TL***	**31.1**	*TL*	46.2	***TL***	**0.2**	*TL*	5.1	*TL*	12.8
40	*TL*	-	***TL***	**33.0**	*TL*	48.2	*TL*	75.0	***TL***	**41.1**	*TL*	60.8	***TL***	**5.4**	*TL*	10.8	*TL*	20.4
N=30 × 30, *Scaled=0.1*																		
5	**5**	**0.0**	8	0.0	10	0.0	**7**	**0.0**	13	0.0	27	0.0	**5**	**0.0**	19	0.0	23	0.0
10	**55**	**0.0**	86	0.0	1778	0.0	**1089**	**0.0**	*TL*	1.9	*TL*	6.6	**648**	**0.0**	*TL*	6.0	*TL*	37.2
20	***TL***	**1.9**	*TL*	3.0	*TL*	15.8	*TL*	66.6	***TL***	**34.5**	*TL*	44.9	***TL***	**20.8**	*TL*	25.8	*TL*	48.0
40	*TL*	-	***TL***	**29.2**	*TL*	44.3	*TL*	74.8	***TL***	**49.7**	*TL*	62.1	*TL*	64.4	***TL***	**28.5**	*TL*	50.4
N=30 × 30, *Scaled=0.5*																		
5	**4**	**0.0**	5	0.0	49	0.0	**12**	**0.0**	17	0.0	55	0.0	**14**	**0.0**	46	0.0	56	0.0
10	5	0.0	**3**	**0.0**	24	0.0	13	0.0	**6**	**0.0**	42	0.0	**6**	**0.0**	**6**	**0.0**	24	0.0
20	381	0.0	**36**	**0.0**	1632	0.0	1687	0.0	**316**	**0.0**	*TL*	0.3	*TL*	0.1	**1209**	**0.0**	*TL*	0.6
40	*TL*	0.0	**270**	**0.0**	*TL*	5.4	*TL*	0.2	**546**	**0.0**	*TL*	9.9	***TL***	**0.0**	***TL***	**0.0**	*TL*	0.4
N=30 × 30, *Scaled=0.9*																		
5	**1**	**0.0**	**1**	**0.0**	**1**	**0.0**	**1**	**0.0**	**1**	**0.0**	2	0.0	**1**	**0.0**	**1**	**0.0**	**1**	**0.0**
10	**5**	**0.0**	**5**	**0.0**	70	0.0	9	0.0	**4**	**0.0**	84	0.0	9	0.0	**4**	**0.0**	68	0.0
20	8	0.0	**7**	**0.0**	95	0.0	7	0.0	**4**	**0.0**	96	0.0	**4**	**0.0**	10	0.0	80	0.0
40	69	0.0	**53**	**0.0**	*TL*	0.0	90	0.0	**32**	**0.0**	720	0.0	113	0.0	**27**	**0.0**	936	0.0
N=30 × 30, *Tiered*																		
5	2	0.0	**1**	**0.0**	3	0.0	**1**	**0.0**	**1**	**0.0**	3	0.0	**1**	**0.0**	**1**	**0.0**	3	0.0
10	28	0.0	**19**	**0.0**	256	0.0	**13**	**0.0**	21	0.0	129	0.0	**7**	**0.0**	8	0.0	99	0.0
20	73	0.0	**32**	**0.0**	*TL*	0.5	228	0.0	**80**	**0.0**	*TL*	0.0	**70**	**0.0**	79	0.0	1635	0.0
40	378	0.0	**65**	**0.0**	*TL*	8.8	**727**	**0.0**	1034	0.0	*TL*	2.3	144	0.0	**62**	**0.0**	*TL*	1.6

Table 4. Test results on dense graphs.

	$\beta = 0.1$						$\beta = 0.25$						$\beta = 0.5$					
	MCF		BF		NODE		MCF		BF		NODE		MCF		BF		NODE	
$\lvert E\rvert/\lvert V\rvert$	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)
$\lvert V\rvert = 1000$, $\lvert P\rvert = 20$, *Constant*																		
4	**255**	**0.0**	372	0.0	1277	0.0	1205	0.0	**544**	**0.0**	710	0.0	65	0.0	**36**	**0.0**	46	0.0
8	*TL*	1.3	537	0.0	**366**	**0.0**	173	0.0	444	0.0	**112**	**0.0**	112	0.0	1057	0.0	**37**	**0.0**
16	*TL*	0.4	855	0.0	**307**	**0.0**	815	0.0	404	0.0	**108**	**0.0**	81	0.0	63	0.0	**12**	**0.0**
32	797	0.0	811	0.0	**42**	**0.0**	460	0.0	415	0.0	**25**	**0.0**	118	0.0	87	0.0	**12**	**0.0**
$\lvert V\rvert = 1000$, $\lvert P\rvert = 20$, *Scaled=0.1*																		
4	185	0.0	**84**	**0.0**	225	0.0	**514**	**0.0**	670	0.0	*TL*	1.2	379	0.0	**320**	**0.0**	328	0.0
8	102	0.0	63	0.0	**33**	**0.0**	225	0.0	168	0.0	**86**	**0.0**	381	0.0	330	0.0	**83**	**0.0**
16	62	0.0	73	0.0	**9**	**0.0**	101	0.0	103	0.0	**11**	**0.0**	83	0.0	138	0.0	**14**	**0.0**
32	51	0.0	108	0.0	**4**	**0.0**	123	0.0	104	0.0	**8**	**0.0**	118	0.0	81	0.0	**6**	**0.0**
$\lvert V\rvert = 1000$, $\lvert P\rvert = 20$, *Scaled=0.5*																		
4	6	0.0	**2**	**0.0**	10	0.0	20	0.0	**6**	**0.0**	17	0.0	39	0.0	**13**	**0.0**	18	0.0
8	23	0.0	**8**	**0.0**	10	0.0	22	0.0	**7**	**0.0**	13	0.0	13	0.0	**3**	**0.0**	11	0.0
16	14	0.0	6	0.0	**5**	**0.0**	51	0.0	**8**	**0.0**	12	0.0	46	0.0	**9**	**0.0**	16	0.0
32	38	0.0	20	0.0	**5**	**0.0**	146	0.0	31	0.0	**8**	**0.0**	241	0.0	53	0.0	**10**	**0.0**
$\lvert V\rvert = 1000$, $\lvert P\rvert = 20$, *Scaled=0.9*																		
4	10	0.0	**2**	**0.0**	13	0.0	11	0.0	**3**	**0.0**	13	0.0	6	0.0	**1**	**0.0**	6	0.0
8	15	0.0	**3**	**0.0**	10	0.0	20	0.0	10	0.0	**8**	**0.0**	24	0.0	**8**	**0.0**	**8**	**0.0**
16	21	0.0	**3**	**0.0**	5	0.0	25	0.0	**4**	**0.0**	5	0.0	20	0.0	**4**	**0.0**	**4**	**0.0**
32	34	0.0	8	0.0	**4**	**0.0**	42	0.0	7	0.0	**4**	**0.0**	43	0.0	8	0.0	**4**	**0.0**
$\lvert V\rvert = 1000$, $\lvert P\rvert = 20$, *Tiered*																		
4	23	0.0	**11**	**0.0**	40	0.0	63	0.0	**21**	**0.0**	31	0.0	26	0.0	27	0.0	**20**	**0.0**
8	29	0.0	**22**	**0.0**	23	0.0	39	0.0	**22**	**0.0**	**22**	**0.0**	76	0.0	32	0.0	**25**	**0.0**
16	13	0.0	9	0.0	**6**	**0.0**	38	0.0	10	0.0	**6**	**0.0**	17	0.0	7	0.0	**5**	**0.0**
32	70	0.0	26	0.0	**10**	**0.0**	53	0.0	29	0.0	**6**	**0.0**	71	0.0	8	0.0	**4**	**0.0**
$\lvert V\rvert = 2000$, $\lvert P\rvert = 20$, *Constant*																		
4	**977**	**0.0**	*TL*	4.1	*TL*	1.8	*TL*	60.9	***TL***	**16.0**	*TL*	33.7	**983**	**0.0**	1001	0.0	1466	0.0
8	*TL*	7.9	*TL*	7.3	***TL***	**5.0**	***TL***	**3.6**	*TL*	7.3	*TL*	26.7	494	0.0	439	0.0	**83**	**0.0**
16	*TL*	-	***TL***	**17.2**	*TL*	29.2	*TL*	64.5	*TL*	8.2	**1792**	**0.0**	291	0.0	510	0.0	**42**	**0.0**
32	*TL*	-	*TL*	19.3	**776**	**0.0**	*TL*	56.7	*TL*	9.4	**273**	**0.0**	1486	0.0	1466	0.0	**96**	**0.0**
$\lvert V\rvert = 2000$, $\lvert P\rvert = 20$, *Scaled=0.1*																		
4	**60**	**0.0**	127	0.0	69	0.0	**276**	**0.0**	518	0.0	303	0.0	447	0.0	*TL*	0.2	**210**	**0.0**
8	85	0.0	139	0.0	**30**	**0.0**	222	0.0	383	0.0	**69**	**0.0**	484	0.0	973	0.0	**133**	**0.0**
16	560	0.0	912	0.0	**125**	**0.0**	563	0.0	520	0.0	**86**	**0.0**	380	0.0	542	0.0	**38**	**0.0**
32	98	0.0	261	0.0	**14**	**0.0**	385	0.0	403	0.0	**31**	**0.0**	893	0.0	1047	0.0	**102**	**0.0**
$\lvert V\rvert = 2000$, $\lvert P\rvert = 20$, *Scaled=0.5*																		
4	12	0.0	**5**	**0.0**	35	0.0	28	0.0	**7**	**0.0**	36	0.0	31	0.0	**13**	**0.0**	47	0.0
8	86	0.0	**21**	**0.0**	60	0.0	64	0.0	**30**	**0.0**	67	0.0	72	0.0	**24**	**0.0**	58	0.0
16	81	0.0	29	0.0	**23**	**0.0**	48	0.0	20	0.0	**19**	**0.0**	77	0.0	26	0.0	**25**	**0.0**
32	70	0.0	42	0.0	**19**	**0.0**	324	0.0	148	0.0	**76**	**0.0**	298	0.0	63	0.0	**17**	**0.0**
$\lvert V\rvert = 2000$, $\lvert P\rvert = 20$, *Scaled=0.9*																		
4	24	0.0	**6**	**0.0**	15	0.0	10	0.0	**2**	**0.0**	21	0.0	33	0.0	**14**	**0.0**	18	0.0
8	26	0.0	**5**	**0.0**	30	0.0	49	0.0	**12**	**0.0**	21	0.0	24	0.0	**7**	**0.0**	27	0.0
16	45	0.0	**7**	**0.0**	15	0.0	21	0.0	**6**	**0.0**	12	0.0	53	0.0	**11**	**0.0**	22	0.0
32	70	0.0	**15**	**0.0**	28	0.0	96	0.0	16	0.0	**13**	**0.0**	108	0.0	27	0.0	**22**	**0.0**
$\lvert V\rvert = 2000$, $\lvert P\rvert = 20$, *Tiered*																		
4	43	0.0	**41**	**0.0**	149	0.0	131	0.0	**115**	**0.0**	239	0.0	**41**	**0.0**	58	0.0	84	0.0
8	110	0.0	**42**	**0.0**	62	0.0	142	0.0	109	0.0	**82**	**0.0**	569	0.0	412	0.0	**193**	**0.0**
16	62	0.0	32	0.0	**20**	**0.0**	76	0.0	43	0.0	**24**	**0.0**	246	0.0	72	0.0	**46**	**0.0**
32	55	0.0	38	0.0	**11**	**0.0**	118	0.0	47	0.0	**12**	**0.0**	63	0.0	37	0.0	**7**	**0.0**

The results in Table 3 compare scalability with respect to the number of paths on grid graphs 20×20 and 30×30. For $\beta = 0.1$, the performance of (BF) is best for delay structures scaled=0.5, scaled=0.9 and tiered, where (MCF) is outperformed even for small values of $|P|$. For delay structures scaled=0.1 and constant, the performance is more erratic, and (MCF) frequently achieves comparable or better performance even for $|P| = 40$.

For higher budgets, (BF) also tends to perform worse in general. As already observed by (Dilkina et al., 2011), delay structures constant and scaled=0.1 are more difficult for (MCF), and this also holds for (BF). The worse performance of (BF) for the aforementioned configurations can be explained by the fact that in these cases far more optimality cuts are generated, which in turn slow down the solution of the master problem. For (NODE), which clearly performs worst in most observed cases, a similar problem occurs. Here a large number of cuts is required to enforce connectivity on extremely sparse grid graphs. The high difficulty of this instance type for algorithms based on branch-and-cut is also known for similar problems, e.g., the Steiner tree problem, where large-scale grid graphs remain challenging even for state-of-the-art approaches (see, e.g., Polzin, 2003).

In Table 4, results on dense graphs with varying values for $|E|/|V|$ are reported. Here (MCF) only manages to outperform other approaches on instances with constant delay structure which are relatively sparse. For all higher densities, (MCF) quickly becomes less practical, and is outperformed both by (BF) and (NODE). Here (NODE) clearly performs best, and is less affected by different delay structures and budget slacks. The performance of (BF) is similar to (NODE) except for delay structures constant and scaled=0.1.

4.2 Multiple Upgrades

In this section the case $|L| > 1$ is explored. For this purpose grid and dense graphs with three upgrade levels have been constructed based on the scaled delay structure, i.e., for each node there exist three possible upgrades using 0.1, 0.5 and 0.9 as scaling factor. Again costs are computed randomly in the range of $[50, 1000]$, but are assigned to upgrades per node such that the upgrade with the lowest delay is assigned the highest cost.

Table 5 reports results for grid graphs with varying values of $|P|$. The results show that on the grid graphs (MCF) performs best for the considered weights. Only for low budgets and a high number of paths becomes (BF) more competitive. Table 6 reports results for dense graphs with varying values of $|E|/|V|$. As for $|L| = 1$, (NODE) performs best on average, only being outperformed by (MCF) for graphs of lowest density. For higher densities, the root relaxation of (MCF) cannot be solved within the time limit.

Table 5. Test results on grid graphs with three upgrade types per node.

	$\beta = 0.1$						$\beta = 0.25$						$\beta = 0.5$					
	MCF		BF		NODE		MCF		BF		NODE		MCF		BF		NODE	
$\|P\|$	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)
$N = 20$, $\|P\| = 20$, $\|L\| = 3$, *Scaled*={*0.1,0.5,0.9*}																		
5	**21**	**0.0**	237	0.0	173	0.0	**81**	**0.0**	*TL*	5.3	804	0.0	**23**	**0.0**	*TL*	3.3	152	0.0
10	**15**	**0.0**	16	0.0	192	0.0	78	0.0	**52**	**0.0**	84	0.0	**12**	**0.0**	46	0.0	44	0.0
20	***TL***	**5.2**	*TL*	7.8	*TL*	16.8	***TL***	**12.2**	*TL*	27.5	*TL*	44.4	**176**	**0.0**	1035	0.0	*TL*	0.1
40	*TL*	54.7	***TL***	**5.9**	*TL*	31.8	***TL***	**9.9**	*TL*	35.6	*TL*	51.6	**397**	**0.0**	540	0.0	*TL*	0.1
$N = 30$, $\|P\| = 20$, $\|L\| = 3$, *Scaled*={*0.1,0.5,0.9*}																		
5	**14**	**0.0**	35	0.0	197	0.0	**34**	**0.0**	459	0.0	496	0.0	**27**	**0.0**	*TL*	13.4	201	0.0
10	***TL***	**7.5**	*TL*	17.9	*TL*	23.8	***TL***	**16.0**	*TL*	49.0	*TL*	53.4	**250**	**0.0**	*TL*	2.2	1365	0.0
20	***TL***	**13.4**	*TL*	14.6	*TL*	29.6	***TL***	**24.8**	*TL*	40.9	*TL*	42.5	**431**	**0.0**	*TL*	5.8	*TL*	3.0
40	*TL*	-	***TL***	**41.0**	*TL*	63.9	*TL*	-	***TL***	**53.9**	*TL*	67.5	***TL***	**6.0**	*TL*	13.7	*TL*	32.4

Table 6. Test results on dense graphs with three upgrade types per node.

	$\beta = 0.1$						$\beta = 0.25$						$\beta = 0.5$					
	MCF		BF		NODE		MCF		BF		NODE		MCF		BF		NODE	
$\|E\|/\|V\|$	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)	t(s)	g(%)
$\|V\| = 1000$, $\|P\| = 20$, $\|L\| = 3$, *Scaled*={*0.1,0.5,0.9*}																		
4	526	0.0	**405**	**0.0**	726	0.0	**94**	**0.0**	235	0.0	208	0.0	**608**	**0.0**	1100	0.0	1471	0.0
8	1346	0.0	809	0.0	**578**	**0.0**	1359	0.0	920	0.0	**805**	**0.0**	212	0.0	94	0.0	**38**	**0.0**
16	*TL*	4.5	*TL*	4.3	**1528**	**0.0**	*TL*	4.6	1558	0.0	**1275**	**0.0**	279	0.0	159	0.0	**31**	**0.0**
32	*TL*	23.6	1369	0.0	**290**	**0.0**	693	0.0	594	0.0	**77**	**0.0**	95	0.0	120	0.0	**21**	**0.0**
$\|V\| = 2000$, $\|P\| = 20$, $\|L\| = 3$, *Scaled*={*0.1,0.5,0.9*}																		
4	1164	0.0	966	0.0	**398**	**0.0**	*TL*	14.7	***TL***	**13.5**	*TL*	36.4	952	0.0	1106	0.0	**709**	**0.0**
8	*TL*	-	***TL***	**21.7**	*TL*	23.1	*TL*	-	***TL***	**22.5**	*TL*	40.4	868	0.0	1648	0.0	**510**	**0.0**
17	*TL*	-	*TL*	4.5	**878**	**0.0**	*TL*	-	*TL*	28.3	***TL***	**26.3**	*TL*	0.3	1450	0.0	**336**	**0.0**
32	*TL*	-	*TL*	7.3	**1519**	**0.0**	*TL*	-	*TL*	31.2	***TL***	**9.8**	756	0.0	843	0.0	**105**	**0.0**

5 Conclusions and Future Work

In this paper, algorithmic expedients along with computational results are presented for the Upgrading Shortest Path Problem (USP). The USP is a recently proposed network optimization problem that enables to model a variety of decision making problems where the goal is to optimize the effectiveness of the sought network while respecting a design budget.

The proposed algorithms show to be effective for quite large instances, being able to reach rather small optimality gaps, within reasonable computing times, for instances with medium to large sizes. Moreover, these tailored strategies outperform the use of a compact formulation even if medium size instances are considered.

Acknowledgements E. Álvarez-Miranda is supported by the Chilean Council of Scientific and Technological Research through the grant FONDECYT N.11140060 and through the Complex Engineering Systems Institute (ICM:P-05-004-F, CONICYT:FBO16). M. Sinnl is supported by the Austrian Research Fund (FWF, Project P 26755-N19). M. Luipersbeck acknowledges the support of the University of Vienna through the uni:docs fellowship programme.

Bibliography

E. Álvarez-Miranda, I. Ljubić, and P. Mutzel. The rooted maximum node-weight connected subgraph problem. In C. Gomes and M. Sellmann, editors, *Proceedings of CPAIOR 2013*, volume 7874 of *LNCS*, pages 300–315. Springer, 2013.

W. Ben-Ameur and J. Neto. Acceleration of cutting-plane and column generation algorithms: Applications to network design. *Networks*, 49(1):3–17, 2007.

A. Campbell, T. Lowe, and L. Zhang. Upgrading arcs to minimize the maximum travel time in a network. *Networks*, 47(2):72–80, 2006.

A. Costa. A survey on benders decomposition applied to fixed-charge network design problems. *Computers & OR*, 32(6):1429–1450, 2005.

B. Dilkina and C. Gomes. Solving connected subgraph problems in wildlife conservation. In A. Lodi, M. Milano, and P. Toth, editors, *Proceedings of CPAIOR 2010*, volume 6140 of *LNCS*, pages 102–116. Springer, 2010.

B. Dilkina, J. Lai, and C. Gomes. Upgrading shortest paths in networks. In T. Achterberg and C. Beck, editors, *Proceedings of CPAIOR 2011*, volume 6697 of *LNCS*, pages 76–91. Springer, 2011.

M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98 (1–3):23–47, 2003.

M. Fischetti and D. Salvagnin. An in-out approach to disjunctive optimization. In A. Lodi, M. Milano, and P. Toth, editors, *Proceedings of CPAIOR 2010*, volume 6140 of *LNCS*, pages 102–116. Springer, 2010.

M. Fischetti, M. Leitner, I. Ljubić, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, and M. Sinnl. Thinning out Steiner trees: A node-based model for uniform edge costs. *Workshop of the 11th DIMACS Implementation Challenge*, 2014.

J. Fournier. *Optimal Paths*, pages 119–147. ISTE, 2010.

W. D Harvey and M. L. Ginsberg. Limited discrepancy search. In *IJCAI (1)*, pages 607–615, 1995.

S. Krumke, M. Marathe, H. Noltemeier, R. Ravi, S. Ravi, R. Sundaram, and H. Wirth. Improving minimum cost spanning trees by upgrading nodes. *Journal of Algorithms*, 33(1):92–111, 1999.

T. Magnanti, P. Mireault, and R. Wong. Tailoring benders decomposition for uncapacitated network design. In G. Gallo and C. Sandi, editors, *Netflow at Pisa*, volume 26 of *Mathematical Programming Studies*, pages 112–154. Springer Berlin Heidelberg, 1986.

D. Paik and S. Sahni. Network upgrading problems. *Networks*, 26(1):45–58, 1995.

T. Polzin. *Algorithms for the Steiner problem in networks*. PhD thesis, Universitätsbibliothek, 2003.

V. Vassilevska. *Efficient Algorithms for Path Problems in Weighted Graphs*. PhD thesis, Carnegie Mellon University, August 2008.