INFORMS JOURNAL ON COMPUTING
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0899-1499 | EISSN 1526-5528 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000

The Recoverable Robust Two-Level Network Design Problem

Eduardo Álvarez-Miranda
Departamento de Modelación y Gestión Industrial, Universidad de Talca, Curicó, Chile,
ealvarez@utalca.cl

Ivana Ljubić
Department of Statistics and Operations Research, University of Vienna, Vienna, Austria,
ivana.ljubic@univie.ac.at

S. Raghavan
Smith School of Business and Institute for Systems Research, University of Maryland, College Park, MD, USA,
raghavan@umd.edu

Paolo Toth
Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione, Università di Bologna, Bologna, Italy,
paolo.toth@unibo.it

We consider a network design application which is modeled as the two level network design problem under uncertainty. In this problem, one of the two available technologies can be installed on each edge and all customers of the network need to be served by at least the lower level (*secondary*) technology. The decision maker is confronted with uncertainty regarding the set of *primary* customers, i.e., the set of nodes that need to be served by the higher level (*primary*) technology. A set of discrete scenarios associated with the possible realizations of primary customers is available. The network is built in two stages. In the first-stage the network topology must be determined. One may decide to install the primary technology on some of the edges in the first stage, or one can wait to see which scenario will be realized, in which case, edges with the installed secondary technology may be upgraded, if necessary to primary technology, but at higher *recovery* cost. The overall goal then is to build a "recoverable robust" spanning tree in the first stage that serves all customers by at least the lower level technology, and that minimizes the first stage installation cost plus the worst-case cost needed to upgrade the edges of the selected tree, so that the primary customers of each scenario can be served using the primary technology. We discuss the complexity of the problem, provide mixed integer programming models and develop a branch-and-cut algorithm to solve it. Our extensive computational experiments demonstrate the efficacy of our approach.

Key words: network design, robust optimization, mixed integer programming, branch-and-cut
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received November 2012; revised October 2013; accepted March 2014.

1. Introduction

In many real-world settings, when planning an expansion of a telecommunication or power distribution network, a network has to be built even before the set of customers is known with complete

certainty. In addition, if different services are offered to customers, uncertainty could be present regarding the type of service that each of the customers needs. Usually, complete information regarding the underlying demand patterns becomes available much later in the planning process. In that case, applying standard deterministic optimization by considering only one of the possible realizations of the input data leads towards solutions that might not be optimal, or for that matter even feasible, for the final data realization. A wait-and-see approach might also be unacceptable from the economical perspective, since the infrastructure cost might significantly increase as time progresses.

Two-stage stochastic optimization and robust optimization (RO) are two possible approaches to deal with these kind of problems. In two-stage stochastic programming (Birge and Louveaux 2011), the solution is built in two stages. In the first phase, a partial network is built which is later on completed, upon the realization of the uncertain data. The objective is to minimize the cost of the first-stage decisions plus the *expected* cost of the recourse (second-stage) decisions. However, this approach relies on the accuracy of the random representation of the parameter values (such as probability distributions) that allow one to estimate the second-stage expected cost. When such accuracy is not available, the use of *deterministic uncertainty models* arises as a suitable alternative (Kouvelis and Yu 1997, Bertsimas and Sim 2003, Ben-Tal et al. 2010). In these models no assumptions are made about the distribution of the uncertain input parameters. Consequently, in these RO approaches, single-stage decisions are made and solutions are sought that are immune in a certain sense to all possible realizations of the parameter values. Clearly, such solutions may be over conservative, since the networks constructed minimize the investment costs for the worst possible data realization.

Two-Stage Robust Optimization (2SRO) is a modeling approach that combines classical two-stage optimization with robust optimization. In this case, probability distributions are unknown, so the cost of the second-stage decisions is calculated by looking at the worst-case realization of data. The goal is to find a first-stage solution that minimizes the first-stage costs plus the worst-case second-stage costs across all possible data outcomes. For references on different models of 2SRO we refer the reader to Ben-Tal et al. (2004), Atamtürk and Zhang (2007), Thiele et al. (2009), Zhao and Zeng (2012).

Recoverable Robustness is an approach that falls within the framework of 2SRO (Liebchen et al. 2009). Recalling our practical context, assume that the network is built in two stages and we are required to find a first-stage solution that should be *robust* against many possible realizations (*scenarios*) of the input data in a second-stage. *Robustness* in this context means that the first-stage solutions are expected to provide a reasonable performance in terms of optimality and/or feasibility, for any possible realization of the uncertain data. For this model, it is instructive to

think there is a possibility to *recover* the solution constructed in the first stage in a second stage (i.e., to modify the previously defined network in order to make it feasible and/or cheaper) once the uncertainty is resolved. The set of allowed *recovery actions* and their cost may be known in advance for each of the possible data/scenario realizations. These *recovery actions* are *limited*, in the sense that the effort needed to recover a solution may be algorithmically (in terms of how a solution may be modified) and economically (in terms of the cost of recovery actions) limited. Therefore, instead of looking for a solution that is robust against all possible scenarios without allowing any kind of recovery (which is the case for many RO approaches, see Ben-Tal et al. 2010) we want a solution *robust enough* so that it can be "recovered" promptly and at low cost once the uncertainty is resolved. This balance between robustness and recoverability is what defines a *recoverable robust optimization* problem.

The Two-Level Network Design (TLND) problem (Balakrishnan et al. 1994a,b) models the design of telecommunication and power distribution networks, in which two types of customers (requiring two different levels of service) are taken into account. *Primary* customers require a higher level of service and are required to be connected using a higher level (*primary*) technology; *secondary* customers can be connected either by the primary or a *secondary*, and cheaper, technology. The difference between the cost of the primary and secondary technology is often called the *upgrade* cost.

In the deterministic version of the TLND problem the set of primary customers and its complement, the set of secondary customers, are known in advance. However, when long term decisions need to be made (i.e., when the topology of the network needs to determined), there is not always complete knowledge about the set of primary customers. The topology of the network needs to be determined well before a precise knowledge of the demand is available because of the long lead times involved in constructing physical links (edges) in telecommunications and power distribution networks (for example in telecommunications networks fiber cables need to be installed underground which can take a significant period of time). Further, even if some rough idea of demand is known, changes in demographic, socio-economic, or political factors can lead to changes in the structure of the demand during the planning horizon. Under these conditions, a decision maker needs to find a first-stage solution (a spanning tree comprised by secondary and primary technology edges) that can be *recovered* in the second stage, and turned into a feasible one, once the actual set of primary nodes becomes known. For this case the recovery action is the *late upgrade* of a given edge from secondary to primary technology. (In telecommunications networks once a fiber link is in place it can be relatively quick to upgrade the capacity/technology on a link by changing the equipment at the end points of the link.) For each possible scenario, this upgrade incurs an extra cost, *recovery cost*, defined as the sum of all late upgrades that are needed to ensure that all primary nodes

are connected by the primary technology. The Recoverable Robust TLND (RRTLND) problem searches for a solution that minimizes the sum of the first-stage cost and the recovery cost of the second stage defined as the worst case recovery cost over all possible scenarios.

1.1. Our Contribution and Outline of the Paper

The RRTLND problem is a new problem not studied previously in the literature. We first study the problem on trees: we show that the RRTLND problem is NP-Hard even on a star (a star is a tree where all nodes except the central one have degree 1) with uniform upgrade and recovery costs; we then propose a preprocessing procedure and a Mixed Integer Programming (MIP) model with a linear number of variables for solving the RRTLND problem on trees to optimality. In the second part of the paper we propose a MIP formulation for the problem on general networks and develop a branch-and-cut algorithm to solve it. We develop problem-dependent techniques for efficiently separating the underlying inequalities within the branch-and-cut framework. In addition, we use a primal heuristic that relies upon the ideas of *matheuristics* and uses an embedded MIP for solving the problem on trees. Finally, an extensive set of computational experiments are carried out in order to assess (1) the performance of the proposed algorithm and its dependence on the problem parameters, and (2) the nature and characteristics of the solutions obtained. The analysis includes a qualitative study of the solutions in terms of Robustness and Recoverability and an assessment of the algorithmic performance. To complement this analysis, we also consider a Steiner-tree variant of the TLND problem and adapt the algorithm to solve its recoverable robust counterpart.

In §1.2, the TLND problem is formally defined and a review of the main literature presented. In §2 the concept of Recoverable Robustness is discussed further, and the RRTLND problem is formally defined. Results regarding the computational complexity of the RRTLND problem on trees are discussed therein and a new MIP model is shown. A MIP formulation for the RRTLND problem on general graphs together with the elements of our branch-and-cut approach is described in §3. In §4 the Steiner tree variant of the TLND problem, the Two-Level Steiner Tree (TLStT) Problem, is defined and a MIP formulation is presented for its Recoverable Robust counterpart (RRTLStT). In §5 we present and analyze the computational results obtained for two sets of instances for the RRTLND problem and for the RRTLStT problem. Concluding remarks are provided in §6.

1.2. The Two-Level Network Design Problem

In this section we provide a formal definition of the TLND problem and give a review of the previous literature on this problem.

The Two-Level Network Design Problem We are given an undirected connected graph $G = (V, E)$, $|V| = n$, $|E| = m$, with a set $P \subseteq V$, which corresponds to the set of *primary nodes*. On each edge $e \in E$ one of two given technologies (*primary* or *secondary*) can be installed. Correspondingly,

primary and *secondary* edge costs, a_e and b_e are associated with each edge $e \in E$, $a_e \geq b_e \geq 0$, where $u_e = a_e - b_e$; u_e is referred to as the *upgrade* cost as it can be viewed as the cost of *upgrading* a secondary edge to a primary one. Let $\mathbf{X} \in \{0,1\}^{|E|}$ be a binary vector such that $X_e = 1$ if edge $e \in E$ is used in the spanning tree and $X_e = 0$ otherwise; and let $\mathbf{Y} \in \{0,1\}^{|E|}$ be a binary vector such that $Y_e = 1$ if on edge $e \in E$ primary technology is installed and $Y_e = 0$ otherwise. Consequently, if $X_e = 1$ and $Y_e = 0$, secondary technology is installed in e. Let $E(\mathbf{X})$ and $E(\mathbf{Y})$ represent the subsets of edges associated with $\mathbf{X}$ and $\mathbf{Y}$, respectively. The TLND problem consists of finding $(\mathbf{X}^*, \mathbf{Y}^*)$ such that

$$f(\mathbf{X}^*, \mathbf{Y}^*) = \min_{(\mathbf{X},\mathbf{Y}) \in \mathcal{D}} \left(\sum_{e \in E(\mathbf{X})} b_e + \sum_{e \in E(\mathbf{Y})} u_e \right) \tag{1}$$

where $\mathcal{D} = \{(\mathbf{X}, \mathbf{Y}) \in \{0,1\}^{|E|} \times \{0,1\}^{|E|}\}$ such that $E(\mathbf{X})$ is a spanning tree in G, $E(\mathbf{Y})$ is a Steiner Tree connecting P, and $\mathbf{Y} \leq \mathbf{X}$.

Literature Review The history of the TLND problem begins with the introduction of the Hierarchical Network Design problem (HND) (see Current et al. 1986), which is a special case of the TLND problem with $|P| = 2$. Duin and Volgenant (1989) present structural properties and reduction tests for the HND problem; Pirkul et al. (1991) develop a Lagrangian-relaxation based heuristic; Sancho (1995) proposes a dynamic programming procedure; and recently, Obreque et al. (2010) present a branch-and-cut algorithm.

The TLND problem was introduced by Duin and Volgenant (1991) where two heuristics and preprocessing procedures are proposed. Several network flow based models for the TLND problem have been proposed and compared in Balakrishnan et al. (1994b). The authors also propose a composite heuristic that provides an approximation ratio of $\frac{4}{4-\rho}$ if the embedded Steiner tree is solved with an approximation ratio of $\rho < 2$. In Balakrishnan et al. (1994a), the authors provide a dual ascent method derived from a flow-based model presented in Balakrishnan et al. (1994b). More recently, Gouveia and Telhada (2008) discuss alternative MIP formulations for the problem and solve them using Lagrangian relaxation approaches. Some extensions of the TLND problem combine it with the *facility location* problem (see Current 1988, Gollowitzer et al. 2013). In addition to telecommunication applications the TLND problem appears in the design of Internet Protocol networks (Chamberland 2010) and electrical power distribution systems (Costa et al. 2011).

The Multi-Level Network Design Problem (MLND) corresponds to the more general case in which L types of customers and L technologies are available, and the goal is to find a subtree that enables each node at level ℓ to communicate with other node of the same type, by using a tree built of edges of type at most ℓ, for each $1 \leq \ell \leq L$, $L \geq 2$. The problem has been defined by Mirchandani (1996), who called the problem the Multi-Tier Tree Problem and provided a heuristic based on the

one proposed for the TLND problem in Balakrishnan et al. (1994a). In Chopra and Tsai (2002), a branch-and-cut approach derived on a layered graph formulation of the problem has been applied to problems with three to five levels.

2. The Recoverable Robust TLND (RRTLND) Problem

In this section we provide references to the recent applications of the recoverable robust optimization, define the RRTLND problem and study the properties of the problem on trees.

Recent Applications of Recoverable Robust Optimization In Liebchen et al. (2009) the authors introduce the Recoverable Robust Optimization (RRO) concept and provide a general framework for optimization problems affected by uncertainty, while focusing on the applications arising in the railway scheduling. Recently, the concept of RRO has also been applied to other application areas as well. The recoverable robust knapsack problem considering different models of uncertainty is studied in Büsing et al. (2011). Formulations and algorithms for different variants of the recoverable robust shortest path problem are given in Büsing (2012). Finally, in Cicerone et al. (2012) a more general framework of the RRO is studied in which multiple recovery stages are allowed. The authors apply the new model to timetabling and delay management applications.

As other robust optimization approaches the RRO approach allows different models of the uncertainty set, e.g., interval, polyhedral and discrete. In this paper, as in Büsing et al. (2011), we use the discrete set model of uncertainty.

2.1. The Recoverable Robust TLND Problem

Suppose that in a given application of the TLND problem it is not known exactly which elements comprise the set of primary customers P. Instead, we are given a finite set of scenarios K such that, for each $k \in K$, there is a set $P^k \subseteq V$ of nodes corresponding to the primary customers if scenario k is realized. Additionally, motivated by the practical applications, we are given a root node r such that $r \in P^k$ for all $k \in K$. We note that while the application typically has a root node (the root represents, for example, a central office, i.e., a connection to the backbone network), if this is not the case it is easy to modify the formulations and procedures described in this paper to address the situation.[1]

The decision maker needs to determine the topology of the spanning tree connecting the nodes in the network in the first stage. He/she may decide to install the primary technology on edge $e \in E$ in the first stage, or to recover the edge in the second (recovery) stage by *upgrading* it from the secondary to the primary technology (in case scenario k is realized and set P^k requires it).

[1] Simply create a fictitious root node that has an edge to every node in the graph, and either (i) make these new edges have zero cost and add the requirement that the degree of the root node is 1, or (ii) give a sufficiently large cost to these new edges so that only one of them will be in the optimal solution.

Hence, for each edge e and for each scenario k, we also define the *late upgrade* (or *recovery*) cost $r_e^k \geq u_e = a_e - b_e$ that needs to be paid if secondary technology is upgraded on edge e in the second stage when *scenario* k is realized; as opposed to u_e being the *regular* (or *first-stage*) *upgrade* cost.

In our problem, for each scenario $k \in K$, each of the customers $v \in P^k$ is to be served by the primary technology, i.e., there exists a path between r and v along that tree, consisting of solely primary edges. These primary edges can either be installed in the first stage, or recovered in the second stage. Further, in the practical application (for administrative reasons generally) it is required that the primary edges form a connected network (i.e., there can be no isolated primary edges). Our goal is to find a spanning tree (along with a prescription of which edges should be installed as primary in the first stage) that minimizes the overall installation cost in the first stage (given as the sum of the costs of primary and secondary edges), plus the worst recovery costs, calculated over all scenarios $k \in K$.

More formally, let $\mathbf{X} \in \{0,1\}^{|E|}$ be a binary vector as defined in §1.2. Let $\mathbf{Y^0} \in \{0,1\}^{|E|}$ be a binary vector such that $Y_e^0 = 1$ if on edge e the primary technology is installed in the first-stage and $Y_e^0 = 0$ otherwise. Let $\mathbf{Y^k} \in \{0,1\}^{|E|\times|K|}$ be a binary vector such that $Y_e^k = 1$ if the secondary technology that was installed in the first-stage on edge e is upgraded into the primary one in scenario $k \in K$.

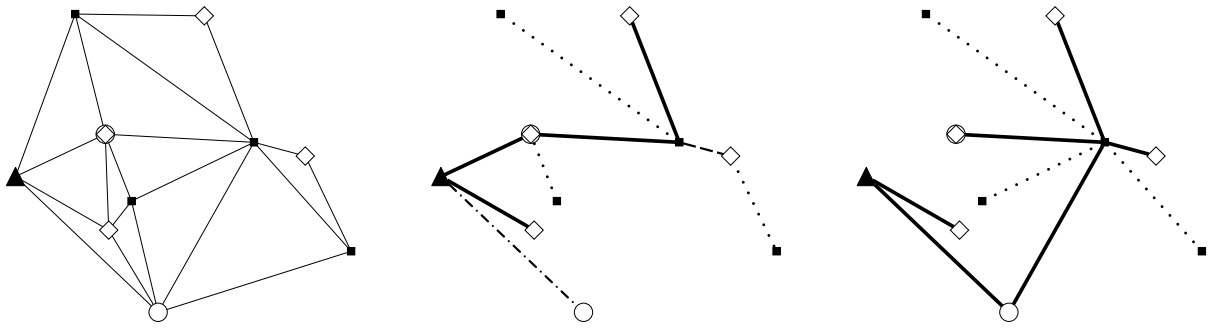
Given a scenario $k \in K$ and a first-stage solution $(\mathbf{X}, \mathbf{Y^0})$ ($\mathbf{X}$ associated to a spanning tree of G and $\mathbf{Y^0} \leq \mathbf{X}$), the *recovery cost* is the minimum total upgrade cost needed to provide feasibility to $(\mathbf{X}, \mathbf{Y^0})$ by recovery actions $\mathbf{Y^k}$. This cost can be expressed as

$$\min_{\mathbf{Y^k} \in \mathcal{Y}(\mathbf{X},\mathbf{Y^0},k)} \left\{ \sum_{e \in E(\mathbf{Y^k})} r_e^k \right\},$$

where $\mathcal{Y}(\mathbf{X}, \mathbf{Y^0}, k)$ is the set of all possible late upgrades for pair $(\mathbf{X},\mathbf{Y^0})$ and the set of primary customers P^k. In other words, vector $\mathbf{Y^k}$ expresses how to *recover* the solution $(\mathbf{X}, \mathbf{Y^0})$ in scenario k in order to make it feasible. For each $k \in K$, the set of all feasible recoveries is given as:

$$\mathcal{Y}\left(\mathbf{X}, \mathbf{Y^0}, k\right) = \{\mathbf{Y^k} \in \{0,1\}^{|E|\times|K|} \,|E(\mathbf{Y^0}) \cup E(\mathbf{Y^k}) \text{ is a Steiner tree spanning } P^k,$$
$$\mathbf{Y^k} \leq \mathbf{X} - \mathbf{Y^0}\}.$$

Note that because of the requirement that the final network (after the uncertainty is resolved) does not allow for isolated primary edges (i.e., $E(\mathbf{Y^0}) \cup E(\mathbf{Y^k})$ is connected), it is easy to see that $E(\mathbf{Y^0})$ must be connected if $r_e^k \geq u_e$. Notice that, given the first stage decision, for each $k \in K$, the optimal recovery solution can be found in $O(n)$ time. The following second-stage objective



(a) Instance of the RRTLND problem. (b) Recovery costs are $r_e = 1.5\ \forall e \in E$. (c) Recovery costs are $r_e = 3\ \forall e \in E$.

Figure 1 Instance and optimal solutions for the RRTLND problem.

Note. Node with symbol ▲ corresponds to r, nodes denoted by ◊ are primary nodes in scenario $k = 1$ and nodes denoted by ◯ are primary nodes in scenario $k = 2$. For each $e \in E$, its primary and secondary costs are $a_e = 2$ and $b_e = 1$, respectively. Dotted, bold, dashed and dot-dashed edges correspond to $E(\mathbf{X})$, $E(\mathbf{Y^0})$, $E(\mathbf{Y^1})$ and $E(\mathbf{Y^2})$, respectively.

function, $R(\mathbf{X}, \mathbf{Y^0})$, expresses the *robust recovery cost* (which is the maximum recovery cost over all scenarios $k \in K$):

$$R(\mathbf{X}, \mathbf{Y^0}) = \max_{k \in K} \min_{\mathbf{Y^k} \in \mathcal{Y}(\mathbf{X}, \mathbf{Y^0}, k)} \left\{ \sum_{e \in E(\mathbf{Y^k})} r_e^k \right\}.$$

The Recoverable Robust TLND (RRTLND) problem is defined as follows

$$OPT_{RR} = \min\{ \sum_{e \in E(\mathbf{X})} b_e + \sum_{e \in E(\mathbf{Y^0})} u_e + R(\mathbf{X}, \mathbf{Y^0}) \mid (\mathbf{X}, \mathbf{Y^0}) \in \{0,1\}^{|E|} \times \{0,1\}^{|E|}, \quad (2)$$
$$E(\mathbf{X}) \text{ is a spanning tree on } G,$$
$$\mathbf{Y^0} \leq \mathbf{X} \text{ and } E(\mathbf{Y^0}) \text{ is connected } \}.$$

In Figure 1(a) an instance of the RRTLND problem with two scenarios is shown. In Figures 1(b) and 1(c) optimal solutions for different cost structures are presented. In the first case, recovery (i.e., late upgrade) costs are 50% more expensive than regular upgrade costs while in the second case the difference goes to 200%. This difference in the cost structure explains why in the solution shown in 1(b) there are edges that are recovered in a second stage for each of the scenarios, while in the solution shown in 1(c) no recovery is performed since it is cheaper to install primary edges in the first stage than recover edges in a second stage. The cost of the first solution is given by $OPT_{RR} = 1 \times 9 + 1 \times 4 + \max\{1 \times 1.5, 1 \times 1.5\} = 14.5$ and the cost of the second solution is given by $OPT_{RR} = 1 \times 9 + 1 \times 6 + \max\{\emptyset\} = 15$.

A first-stage solution $(\mathbf{X}, \mathbf{Y^0})$ is *robust* because, regardless of which scenario is realized, it ensures that the second-stage actions will be efficient (due to the minimization of the worst case) and easy

to implement (because only upgrades are needed). In this sense, the more scenarios we take into account to find $(\mathbf{X}, \mathbf{Y^0})$, the more *robust* the solution is; because we are foreseeing more possible states of the future uncertainty. Along the same line, recoverability is the capability of a first-stage solution to become feasible by means of second-stage actions.

We wish to emphasize that in this two-stage setting the classical single-stage RO approaches such as those proposed in Kouvelis and Yu (1997) or Ben-Tal and Nemirovski (2000) are not good models, and can be overconservative. In intuitive terms, because the typical RO approaches are single-stage approaches without the possibility of recovery in the second stage, they require the constructed solutions to be feasible for all scenarios!

In RRO the first-stage solution lies between two extremes: an *absolute robust* (AR) solution and a *pure wait-and-see* (W&S) solution. The first case corresponds to a solution for which no recovery is allowed, i.e., $E(\mathbf{Y^0})$ spans $\check{P} = \bigcup_{k \in K} P^k$, so the solution is feasible for all scenarios (this solution in the first case can be viewed as one that would be obtained under the classical single-stage RO approach). On the contrary, the second case corresponds to a solution for which $E(\mathbf{Y^0}) = \emptyset$, so a complete primary Steiner tree should be constructed (but only the most expensive one is considered in the total cost) in the second-stage for each P^k, $k \in K$ (the solution in the second case can be viewed as one with maximum recovery as it requires each primary edge to be obtained via recovery). Both solutions can lead to very high total costs, either because unnecessarily many primary edges have to be installed in the first-stage or because the second-stage primary costs are considerably higher than those of the first stage. The *Gain of Recovery* (GoR) is defined as the relative difference between OPT_{RR} and the cost of these two solutions, i.e., $GoR(AR) = \frac{OPT_{AR} - OPT_{RR}}{OPT_{AR}} \times 100\%$ and $GoR(W\&S) = \frac{OPT_{W\&S} - OPT_{RR}}{OPT_{W\&S}} \times 100\%$, where OPT_{AR} and $OPT_{W\&S}$ are the costs of the optimal AR and W&S solutions respectively.

2.2. The RRTLND Problem on Trees

In this section we consider the RRTLND problem on trees.

2.2.1. Complexity of the RRTLND Problem on Trees

THEOREM 1. *Solving the RRTLND problem is NP-hard even if the input graph G is a tree, and all regular and late upgrade costs are uniform.*

Proof. Because the input graph is a tree, every edge in the graph will have at least secondary technology installed. Therefore the optimization only needs to consider regular and late upgrade costs.

We will show the result by a transformation (the main idea in this transformation is similar to Garg et al. 1997) from the *minimum vertex cover problem*. Given a graph $H = (V_H, E_H)$, $V_H =$

$\{v_1, \ldots, v_n\}$, a set of vertices such that each edge of the graph is incident to at least one vertex of the set is called a *vertex cover*. In the minimum vertex cover problem we wish to find a vertex cover of smallest cardinality. Given an instance of a vertex cover on the graph H, construct an instance of the RRTLND problem with K scenarios as follows. First, construct a star graph $S = (V_S, E_S)$ from H as follows. Let $V_S = v_0 \cup V_H$, and $E_S = \bigcup_{i=1\ldots,n}\{v_0, v_i\}$. Let the upgrade costs in the first stage be $u_e = 1$, for all $e \in E_S$, and let $M = n/2$ be the uniform second-stage upgrade cost, i.e., $r_e^k = M$, for all $e \in E_S$, $k \in K$. For each edge $\{u_k, v_k\}$ in E_H, create a scenario $k \in K$ in S, by setting $P^k = \{v_0, u_k, v_k\}$.

We now show that the optimal solution of the RRTLND problem on S provides us the minimum vertex cover on H. Without loss of generality, assume that the value of the vertex cover, C, on H is such that $C \leq \frac{n-1}{2}$.[2] Consider the possible values for the maximum recovery cost R^*: (i) If there exists $k \in K$, such that the edges $\{u_k, v_0\}$ and $\{v_0, v_k\}$ were not purchased in the first stage, then the maximum recovery cost will be $R^* = 2M$. (ii) If for all $k \in K$ at least one of the two edges is purchased in the first stage, but there also exists $\hat{k}$ such that exactly one of the two edges is purchased, then $R^* = M$. Since for each scenario $k \in K$, at least one of the edges $\{u_k, v_0\}, \{v_0, v_k\}$ need to be installed in the first stage, the minimum cost first-stage solution that satisfies this property corresponds to the minimum vertex cover on H (edge $\{v_0, v_k\}$ installed in the first stage (on S) corresponds to node v_k in the vertex cover on H). Therefore, the total cost of such a constructed solution is upper bounded by $C + M$. (iii) Finally, if for all $k \in K$, both edges are purchased in the first stage, $R^* = 0$, but the first-stage cost is equal to n. It is not difficult to see that the second solution will be the optimal one (recall that we chose H such that $C < n/2$) since: $C + M < 2M$ and $C + M < n$. □

2.2.2. A MIP Model for the RRTLND Problem on Trees We now provide a MIP formulation for the RRTLND problem on trees for which it is necessary to perform an $O(nK)$ preprocessing. For constant K this formulation is of compact size. Furthermore, it involves only binary variables associated with the installation of the primary technology in the first stage. Due to the preprocessing, this model does not involve the variables associated with the second-stage decisions.

Preprocessing: Given G that has a tree structure, for each scenario $k \in K$ we first solve the Steiner tree problem with the set P^k being the terminal nodes of that tree. We assume that on all edges in G secondary technology is installed in the first stage, so that all edges of the Steiner Tree

[2] Given a graph H we can create another graph H' by duplicating the set of nodes and add an additional node (i.e., $V_{H'} = V_H \cup v_0 \cup \{v_{n+1}, \ldots, v_{2n}\}$). The set of edges $E_{H'}$ in this new graph includes the previous edges and an edge from v_0 to each node v_i, $i = 1, \ldots, 2n$. It is easy to see that the minimum vertex cover on $V_{H'}$ is the union of v_0 and the minimum vertex cover on V_H, and satisfies our assumption on the size of the vertex cover.

need to be recovered in the second stage. Therefore, to find the optimal Steiner tree, we consider the edge cost defined by r_e^k, for each $e \in E$, for each $k \in K$. Let $\mathcal{P}_k$ be the set of edges corresponding to the optimal Steiner tree, for $k \in K$, and let $\omega_k = \sum_{e \in \mathcal{P}_k} r_e^k$ be the recovery cost for that tree, assuming that there were no primary edges in the first stage. Obviously, finding the optimal Steiner trees can be done in $O(n)$ time, for each $k \in K$. We now state a useful property concerning the structure of RRTLND problem solutions on trees.

PROPERTY 1. Let $\mathcal{P} = \bigcap_{k \in K} \mathcal{P}_k \neq \emptyset$ denote the set of edges for which the recovery is needed over all scenarios $k \in K$. Given that for all $e \in E$, $k \in K$ we have $r_e^k \geq u_e$, there always exists an optimal solution to the RRTLND problem on trees such that the primary edges are installed in the first stage along all edges in $\mathcal{P}$. Further, the subgraph induced by $\mathcal{P}$ is connected.

Hence, the optimal primary subtree of the first stage is a rooted subtree of G which is a superset of $\mathcal{P}$ and a subset of E. Therefore, if $\mathcal{P} \neq \emptyset$, we can shrink all the edges of $\mathcal{P}$ into the root node and continue solving the problem on the shrunken tree. Consequently, we can assume w.l.o.g. that $\mathcal{P} = \emptyset$. Given that G is a tree with a pre-specified root node, for each edge $e : \{u, v\} \in E$ $(u, v \neq r)$, we can uniquely determine the *predecessor* edge e' on the path between r and e. Let $\mathbf{s} \in \{0,1\}^{|E|}$ be a binary vector such that $s_e = 1$ if a primary technology is installed in the first stage and $s_e = 0$ otherwise. The following formulation allows us to solve the RRTLND problem on trees:

$$f(\mathbf{s}^*) = \min \sum_{e \in E} u_e s_e + \lambda \tag{T.1}$$

$$\text{s.t.} \quad s_{e'} \geq s_e \qquad \forall e \in E,\ e' \text{ is predecessor of } e : \{u, v\}, u, v \neq r \tag{T.2}$$

$$\lambda \geq \omega_k - \sum_{e \in \mathcal{P}_k} r_e^k s_e \qquad \forall k \in K \tag{T.3}$$

$$\mathbf{s} \in \{0,1\}^{|E|},\ \lambda \geq 0 \tag{T.4}$$

In the formulation (T.1)-(T.4) we only have one set of binary variables, $\mathbf{s}$, and $O(n + K)$ constraints. Therefore, for a constant number of scenarios, this is a compact formulation. Constraints (T.2) force first-stage primary edges to form a connected component rooted at r. Inequalities (T.3) model the nested maximization problem associated with the robust recovery cost; if primary technology is installed on edge e in the first stage, then its recovery cost is subtracted from ω_k for those sets for which e is supposed to be upgraded in the second stage (i.e., for $e \in \mathcal{P}_k$). This MIP model will be used in a matheuristic fashion for finding feasible solutions of the RRTLND problem in general graphs. This will be the core of the primal-heuristic embedded into a branch-and-cut approach framework that we discuss in §3.2.

3. MIP Model and Branch-and-Cut Algorithm

Before we provide a MIP model for the RRTLND problem, we observe that for every feasible solution of the problem, we can associate a rooted spanning arborescence consisting of a rooted primary sub-arborescence embedded into the secondary one. In addition, for each $k \in K$, edges from $E(\mathbf{Y^k})$ can uniquely be oriented, so that the set of directed primary edges from the first-stage solution, plus the set of directed edges from $E(\mathbf{Y^k})$ builds a Steiner arborescence spanning P^k. Henceforth, instead of dealing with MIP models containing binary variables associated with edges of the graph G, we will consider its bidirected counterpart, $G_A = (V, A)$, where $A = \{(r,i) \mid e : \{r,i\} \in E\} \cup \{(i,j),(j,i) \mid e : \{i,j\} \in E,\ i,j \neq r\}$.

3.1. MIP formulation for the RRTLND Problem

The MIP formulation investigated in this paper is based on directed cut-set inequalities. The LP-relaxation of this model usually accomplishes good quality lower bounds, since many facet-defining inequalities can be projected out of the directed model for tree problems (see Grötschel et al. 1992).

Let $\mathbf{x} \in \{0,1\}^{|A|}$ be a binary vector such that $x_{ij} = 1$ if arc $(i,j) \in A$ belongs to the spanning arborescence and $x_{ij} = 0$ otherwise, let $\mathbf{y^0} \in \{0,1\}^{|A|}$ be a binary vector such that $y^0_{ij} = 1$ if primary technology is installed in arc $(i,j) \in A$ in the first stage and $y^0_{ij} = 0$ otherwise. Let $\mathbf{y^k} \in \{0,1\}^{|A|\times|K|}$ be a binary vector such that $y^k_{ij} = 1$ if the secondary technology installed on arc $(i,j) \in A$ is upgraded into the primary one in scenario $k \in K$ and $y^k_{ij} = 0$ otherwise. We use the following notation: A set of vertices $S \subseteq V$ $(S \neq \emptyset)$ and its complement $\bar{S} = V \backslash S$, induce two directed cuts: $\delta^+(S) = \left\{(i,j) \mid i \in S, j \in \bar{S}\right\}$ and $\delta^-(S) = \left\{(i,j) \mid i \in \bar{S}, j \in S\right\}$; we write $\mathbf{x}(A') = \sum_{(i,j)\in A'} x_{ij}$ for any subset $A' \subset A$.

Vector $\mathbf{x}$ is associated with a directed spanning tree of G_A (spanning arborescence) rooted at r if it satisfies the following set of inequalities

$$\mathbf{x}\left(\delta^-(S)\right) \geq 1 \quad \forall S \subseteq V \backslash \{r\},\ S \neq \emptyset, \tag{3}$$

a vector $\mathbf{y^0}$ is associated with a directed arborescence of G_A rooted at r if it satisfies

$$\mathbf{y^0}\left(\delta^-(S)\right) \geq \mathbf{y^0}\left(\delta^-(i)\right) \quad \forall i \in S,\ \forall S \subseteq V \backslash \{r\},\ S \neq \emptyset, \tag{4}$$

and a vector of recovery actions $\mathbf{y^k}$ along with a vector $\mathbf{y^0}$ are associated with a directed Steiner arboresence of P^k for all scenarios $k \in K$ if they fulfil

$$\left(\mathbf{y^0} + \mathbf{y^k}\right)\left(\delta^-(S)\right) \geq 1 \quad \forall S \subseteq V \backslash \{r\},\ S \cap P^k \neq \emptyset,\ \forall k \in K. \tag{5}$$

Constraints (3), (4) and (5) are called $\mathbf{x}$-cuts, $\mathbf{y^0}$-cuts and *scenario*-cuts, respectively. As we will describe in detail later, our branch-and-cut performs at a given node of the branch-and-bound

tree a separation procedure of $\mathbf{x}$-cuts, $\mathbf{y^0}$-cuts and *scenario*-cuts by means of (i) the resolution of a max-flow problem on a *support graph* induced by the Linear Programming (LP) relaxation and (ii) a combinatorial enumeration of those cuts on a *support tree* also induced by the current LP relaxation.

The MIP model for the RRTLND problem then reads as follows:

$$\min \sum_{e \in E} b_e X_e + \sum_{e \in E} u_e Y_e^0 + \omega$$

$$\text{s.t.} \quad \omega \geq \sum_{e \in E} r_e^k Y_e^k \qquad \forall k \in K \tag{6}$$

$$(3),\ (4),\ (5)$$

$$X_e = x_{ij} + x_{ji} \quad Y_e^0 = y_{ij}^0 + y_{ji}^0 \quad Y_e^k = y_{ij}^k + y_{ji}^k \qquad \forall e : \{i,j\} \in E, \forall k \in K \tag{7}$$

$$X_e, Y_e^0, Y_e^k \in \{0,1\} \qquad \forall e \in E, k \in K \tag{8}$$

Before concluding this section we note that it is possible to also write a compact formulation using three sets of flow variables—to model the three sets of connectivities imposed by constraints (3), (4) and (5). However, given the number of scenarios, this model blows up rapidly and is not a computationally viable approach for the problem.

3.2. Branch-and-Cut Algorithm

The MIP formulation based on cut-set inequalities for the RRTLND problem cannot be solved directly, even for small instances, since there are an exponential number of $\mathbf{x}$-, $\mathbf{y^0}$- and *scenario*-cuts. In this section we describe a branch-and-cut approach used for solving the problem. We first explain different schemes designed to separate the directed cut-set constraints (i.e., (3), (4) and (5)). Next, the initialization performed to improve the quality of the lower bounds of the initial MIP model is described. Finally, we provide a description of the primal heuristic embedded within the branch-and-cut framework that helps in establishing high-quality upper bounds early in the search process.

3.3. Separation of Cut-set Inequalities

Cut-set inequalities are usually separated using maximum-flow algorithms. Basic ideas of this separation for the RRTLND problem are provided below. In addition, we also explain two advanced separation mechanisms that are called *mixed* and *combinatorial cuts separation*. The latter approach uses the problem-specific structure to speed-up the separation process and improves lower bounds in the earlier phase of the search process.

Basic Separation Procedures (Max-Flow Based Cuts) Violated cut-set inequalities can be found in polynomial time using a maximum-flow algorithm on the *support graph* with arc-capacities given by the current fractional solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}^{\mathbf{0}}, \tilde{\mathbf{y}}^{\mathbf{k}})$. When separating $\mathbf{x}$-, $\mathbf{y^0}$- and *scenario*-cuts, the

capacities of the support graph are set to be equal to the values of $\tilde{\mathbf{x}}$, $\tilde{\mathbf{y}}^{\mathbf{0}}$ and $(\tilde{\mathbf{y}}^{\mathbf{0}} + \tilde{\mathbf{y}}^{\mathbf{k}})$, respectively. For finding the maximum flow in a directed graph, we used an adaptation of Cherkassky and Goldberg (1995) maximum flow algorithm.

The separation is performed in the following order: First, we randomly select a node from $V \setminus \{r\}$ and if there is a violated $\mathbf{x}$-cut separating v from r, we insert it into the LP (together with the corresponding set of nested and backward cuts, see the explanation below). We resolve the LP and continue as long as such violated cuts are found. After that, we attempt to find violated $\mathbf{y}^{\mathbf{0}}$-cuts. This time, we perform the maximum-flow calculation between r and each $i \in V \setminus \{r\}$, such that $\mathbf{y}^{\mathbf{0}}(\delta^-(i)) > 0$. In the final phase, when no more violated $\mathbf{x}$-cuts and $\mathbf{y}^{\mathbf{0}}$-cuts can be found, we search for violated *scenario*-cuts. For each scenario $k \in K$, we perform the maximum-flow calculation between r and each $i \in P^k$.

By following this order in the separation procedure, we avoid inserting cuts that have a greater likelihood of being weak (i.e., dominated by others) and thus reduce the computational effort of the separation. For example, for a given set S and $i \in S$ such that $y^0(\delta^-(i)) = 1$, the corresponding y^0-cut dominates all scenario-cuts associated with the same S.

Mixed Separation Because $\mathbf{y}^{\mathbf{0}} \leq \mathbf{x}$, if a set $S \subseteq V \setminus \{r\}$ induces a violated $\mathbf{x}$-cut then it might also induce a violated $\mathbf{y}^{\mathbf{0}}$-cut, if there exist $i \in S$ such that $\mathbf{y}^{\mathbf{0}}(\delta^-(S)) < \mathbf{y}^{\mathbf{0}}(\delta^-(i))$. Because $\mathbf{y}^{\mathbf{k}} \leq \mathbf{x} - \mathbf{y}^{\mathbf{0}}$, if there exists a scenario $k \in K$, such that $S \cap P^k \neq \emptyset$, the same set S also induces a violated *scenario*-cut. Hence, within the separation process applied to $\mathbf{x}$-cuts we can also separate $\mathbf{y}^{\mathbf{0}}$-cuts and *scenario*-cuts without solving another max-flow problem. We use these facts to develop a separation procedure that we refer to as *mixed separation*. The outline of this procedure is given in Algorithm 1. In this procedure, we call the maximum-flow algorithm `MaxFlow` $(G_A, \tilde{\mathbf{x}}', r, v, S_r, S_v)$ that, for a given directed graph G_A, calculates the maximum flow between r and v with capacities $\tilde{\mathbf{x}}'$. The algorithm returns two subsets of nodes: S_r, $r \in S_r$ and S_v, $v \in S_v$, such that the edges of the cut $\delta^+(S_r)$ and $\delta^-(S_v)$ induce the maximum flow. Inequalities associated with the set S_r and S_v are called *forward* and *backward cuts*, respectively. Then, we continue recalculating maximum flows on the same graph G_A, on which the capacities of the edges from the two previously found cut-sets $\delta^+(S_r)$ and $\delta^-(S_v)$ are set to one. That way, we detect disjoint cuts and we reuse the previous maximum flow computation to speed up the overall separation. The latter strategy is known as the *nested cuts* approach (see Ljubić et al. 2006). Variable *MAX-CUTS* denotes the number of cuts to be inserted before the LP is resolved. In our implementation *MAX-CUTS* was set to 25.

Finally, we apply two variants of the mixed cut separation. The first one is described in Algorithm 1: whenever we detect a violated $\mathbf{x}$-cut, we also add corresponding violated $\mathbf{y}^{\mathbf{0}}$-cuts and *scenario*-cuts. On the other hand, when performing the separation of $\mathbf{y}^{\mathbf{0}}$-cuts in a later phase, we basically use the same idea to add violated *scenario*-cuts, whenever a violated $\mathbf{y}^{\mathbf{0}}$-cut is detected.

Algorithm 1 Mixed Separation

Input: Graph $G_A = (V, A)$, fractional solution $\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}^{\mathbf{0}}, \tilde{\mathbf{y}}^{\mathbf{k}}\right)$

Choose a random node v from $V \setminus \{r\}$
$\tilde{\mathbf{x}}' = \tilde{\mathbf{x}}$
repeat
  $f = \texttt{MaxFlow}(G_A, \tilde{\mathbf{x}}', r, v, S_r, S_v)$
  Detect the cut $\delta^+(S_r)$ such that $\tilde{\mathbf{x}}'\left(\delta^+(S_r)\right) = f$, $r \in S_r$
  if $f < 1$ **then**
    Insert violated cut $\mathbf{x}\left(\delta^+(S_r)\right) \geq 1$ into the LP
    $\tilde{i_r} = \arg\max_{i \notin S_r} \tilde{\mathbf{y}}^{\mathbf{0}}\left(\delta^-(i)\right)$
    if $\tilde{\mathbf{y}}^{\mathbf{0}}\left(\delta^+(S_r)\right) < \tilde{\mathbf{y}}^{\mathbf{0}}\left(\delta^-\left(\tilde{i_r}\right)\right)$ **then**
      Insert violated cut $\mathbf{y}^{\mathbf{0}}\left(\delta^+(S_r)\right) \geq \mathbf{y}^{\mathbf{0}}\left(\delta^-\left(\tilde{i_r}\right)\right)$ into the LP
    for all $k \in K$, $\bar{S_r} \cap P^k \neq \emptyset$ **do**
      Insert the violated cut $\left(\mathbf{y}^{\mathbf{0}} + \mathbf{y}^{\mathbf{k}}\right)\left(\delta^+(S_r)\right) \geq 1$ into the LP
    $\tilde{x}'_{ij} = 1$, $\forall (i,j) \in \delta^+(S_r)$
    Detect the cut $\delta^-(S_v)$ such that $\tilde{\mathbf{x}}'\left(\delta^-(S_v)\right) = f$, $v \in S_v$
    if $S_v \neq \bar{S_r}$ **then**
      Insert the violated cut $\mathbf{x}\left(\delta^-(S_v)\right) \geq 1$ into the LP
      $\tilde{i_v} = \arg\max_{i \in S_v} \tilde{\mathbf{y}}^{\mathbf{0}}\left(\delta^-(i)\right)$
      if $\tilde{\mathbf{y}}^{\mathbf{0}}\left(\delta^-(S_v)\right) < \tilde{\mathbf{y}}^{\mathbf{0}}\left(\delta^-\left(\tilde{i_v}\right)\right)$ **then**
        Insert the violated cut $\mathbf{y}^{\mathbf{0}}\left(\delta^-(S_v)\right) \geq \mathbf{y}^{\mathbf{0}}\left(\delta^-\left(\tilde{i_v}\right)\right)$ into the LP
      for all $k \in K$, $S_v \cap P^k \neq \emptyset$ **do**
        Insert the violated cut $\left(\mathbf{y}^{\mathbf{0}} + \mathbf{y}^{\mathbf{k}}\right)\left(\delta^-(S_v)\right) \geq 1$ into the LP
      $\tilde{x}'_{ij} = 1$, $\forall (i,j) \in \delta^-(S_v)$
until $f \geq 1$ or *MAX-CUTS* constraints added

Resolve the LP

Combinatorial Cuts The separation of *combinatorial cuts* relies on the following idea: if we knew the structure of the optimal spanning tree built in the first stage, to find the optimal recoverable robust solution it is sufficient to consider the cut-sets associated with the edges of that tree. Let $\tilde{T} = (\tilde{V}, \tilde{A})$ denote the rooted spanning arborescence associated with $\mathbf{x}$-variables of the optimal solution. Observe that the removal of an arc $(j, \ell) \in \tilde{A}$ separates $\tilde{T}$ into two components. Let V_ℓ be the set of nodes of the sub-arborescence rooted at ℓ, and K_ℓ be the set of *relevant scenarios*, i.e., $K_\ell = \{k \in K \mid V_\ell \cap P^k \neq \emptyset\}$. The values of the variables $\mathbf{y}^{\mathbf{0}}$ and $\mathbf{y}^{\mathbf{k}}$ could then be determined by solving the following Integer Program (IP):

$$\min \sum_{(i,j) \in \tilde{A}} u_{ij} y^0_{ij} + \max_{k \in K} \min \sum_{(i,j) \in \tilde{A}} r^k_{ij} y^k_{ij} \tag{9}$$

$$\text{s.t.} \quad (\mathbf{y}^{\mathbf{0}} + \mathbf{y}^{\mathbf{k}})(\delta^-(V_\ell)) \geq 1 \qquad \forall (j,\ell) \in \tilde{A},\ \forall k \in K_\ell \tag{10}$$

$$\mathbf{y}^{\mathbf{0}}(\delta^-(V_\ell)) \geq \mathbf{y}^{\mathbf{0}}(\delta^-(i)) \qquad \forall i \in V_\ell,\ \forall (j,\ell) \in \tilde{A} \tag{11}$$

$$\mathbf{y}^{\mathbf{0}} \in \{0,1\}^{|\tilde{A}|}, \mathbf{y}^{\mathbf{k}} \in \{0,1\}^{|\tilde{A}|} \qquad \forall k \in K \tag{12}$$

Obviously, in this model there are only $O(nK)$ constraints, and the associated sets V_ℓ can be determined in $O(n)$ time using a dynamic programming procedure. Furthermore, formulation (9)-(12) is equivalent to formulation (T.1)-(T.4).

Since we do not know the structure of the optimal arborescence in the first stage, we try to heuristically approximate it and generate cut-sets of type (10) and (11) on graph G (with the

Algorithm 2 Combinatorial Cuts

Input: Graph $G_A = (V, A)$, $\tilde{T} = \left(\tilde{V}, \tilde{A}\right)$ a spanning arborescence of G_A, fractional solution $\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}^{\mathbf{0}}, \tilde{\mathbf{y}}^{\mathbf{k}}\right)$

$\mathcal{L} = \left\{v \in \tilde{V} \mid \delta^+(v) = \emptyset\right\}$
for all $\ell \in \mathcal{L}$ **do**
  $V_\ell = \{\ell\}$
  $K_\ell = \left\{k \in K \mid \ell \in P^k\right\}$
repeat
  Chose $\ell \in \mathcal{L}$
  Let j be the parent of ℓ in $\tilde{T}$, i.e., $(j, \ell) \in \tilde{A}$
  if $\tilde{y}^0_{j\ell} < 1$ **then**
    for all $k \in K_\ell$ **do**
      if $\left(\tilde{\mathbf{y}}^{\mathbf{0}} + \tilde{\mathbf{y}}^{\mathbf{k}}\right)(\delta^-(V_\ell))\big) < 1$ **then**
        Insert the violated cut $\left(\mathbf{y}^{\mathbf{0}} + \mathbf{y}^{\mathbf{k}}\right)(\delta^-(V_\ell))\big) \geq 1$ into the LP
  $K_j = K_\ell \cup \{k \in K \mid j \in P^k\}$
  $V_j = V_\ell \cup \{j\}$; $\mathcal{L} = \mathcal{L} \backslash \{\ell\}$; $\mathcal{L} = \mathcal{L} \cup \{j\}$
until $\mathcal{L} = \{r\}$

heuristic tree) and insert them into the model. Thus, we are able to insert $O(nK)$ cuts into the LP, in $O(mK)$ running time. For good approximations of $\tilde{T}$, these combinatorial cuts can bring a significant speed-up to the separation procedure, especially in the early stages of the cutting plane algorithm. In Algorithm 2 the outline of the procedure is presented. The main idea of the algorithm is to recursively generate sets V_ℓ and K_ℓ and insert the violated cuts into the current LP. We start with the leaf nodes of $\tilde{T}$ and process the arborescence in a bottom-up fashion until we reach the root node. Whenever we process an arc of $\tilde{T}$, we insert violated cuts into the current LP. In total, each edge from G is "visited" at most twice and therefore, the total running time of this procedure is at most $O(mK)$.

Combinatorial cuts are separated together with $\mathbf{y^0}$-cuts and before the (more time consuming) separation of *scenario*-cuts is performed. To approximate the tree $\tilde{T}$, we run the minimum spanning tree algorithm on G with edge weights set to

$$w_e = b_e \min\{(1 - \tilde{x}_{ij}), (1 - \tilde{x}_{ji})\} \text{ for each } e : \{i, j\} \in A, \tag{13}$$

where $\tilde{\mathbf{x}}$ is the value of the current fractional solution. Combinatorial cuts are also added, whenever in the current LP, $\mathbf{x}$ is a binary vector.

3.4. MIP Initialization

In our branch-and-cut approach we first drop all $\mathbf{x}$-, $\mathbf{y^0}$- and *scenario*-cuts, and add them in an iterative fashion only when violated. However, to improve the quality of the lower bounds we incorporate additional constraints to the initial model. Since for the RRTLND problem the $\mathbf{x}$ variables should construct a spanning arborescence of G, the following in-degree constraints

$$\mathbf{x}\left(\delta^-(i)\right) = 1, \, \forall i \in V, \tag{14}$$

Algorithm 3 Primal Heuristic

Input: Graph $G_A = (V, A)$, fractional solution $\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}^{\mathbf{0}}, \tilde{\mathbf{y}}^{\mathbf{k}}\right)$, cost vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{u} = \mathbf{a} - \mathbf{b}$ and $\mathbf{r}$.
Output: A feasible solution $\left(\hat{\mathbf{x}}, \hat{\mathbf{y}}^{\mathbf{0}}, \hat{\mathbf{y}}^{\mathbf{k}}\right)$ for the RRTLND problem
1: $\tilde{T} = (V_{\tilde{T}}, E(\tilde{\mathbf{x}})) =$`spanningTree`$(G, \mathbf{w})$, where $\mathbf{w}$ is defined by (13).
2: **for all** $k \in K$ **do**
3: $\quad \mathcal{P}_k =$ `steinerTree`$\left(P^k, \tilde{T}\right)$
4: $\quad \omega_k = \sum_{(i,j)\in\mathcal{P}_k} r^k_{ij}$
5: Solve problem (T.1)-(T.4) with $\tilde{T}$ as input graph, cost vectors $\mathbf{u}$ and $\mathbf{r}$, and vectors $\mathcal{P}_k$ and ω_k.
6: Let $\mathbf{s}^*$ be an optimal solution for (T.1)-(T.4) and $A(\tilde{\mathbf{x}})$ be the arcs of $E(\tilde{\mathbf{x}})$ oriented away from r. A feasible solution $\left(\hat{\mathbf{x}}, \hat{\mathbf{y}}^{\mathbf{0}}, \hat{\mathbf{y}}^{\mathbf{k}}\right)$ for the RRTLND problem is defined by $\hat{x}_{ij} = 1$ if $(i,j) \in A(\tilde{\mathbf{x}})$ and $\hat{x}_{ij} = 0$ otherwise, $\hat{y}^0_{ij} = 1$ if $s^*_{ij} = 1$ and $\hat{y}^0_{ij} = 0$ otherwise, $\hat{y}^k_{ij} = 1$ if $(i,j) \in \mathcal{P}_k$ and $s^*_{ij} = 0$ and $\hat{y}^k_{ij} = 0$ otherwise.

are valid inequalities that stress the tree-like topology of the corresponding solution. We also include the constraints

$$\left(y^0_{ij} + y^k_{ij}\right) + \left(y^0_{ji} + y^k_{ji}\right) \leq 1, \; \forall (i,j) \in A, \; \forall k \in K, \tag{15}$$

that correspond to subtour elimination constraints of size 2 for arcs with primary technology.

Finally, we also use combinatorial cuts described above as part of the initialization of the MIP model. The arborescence $\tilde{T}$ is approximated by the minimum spanning tree considering edge weights b_e, $\forall e \in E$. This initialization provides good initial lower bounds since many important cut-sets are inserted into the model at an early stage of the cutting plane procedure without the resolution of a maximum flow problem.

3.5. Primal Heuristic

An important component of our branch-and-cut is the embedded Primal Heuristic, whose pseudo-code is given in Algorithm 3. The core of the heuristic is to solve an instance of the RRTLND problem on an induced spanning arborescence $\tilde{T}$ of G_A to optimality. For constructing the spanning arborescence $\tilde{T}$ we use LP-values of $\mathbf{x}$ variables from the current LP relaxation. We run the minimum spanning tree algorithm on G with edge weights defined by (13) (Step 1 of the algorithm).

In the loop (2-4) the preprocessing described in §2.2.2 is applied. We find the optimal Steiner Tree (constructed by recovered edges) on $\tilde{T}$ considering terminal set P^k; ω_k denotes the corresponding total recovery cost for each scenario. The main step of the algorithm is Step 5, where the MIP problem (T.1)-(T.4) is solved. The feasible primal solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}}^{\mathbf{0}}, \hat{\mathbf{y}}^{\mathbf{k}})$ of our problem is obtained by mapping the solution $\mathbf{s}^*$ and the structure of $\tilde{T}$ as shown in Step 6. All arcs in $\tilde{T}$ define the spanning arborescence associated with $\hat{\mathbf{x}}$. The values of $\hat{\mathbf{y}}^{\mathbf{0}}$ correspond to the values of $\mathbf{s}^*$ and the values of $\hat{\mathbf{y}}^{\mathbf{k}}$ are calculated by a simple inspection using the information contained in $\mathcal{P}_k$, $\forall k \in K$, and $\mathbf{s}^*$.

Although we know that the problem is NP-Hard, in practice the computational effort to solve the problem is remarkably little (usually only a fraction of a second or a couple of seconds). This makes the primal heuristic very effective since feasible solutions are quickly computed.

4. The Recoverable Robust Two-Level Steiner Tree Problem

In some real-world instances of the TLND problem, in addition to the customer nodes, there are additional nodes in the network (corresponding to street intersections, for example) that do not require any service. The definition of the TLND problem can be extended correspondingly. In this variant of the TLND problem that we refer to as the *Two-Level Steiner Tree* (TLStT) problem, we are given a set $R \subset V$ representing the customers that have to be served either by primary or secondary technology. The set of primary customers, P, is such that $P \subseteq R$. The goal is to find a minimum-cost Steiner tree in G spanning all nodes from R and such that all nodes from P are connected with each other using the primary technology. Using the notation presented before, binary vector $\mathbf{X}$ instead of being associated with a spanning tree of G is now associated with a Steiner tree connecting nodes from R. The remaining conditions remain the same ($\mathbf{Y}$ is associated with a Steiner Tree connecting P, $\mathbf{Y} \leq \mathbf{X}$, and the objective function is given by (1)). Those nodes that do not belong to R but that are spanned by a solution given by a pair $(\mathbf{Y}, \mathbf{X})$ are called *Steiner-nodes*.

The RRTLStT Problem For the Recoverable Robust counterpart of the TLStT problem (RRTLStT) the set $R \subset V$ is given at the outset, while the set of primary customers is only determined after the uncertainty is resolved (i.e., the scenarios are such that $P^k \subseteq R$, $\forall k \in K$).

The MIP formulation provided for the RRTLND problem can be easily adapted for the RRTLStT problem by imposing that feasible values of vector $\mathbf{x}$ instead of being associated with a spanning arborescence of G_A, have to instead be associated with a Steiner arborescence of set R. This is expressed by replacing $\mathbf{x}$-cuts by

$$\mathbf{x}\left(\delta^{-}(S)\right) \geq 1,\ \forall S \subseteq V \setminus \{r\},\ S \cap R \neq \emptyset. \tag{16}$$

This set of constraints, which we call $\mathbf{x_R}$-cuts, ensures that there is a directed path from r to every node in $R\setminus\{r\}$.

In the algorithmic framework outlined in §3.2 some procedures should be adapted for solving the RRTLStT problem. In the MIP initialization, the "=" sign in (14) should be replaced by "$\leq$". In the separation described in Algorithm 1, $\mathbf{x_R}$-cuts are separated instead of $\mathbf{x}$-cuts; in this case instead of selecting a random node v in $V\setminus\{r\}$ and performing the separation from r to v, the separation is performed from r to every node in $v \in R\setminus\{r\}$. When applying Combinatorial-Cuts, instead of giving as input a spanning arborescence $\tilde{T}$ of G_A, we give as input a Steiner arborescence which

spans all nodes in R; this arborescence is found by means of an algorithm that succesively solves shortest-path problems from r to $v \in R \backslash \{r\}$ with arc costs given by (13) and merges these paths to conform an arborescence of G_A spanning R. The same idea is used in our primal heuristic, in which instead of finding an spanning arborescence of G_A we find a Steiner arborescence connecting nodes in R.

5. Computational Results

In this section we report on our computational experience on two sets of instances that are used to test the branch-and-cut algorithm for both, the RRTLND problem and the RRTLStT problem.

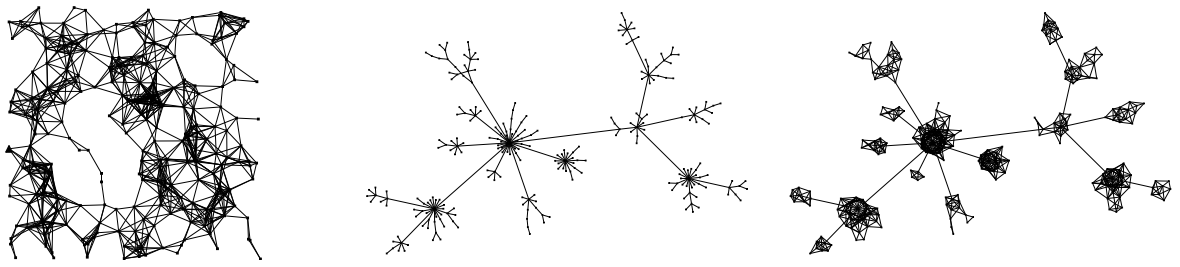
All the experiments were performed on an Intel Core™ i7 (2600) 3.4GHz machine with 16 GB RAM, where each run was performed on a single processor. The branch-and-cut was implemented using CPLEX™ 12.3 and Concert Technology framework. All CPLEX parameters were set to their default values, except the following ones: (i) All cuts were turned off, (ii) heuristics were turned off, (iii) preprocessing was turned off, (iv) time limit was set to 1800 seconds, and (v) higher branching priorities were given to $\mathbf{y^0}$ variables. We have turned these CPLEX features off in order to make a fair assessment of the performance of the techniques described in §3.2.

Interestingly, and somewhat to our surprise, it turns out that turning on CPLEX cuts and heuristics actually slows down the performance of our algorithm. With regards to the branching priorities, notice that whenever a variable, say y^0_ℓ, is fixed to one, variables x_ℓ and y^k_ℓ, $\forall k \in K$ can be immediately fixed as well ($x_\ell = 1$ and $y^k_\ell = 0\ \forall k \in K$). Furthermore, we know that only a few $\mathbf{y^0}$ variables (at most $n-1$) will, at the end of the optimization, take the value of 1. Therefore, we give higher branching priorities to these variables, as they mostly influence the overall solution structure and reduce the underlying search space.

5.1. Instances

We consider two classes of randomly generated instances, that we refer to as `G` and `SC` instances. Their topologies resemble different geographic local structures of communication and distribution networks.

`G` Instances This group of instances is generated following a similar scheme as Johnson et al. (2000) (where the authors intended to generate instances that coincide with the street maps of real-world instances). Here n nodes are randomly located in a unit Euclidean square. An edge e between two nodes is created if the Euclidean distance between them is no more than $\alpha/\sqrt{n}$, for a fixed $\alpha > 0$. Coordinates are generated with five significant digits. The secondary cost of an edge b_e corresponds to the Euclidean distance between its end points multiplied by 10^4 and rounded to the closest integer; the primary cost a_e is calculated as $(1+\beta)\, b_e$, where $\beta \in [0,1]$ is a pre-defined parameter and the recovery cost $r_e = r^k_e$ is assumed to be equal for all $k \in K$ and is set to $(\epsilon+\beta)\, b_e$,



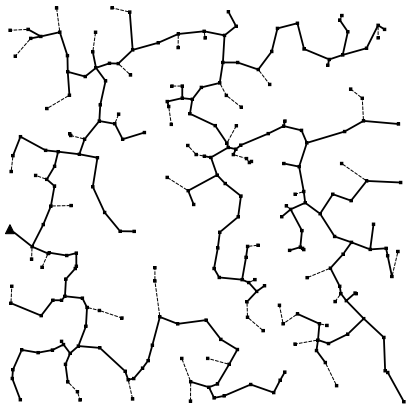
(a) G instance with $\alpha = 0.6$. (b) *Scale-free* tree with $n = 250$. (c) SC instance with $\alpha = 0.2$.

Figure 2 Examples of the generated instances.

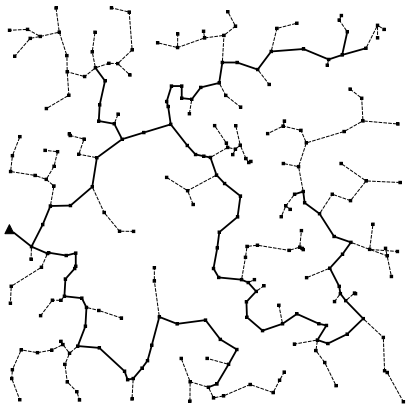
for a fixed $\epsilon \in [0, 1]$. Both, primary and recovery costs are rounded to the nearest integer value. A single node is randomly selected and chosen to be the root node r. For the RRTLND problem, in each scenario $k \in K$, $\pi\%$ of nodes are uniformly randomly selected from V to constitute the set of primary nodes P^k. For the RRTLStT problem, the set R of *all potential customers* is constructed by uniformly randomly selecting $\varphi\%$ of all nodes from V. Similarly as for the RRTLND problem, $\pi\%$ of all nodes from R are then uniformly randomly selected to build the set P^k, for each $k \in K$.

In our experiments we consider the following parameter settings: $\beta, \epsilon \in \{0.5, 1.0, 2.0, 3.0\}$ (which produces $r_e/u_e \in \{7/6, \ldots, 7\}$), $\pi \in \{10\%, 20\%, 30\%\}$, and $\varphi = 50\%$. Four instances were generated for each combination of those parameters. Graphs of different size are considered as well. We choose $n \in \{50, 75, 100, 250\}$ and set $\alpha = 0.6$. The value of α is incremented in steps of 0.001 until a connected graph is obtained (in only one case, for $n = 250$, 0.6 was not enough to define a connected graph and the real value of α was 0.613). This leads to 192 instances for a given n. Figure 2(a) illustrates an example of a graph with 250 nodes and $\alpha = 0.6$ (which produces 1134 edges).

SC Instances These instances are generated on the basis of the well-known *scale-free* networks (see Barabasi and Albert 1999). *Scale-free* networks frequently appear in the context of complex systems, including the World Wide Web, the internet backbone, infrastructure networks, airline connections, cellular networks, wireless networks, electric-power grids and many other contexts. Using the `igraph` library package (see `igraph` Project 2012) a *scale-free* graph of n nodes is created using default settings. This actually produces a tree since linear preferential attachment (*power-law* equal 1) is the default parameter for the generation. The resulting graph is simply an array of binary relations. We then use the yEd Graph Editor software (see yWorks 2012) and draw the tree using the "organic" layout. This layout determines node coordinates which are then used to add additional edges and augment the tree. A new edge between two nodes is added if its Euclidean distance is no more than $\alpha/\sqrt{n}$. The root node corresponds to the node with label 0 in the *scale-free* tree. Edge costs (b_a, a_e, r_e) and scenarios for both the RRTLND problem and the RRTLStT problem are generated identically as the G instances.



(a) $\beta = 0.5$, $\epsilon = 3$, $OPT_{RR} = 50982$, $|E(\mathbf{y^0})| = 187$, $|E(\mathbf{y^k})|\,/\,|K| = 4$.



(b) $\beta = 3$, $\epsilon = 0.5$, $OPT_{RR} = 96795$, $|E(\mathbf{y^0})| = 95$, $|E(\mathbf{y^k})|\,/\,|K| = 34$.

Figure 3 Examples of the solution of the RRTLND problem for a `G` instance with 250 nodes and with different values of β and ϵ.

Note. $\alpha = 0.6$, $|K| = 20$. Bold edges correspond to first-stage primary edges, dashed edges are secondary edges that might be recovered in some scenarios.

In Figure 2(b) we show a *scale-free* tree with a layout fixed by yEd and in Figure 2(c) the same instance augmented with a set of complementary edges (922 in total). For $n = 50, 75, 100, 250, 500, 750, 1000$ we use $\alpha = 0.1, 0125, 0.15, 0.2, 0.3, 0.35, 0.4$ respectively. The other parameters were set as in the case of the `G` instances. Four instances were generated for each combination of the parameters n, π, β and ϵ. This leads to 192 instances for a given n.

5.2. Robustness and Recoverability

In our computations we consider up to 30 scenarios which are created in advance. By doing this, when considering problems with 10 scenarios, we simply use the first 10 scenarios out of those 30. The same applies when considering 20 scenarios. The scenarios are identical for the different values of β and ϵ. By proceeding in this way, it is easier to measure the impact of considering a larger number of scenarios.

The way that *robust* first-stage solutions and the corresponding recovery actions are calculated depends not only on the scenario structure but also on the cost structure; the relations between a_e, b_e and r_e. If the recovery costs r_e are high compared to the first-stage upgrade costs $u_e = a_e - b_e$, then the solutions of the RRTLND problem are more likely to have a larger first-stage primary tree. On the contrary, if recovery is relatively cheap, then the optimal solutions will be comprised by a smaller first-stage primary tree and more recovery actions will be performed (as in a wait-and-see approach). This can be seen when comparing the solutions in Figures 3(a) and 3(b) of a 250 nodes `G` instance with 20 scenarios. In the first case, recovering an edge in the second stage is seven times more expensive than installing a primary technology in the first stage (which is

Table 1 **Solution characteristics and algorithm performance averages for different values of $|K|$ for classes G and SC.**

Class	n	m	$\|K\|$	Gap(%)	Time(s)	$\|E(\mathbf{y^0})\|$	$\|E(\mathbf{y^k})\|/\|K\|$	PH(%)	#BBN's	#(3)	#(4)	#(5)	#Opt
G	50	163	10	0.00	31.27	17	5	7.62	367	25	131	1459	64
			20	0.01	222.60	17	5	6.79	837	30	207	4314	63
			30	0.00	230.31	18	5	7.5	642	28	181	5554	64
	75	257	10	0.00	53.11	24	6	7.91	471	50	151	1986	64
			20	0.09	540.85	25	6	7.02	1197	54	272	6407	56
			30	0.24	1004.85	25	6	6.78	1416	57	264	9292	40
	100	356	10	0.01	458.67	35	9	7.54	1491	105	292	4136	62
			20	0.36	1470.20	35	10	6.61	1056	105	344	9066	23
			30	0.83	1780.54	36	10	6.95	434	103	271	11426	2
	250	1114	10	0.86	†	90	23	9.81	237	64	172	6861	0
			20	6.10	†	111	23	11.26	15	36	37	7497	0
			30	10.67	†	119	23	15.28	5	24	13	6995	0
SC	50	175	10	0.00	32.31	11	6	5.93	198	2	38	409	64
			20	0.00	81.68	11	6	7.48	439	1	63	1162	64
			30	0.00	150.98	12	6	6.45	769	1	84	2167	64
	75	287	10	0.01	196.53	17	8	7.04	4016	15	109	1202	63
			20	0.02	470.32	18	9	7.05	1460	15	151	3126	61
			30	0.08	820.36	18	9	7.23	1486	15	185	5446	49
	100	410	10	0.01	452.42	23	11	7.45	2490	13	149	1540	61
			20	0.08	878.75	25	11	7.75	2731	11	179	3559	42
			30	0.14	1177.93	24	12	7.7	1779	10	212	6096	32
	250	932	10	0.07	1778.68	60	29	5.76	1313	59	288	3826	1
			20	0.17	†	62	32	5.63	518	55	217	6342	0
			30	0.26	†	63	32	5.54	228	53	121	6896	0
	500	2345	10	0.06	†	124	58	5.44	385	36	243	5689	0
			20	0.26	†	126	63	5.26	16	20	93	7706	0
			30	1.53	†	142	65	5.36	1	15	68	9289	0
	750	3460	10	0.08	†	189	87	5.39	132	38	210	7095	0
			20	0.95	†	209	94	5.39	4	20	94	10051	0
			30	3.50	†	241	94	6.38	1	11	50	10622	0
	1000	4658	10	0.16	†	261	114	5.58	65	36	201	8792	0
			20	2.34	†	308	125	5.83	1	16	88	11970	0
			30	6.64	†	367	124	8.34	0	7	33	9911	0

Note. RRTLND problem, $\pi = 10\%$, $\beta, \epsilon \in \{0.5, 1, 2, 3\}$. †: Time limit of 1800 seconds reached.

50% more expensive than secondary technology), consequently the first-stage primary tree (bold edges, $E(\mathbf{y^0})$), spans a large portion of the graph (186 nodes) and only a few recovery actions are needed per scenario ($|E(\mathbf{y^k})|/|K| = 4$). The opposite occurs in the second case, when the recovery cost is slightly more expensive than the upgrade cost (which is four times more expensive than secondary cost); in this case, the $E(\mathbf{y^0})$ component is smaller, spanning only 94 nodes, and much more recovery actions take place in each scenario ($|E(\mathbf{y^k})|/|K| = 37$). The differences in the value of the objective functions, OPT_{RR}, can be explained similarly.

In Table 1 we report average values of the experimental results obtained for the RRTLND problem for classes G and SC considering different number of nodes and different number of scenarios (columns Class, n and $|K|$ respectively). The presented statistics concern the solution characteristics as well as indicators of the algorithmic performance. Column m corresponds to the average number of edges among the instances created for each value of n. Column Gap(%) shows the average gap obtained after the time limit of 1800 seconds is reached. This average is calculated over 64 instances per each group. The corresponding average running times are shown in seconds in

Table 2 **Running times and gap statistics of all instances of classes G and SC for different values of $|K|$.**

		Running times statistics ($t \leq 1800s$)					Gap (%) statistics ($t > 1800s$)				
Class	$\|K\|$	#	Min	Median	Mean	Max	#	Min	Median	Mean	Max
G	10	190	0.30	40.11	164.00	1690.00	66	0.02	0.25	0.84	12.39
	20	142	2.56	189.40	372.80	1605.00	114	0.07	0.62	3.68	22.98
	30	106	6.79	216.80	360.00	1594.00	150	0.07	0.78	5.01	29.60
SC	10	189	0.72	60.00	194.20	1787.00	259	0.01	0.05	0.10	1.14
	20	167	4.60	125.60	278.70	1758.00	281	0.03	0.22	0.87	6.39
	30	145	5.59	222.20	364.80	1535.00	303	0.03	0.90	2.56	13.86

Note. RRTLND problem, $\pi = 10\%$, $\beta, \epsilon \in \{0.5, 1, 2, 3\}$.

column Time(s). The average size of the first-stage primary subtree of the optimal, or best known feasible solution, is indicated in column $|E(\mathbf{y^0})|$. The mean number of recovery actions performed in each scenario can be expressed by $|E(\mathbf{y^k})|$ divided by $|K|$; the average values of this measure, for the optimal or best known solution, are reported in column $|E(\mathbf{y^k})|\,/\,|K|$. In column #Opt the number of problems that can be solved to optimality (out of 64 for each row) is shown.

A first-stage solution is expected to be more robust with respect to data perturbations if more scenarios (possible data realizations) are taken into account. However, this robustness is not for *free*. On the one hand the difficulty of the problem increases since a larger search space should be considered; while on the other hand, the cost of the solutions, OPT_{RR}, increases due to a possible enlargement of the first-stage primary component or because a new worst-case scenario induces a higher robust recovery cost. The first phenomenon is what we call the *Effort for Robustness*. Table 1 demonstrates this phenomenon. Increasing the number of scenarios results in a deterioration of the algorithmic performance for both classes of instances: (i) the average running times increase (this is more apparent in the case of small instances, which could be solved to optimality within the time limit); (ii) the average gap of the obtained solutions deteriorates; and, therefore, (iii) the number of solution for which the proof of optimality is obtained decreases. From the perspective of the solutions structure and the corresponding cost, from columns $|E(\mathbf{y^0})|$ and $|E(\mathbf{y^k})|\,/\,|K|$ we can see the size of the first-stage primary tree is almost constant for a given n, as well as the average number of recovery actions performed by scenario. The fact that the average values are almost constant for a given n means that our recoverable robust solutions are protected against data perturbation and are able to balance robustness and recoverability: the robust first-stage solutions and their corresponding recovery actions depend more on the cost structure (as shown in the Figure 3 example) than on the level of uncertainty. Nevertheless, the absolute number of recovery actions ($|E(\mathbf{y^k})|$) increases proportionally to $|K|$, which means that the cost of the corresponding solutions is also likely to increase due to the augmentation of the worst-case recovery cost induced by a new scenario.

Table 2 provides further analysis on the the impact of $|K|$ on the algorithmic performance. We report the statistics (the number of instances (#), min, median, mean and max values) of the

Table 3 Gain of Recovery with respect to the Absolute Robustness and the Wait-and-See approaches for instances of classes G and SC.

			GoR with respect to AR				*GoR* with respect to W&S			
Class	n	$\|K\|$	Min	Median	Mean	Max	Min	Median	Mean	Max
G	75	10	0.00	13.56	13.45	29.89	3.58	16.58	18.48	44.04
		20	1.01	19.35	18.77	37.15	3.52	17.09	19.27	45.16
		30	0.84	20.89	19.84	39.39	3.62	17.46	18.98	44.82
	100	10	0.00	13.15	13.57	31.44	2.39	16.97	18.54	45.35
		20	0.79	19.28	18.77	36.68	3.18	18.66	19.92	45.27
		30	0.96	19.79	19.50	37.58	4.85	19.26	20.29	47.55
SC	75	10	0.00	13.23	13.65	33.71	3.25	15.37	17.32	43.53
		20	0.00	17.56	17.38	38.08	3.28	15.60	17.60	44.46
		30	1.01	20.22	19.23	38.43	1.70	15.85	17.73	43.56
	100	10	0.39	13.81	13.86	31.18	4.31	19.04	21.14	49.02
		20	0.71	16.57	16.14	32.26	3.44	19.88	21.36	48.09
		30	1.03	17.85	17.44	34.24	3.00	19.31	20.87	47.44

Note. RRTLND problem, $\pi = 10\%$, $\beta, \epsilon \in \{0.5, 1, 2, 3\}$.

running times of those problems that are solved to optimality and the statistics of the gaps of those problems that cannot be solved within 1800 seconds; these statistics are summarized for all values of n, β, ϵ and π, for the two classes of instances. Hence, each row summarizes statistics over 256 instances of each group. As observed before, increasing the number of scenarios, $|K|$, clearly deteriorates the performance of the algorithm: the median and mean running times of those problems that are solved to optimality increase notably; while the median, mean and maximum gaps of those problems that cannot be solved within the time limit, and their quantity also increases.

In Table 3 we report basic statistics (Min, Median, Mean, Max) of the values of the *Gain of Recovery* of the recoverable robust solutions with respect to the AR and W&S approaches for a subset of instances of classes G and SC. The economical advantage of the RR solutions is clearly shown by the reported values: both AR and W&S solutions are, in general, more than 15% more expensive than the recoverable robust ones. Moreover, the AR solutions can be 39% more expensive, while the W&S solutions 49% more expensive! This means that the recoverable robustness approach is able to provide economically robust solutions by means of balancing first-stage and second-stage actions depending on the cost and scenario structure.

5.3. Algorithmic Performance

More specific performance measures are presented in the remaining columns of Table 1. In column PH(%) we report the average gap between the *initial upper bound* (obtained by running Algorithm 3 in which $\mathbf{w} = \mathbf{b}$ in Step 1) and the optimal, if known, or the best lower bound attained within the time limit. The average number of nodes of the branch-and-bound tree is shown in column #BBN's. In columns #(3), #(4) and #(5) we summarize the average number of $\mathbf{x}$-, $\mathbf{y^0}$- and *scenario*-cuts, respectively, that are added during the optimization process.

As discussed in §3.2, one of the main features of our branch-and-cut is the embedded primal heuristic. From column PH(%) we observe that, in most cases, this average value is below 10%,

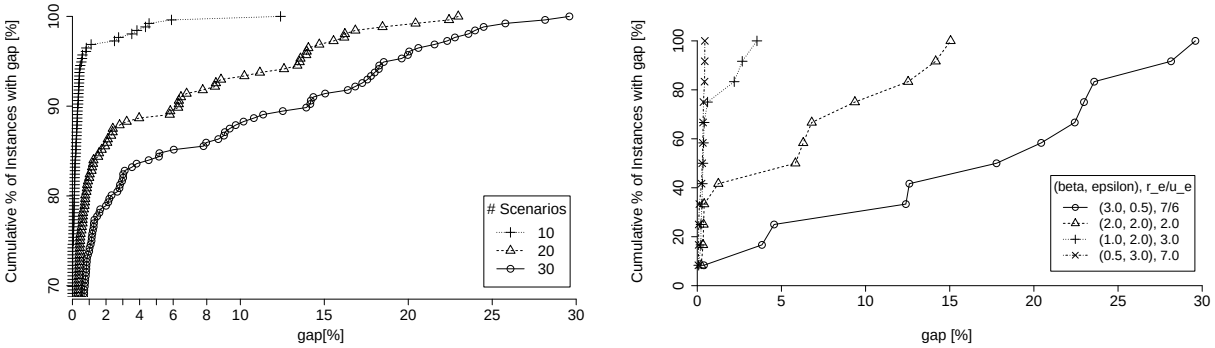
which reinforces our conviction that this procedure is crucial as part of the algorithmic approach. These initial upper bounds can be obtained in a couple of seconds or even fractions of a second for small instances.

For small instances (50 and 75 nodes for `G` instances, and 50 nodes for `SC` instances), we notice that the number of nodes of the branch-and-bound tree increases with the number of scenarios. However, for larger instances the situation is the opposite: an increased number of scenarios implies a reduced value of #BBN's. The more scenarios we consider, the more complex the problem is. In some cases, especially for the largest instances, only a few nodes are explored or, even worse, no branching is performed and the optimization terminates while cutting planes are still being added at the root node.

With respect to the separation of $\mathbf{x}$-, $\mathbf{y^0}$- and *scenario*-cuts, the first observation is that, for small and medium size instances, when increasing the number of scenarios the number of $\mathbf{x}$- and $\mathbf{y^0}$-cuts that are added is approximately constant and, the number of *scenario*-cuts increases proportionally. Additionally, as might be expected, for a given n and a given $|K|$, more *scenario*-cuts are added than $\mathbf{y^0}$-cuts, and more $\mathbf{y^0}$-cuts are added than $\mathbf{x}$-cuts. These behaviors are not verified for larger instances, which is probably due to the fact that in this case the separation is mainly performed at the root node, while it is actually during branching that the separation reaches a more stable behavior. Despite the differences, it is interesting to notice that, in general, not many cutting planes are needed to obtain strong lower bounds, which is the case for a large percentage of instances.

In Figure 4(a), we show the cumulative percentage of problems of class `G`, for different values of $|K|$, for which we reach less than a given gap (%) within the time limit (for each number of scenarios there are 256 problems to be solved in class `G`). This complements the information presented in Tables 1 and 2 about the average gap in relation to the number of scenarios. For 10 scenarios, we notice that more than 95% of problems can be solved with less than a 2% gap within the time limit, and only a few outliers present gaps greater than 5%. When considering problems with 20 scenarios approximately 85% of the instances are solved to within a 2% gap in the time limit. In this case, almost 10% of the instances present a gap larger than 10%, which can be even higher than 20% for a few cases (less than 2% of the problems). However, when considering $|K| = 30$, the quality of the solutions significantly deteriorates. More than 15% of the instances present gaps greater than 10% when reaching the time limit, and these gaps are even higher than 25% for a few problems.

Figure 4(b) considers the group of instances with 250 nodes of class `G` (considering $|K| = \{10, 20, 30\}$ and $\pi = 10\%$) and provides the cumulative percentage of problems (%), for four combinations of β and ϵ, for which we reach less than a given gap (%) within the time limit. It follows that when the recovery costs are significantly higher than the upgrade costs the problem turns out



(a) All instances of group G for different values of $|K|$ ($\pi = 10\%$, $\beta, \epsilon \in \{0.5, 1, 2, 3\}$, RRTLND problem)).

(b) Influence of β and ϵ in the algorithm performance for the 250 nodes group of class G ($|K| = \{10, 20, 30\}$, $\pi = 10\%$, RRTLND problem).

Figure 4 Cumulative percentage of instances with a given gap (%) obtained within the time limit for the RRTLND problem.

to be easier to solve. This can be explained by the fact that if recovery costs are expensive, then the induced solutions tend to be comprised of a larger first-stage primary component (reducing the number of recovery actions, see Fig. 3(a)). These solutions have a closer resemblance to the easier deterministic TLND problem with $P = \bigcup_{k \in K} P^k$. On the other hand, when recovery costs are more "comparable" to first-stage upgrade costs the structure of solutions has more of a "wait-and-see" flavor: the first-stage primary component is smaller and a large number of recovery actions is performed in the second stage; this emphasizes the combinatorial nature of the problem and it makes the optimization task harder.

In all the results analyzed so far, we have considered $\pi = 10\%$ (in each scenario 10% of the nodes are primary nodes). However, and in order to provide an accurate evaluation of our algorithm we have performed computations by also considering $\pi = 20\%$ and $\pi = 30\%$. For both class G and class SC we selected the group of instances with 100 nodes and tested the developed algorithm for $\beta, \epsilon \in \{0.5, 1, 2, 3\}$ and $|K| = \{10, 20, 30\}$, considering $\pi = 20\%$ and $\pi = 30\%$. For each value of π, 256 problems are solved. In Table 4 we report the statistics regarding the running times of those instances that are solved to optimality and the statistics of the gaps of those that reached the time limit before optimality. We observe that increasing the fraction of nodes that are primary in each scenario results in a fewer number of instances that are solved to optimality. However, the gap statistics (over the instances not solved to optimality) are similar for different values of π, in particular the median and mean values remain in all cases below 1%. Hence we may conclude that the overall quality of the solutions produced by our algorithm is not significantly affected for different values of π.

Table 4 **Influence of the value of π on the algorithmic performance for instances of both classes G and SC.**

		Running times statistics ($t \leq 1800s$)					Gap (%) statistics ($t > 1800s$)				
Class	π	#	Min	Median	Mean	Max	#	Min	Median	Mean	Max
G	10%	87	6.01	368.80	555.80	1690.00	105	0.07	0.48	0.73	5.18
	20%	57	5.85	486.80	605.80	1630.00	135	0.02	0.47	0.60	3.95
	30%	64	6.13	506.30	649.10	1790.00	128	0.01	0.37	0.43	1.41
SC	10%	135	12.01	263.60	429.00	1787.00	57	0.03	0.21	0.23	0.67
	20%	71	1.23	486.60	520.20	1785.00	121	0.01	0.23	0.32	3.22
	30%	62	0.97	369.10	524.00	1744.00	130	0.03	0.20	0.22	0.54

Note. RRTLND problem, 100 nodes instances with $\beta, \epsilon \in \{0.5, 1, 2, 3\}$, and $|K| = \{10, 20, 30\}$.

Table 5 **Impact of the branch-and-cut strategies on the algorithmic performance for instances of class G.**

		Running times statistics ($t \leq 1800s$)					Gap (%) statistics ($t > 1800s$)				
n	Separation Strategy	#	Min	Median	Mean	Max	#	Min	Median	Mean	Max
50	Basic	56	2.51	129.00	290.70	1487.00	136	0.01	0.21	0.25	0.99
	+ Mixed Sep.	186	1.44	153.40	267.50	1550.00	6	0.08	0.24	0.25	0.50
	+ Comb. Cuts	191	0.30	62.79	152.80	1589.00	1	0.46	0.46	0.46	0.46
75	Basic	56	4.23	342.50	463.00	1700.00	136	0.01	0.15	0.27	2.84
	+ Mixed Sep.	142	4.09	275.60	430.10	1712.00	50	0.05	0.45	0.60	3.01
	+ Comb. Cuts	160	0.98	128.20	279.50	1594.00	32	0.07	0.45	0.63	3.02
100	Basic	40	27.13	457.40	650.60	1724.00	152	0.01	0.38	0.59	6.09
	+ Mixed Sep.	64	27.80	527.90	637.50	1773.00	128	0.03	0.59	0.89	5.14
	+ Comb. Cuts	87	6.01	368.80	555.80	1690.00	105	0.07	0.48	0.73	5.18
250	Basic	0	-	-	-	-	192	0.03	16.36	17.70	44.50
	+ Mixed Sep.	0	-	-	-	-	192	0.02	3.69	7.99	30.65
	+ Comb. Cuts	0	-	-	-	-	192	0.02	0.99	5.88	29.60

Note. RRTLND problem, $\beta, \epsilon \in \{0.5, 1, 2, 3\}$, $|K| = \{10, 20, 30\}$, and $\pi = 10\%$.

To give clear insights about the utility of the specific separation strategies designed for our algorithmic framework (Mixed Separation and Combinatorial Cuts) Table 5 provides a comparison scheme that helps to evaluate the improvement of the algorithmic performance when including these two procedures. We have selected the groups of instances of class G with 50, 75 and 100 nodes and considered $\beta, \epsilon \in \{0.5, 1, 2, 3\}$, $|K| = \{10, 20, 30\}$, $\pi = 10\%$; therefore, 192 problems were solved for each value of n. Rows denoted by "Basic" correspond to the results obtained without Mixed Separation and Combinatorial Cuts, rows "+Mixed Sep." represent those results obtained when Mixed Separation is included in the separation as described in §3.3, and in rows "+Comb. Cuts" we report the results obtained when also the Combinatorial Cuts are included. The most important indicator is the number of instances that can be solved to optimality and the average time needed to solve them. As can be seen the performance of the algorithm notably improves when mixed separation and combinatorial cuts are turned on. For the group of instances with 250 nodes, only the gaps are compared since no instance could be solved to optimality. Before we conclude this section, we should note that our instances are motivated by real-world applications and are thus somewhat sparse. It should be clear that as graph density increases the number of variables in our model will increase and the performance of the branch-and-cut algorithm will deteriorate.

Table 6 **Solution characteristics and performance measures for different values of $|K|$ for classes G and SC.**

Class	n	m	$\|K\|$	Gap(%)	Time (s)	$\|E(\mathbf{y^0})\|$	$\|E(\mathbf{y^k})\|/\|K\|$	PH(%)	#BBN's	#(16)	#(4)	#(5)	#Opt
G	50	163	10	0.00	24.01	12	3	13.25	677	207	60	330	64
			20	0.00	51.73	13	3	11.32	202	218	98	867	64
			30	0.00	80.54	13	3	10.36	288	227	110	1369	64
	75	257	10	0.00	78.52	18	4	13.62	260	318	125	1496	64
			20	0.14	768.01	18	4	13.49	577	352	220	3973	49
			30	0.38	1250.94	19	4	13.81	274	346	195	5470	33
	100	356	10	0.00	341.89	22	6	16.65	637	732	338	1421	64
			20	0.34	1154.46	22	6	16.31	653	726	365	3315	34
			30	1.00	1435.43	22	6	17.05	323	687	287	4379	21
	250	1114	10	1.45	†	55	14	19.24	204	505	0	4604	0
			20	7.58	†	64	14	22.90	8	252	0	5157	0
			30	13.24	†	71	14	28.16	1	157	0	5262	0
SC	50	175	10	0.00	21.77	7	3	3.93	251	167	27	96	64
			20	0.00	31.07	7	4	4.01	58	165	32	209	64
			30	0.00	42.73	8	3	4.15	154	163	47	356	64
	75	287	10	0.00	62.38	10	5	5.34	124	371	57	302	64
			20	0.00	120.31	10	6	4.63	352	364	77	631	64
			30	0.01	155.90	10	5	4.7	282	373	87	1009	63
	100	410	10	0.01	159.72	12	7	2.87	5192	501	84	318	63
			20	0.01	276.45	12	7	3.09	3372	511	130	728	60
			30	0.01	302.83	12	7	3.66	2013	496	136	1142	62
	250	932	10	0.04	1050.02	29	18	4.15	1581	1782	315	1194	45
			20	0.12	1400.67	28	18	4.16	1025	1786	315	2347	25
			30	0.29	1581.26	31	19	4.12	448	1733	245	3151	19
	500	2345	10	0.10	1778.98	61	37	5.98	972	425	0	4179	2
			20	0.19	†	61	39	5.79	56	188	0	6262	0
			30	0.72	†	62	39	5.85	3	125	0	8074	0
	750	3460	10	0.18	†	95	56	5.97	172	454	0	5344	0
			20	1.15	†	101	57	6.35	1	164	0	8436	0
			30	3.75	†	111	60	8.02	0	94	0	10196	0
	1000	4658	10	0.59	†	134	68	6.73	95	570	0	7245	0
			20	2.37	†	141	76	7.73	1	193	0	11440	0
			30	6.52	†	160	81	10.83	0	103	0	12947	0

Note. RRTLStT problem, $\pi = 10\%$, $\beta, \epsilon \in \{0.5, 1, 2, 3\}$. †= Time limit of 1800 seconds reached.

5.4. Results for the RRTLStT Problem

For the RRTLStT problem, we performed the same computational experiments as the RRTLND problem. The corresponding adaptations of the branch-and-cut algorithm were previously described in §4.

Robustness and Recoverability As expected, for the RRTLStT problem the *Effort for Robustness* is *paid* as well. As for the RRTLND problem, increasing the number of scenarios results in a deterioration of the algorithmic performance which can be seen from the columns Gap(%), Time(s) and #Opt of Table 6. In general, the average value of these indicators are slightly better than for the RRTLND problem.

A deeper analysis can be done on the basis of the results presented in Table 7, where statistics of the running times and of the gaps are presented for both classes of instances. We observe that the number of instances that are solved to optimality decreases and the gap of those that are not solved to optimality increases when increasing $|K|$. These measures are quite similar to those for the RRTLND problem in the case of G instances; but it seems that on average for the SC instances

Table 7 Running times and gap statistics of all instances of classes G and SC.

		Running times statistics ($t \leq 1800s$)					Gap (%) statistics ($t > 1800s$)				
Class	$\|K\|$	#	Min	Median	Mean	Max	#	Min	Median	Mean	Max
G	10	192	2.86	61.07	148.10	1349.00	64	0.10	0.43	1.45	12.08
	20	147	4.91	126.00	308.40	1728.00	109	0.08	1.39	4.73	22.27
	30	118	6.27	136.30	371.60	1750.00	138	0.12	1.51	6.77	27.59
SC	10	238	1.44	46.16	204.40	1708.00	210	0.01	0.08	0.27	3.48
	20	213	3.53	54.72	185.40	1552.00	235	0.02	0.31	1.04	6.58
	30	208	4.85	78.50	224.70	1730.00	240	0.02	1.56	3.01	16.42

Note. RRTLStT problem, $\beta, \epsilon \in \{0.5, 1, 2, 3\}$, $\pi = 10\%$.

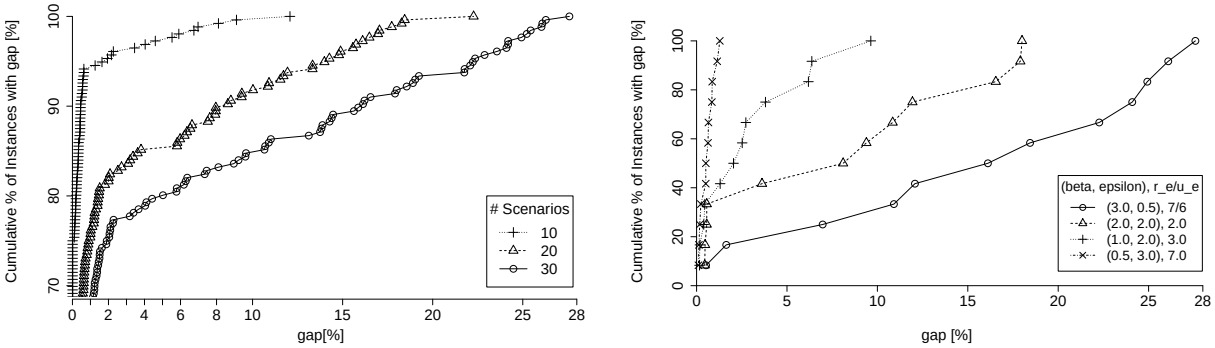
the effort for robustness is "lower" than for the RRTLND problem.

As can be seen from the columns $|E(\mathbf{y^0})|$ and $|E(\mathbf{y^k})|\,/\,|K|$ of Table 6, just like the RRTLND problem there is a clear balance between the robustness of the first-stage solutions and their recoverability. In this case, again the cost structure has more influence on the configuration of solutions than the level of uncertainty.

Algorithmic Performance For the class G the average values of PH(%) in Table 6 are considerably worse than those for the RRTLND problem presented in Table 1 (the values are almost doubled). Nevertheless, for the case of class SC the first primal solutions are, on average, as good as for the RRTLND problem. The fact that $\mathbf{x}$, instead of defining a spanning arborescence on G_A, actually defines a Steiner arborescence on R, helps to explain this. The initial secondary Steiner arborescence on which we calculate the corresponding feasible solution is obtained by means of a heuristic procedure as explained in §4; while in the case of the RRTLND problem we find the primal solution on the optimal spanning arborescence with costs equal to b_e, $\forall e \in E$.

The average number of explored branch-and-bound nodes (column #BBN's) has, more or less, the same order of magnitude and the same dependance on n and $|K|$, as in the case of the RRTLND problem. From columns #(16), #(4) and #(5), where the average numbers of inserted $\mathbf{x_R}$-, $\mathbf{y^0}$- and *scenario*-cuts are reported, we notice that the separation process behaves differently from the one of the RRTLND problem. Since the separation of $\mathbf{x_R}$-cuts is performed by solving a max-flow from r to all nodes in $R\backslash\{r\}$ (instead of a max-flow from r to a single node in $V\backslash\{r\}$), more $\mathbf{x_R}$-cuts are added compared to the number of $\mathbf{x}$-cuts that are added for the RRTLND problem. We observe that fewer $\mathbf{y^0}$-cuts are inserted during the separation than for the RRTLND problem. This can be explained by the size of the primary subtree built in the first stage which is much smaller for the RRTLStT problem.

In Figure 5(a) we find further insights about the quality of the solutions for the RRTLStT problem for class G and its dependence to $|K|$. The results are analogous to those for the RRTLND problem. The influence of the cost structure, which depends on β and ϵ, on the algorithmic performance is outlined in Figure 5(b), where results for the group of instances with 250 nodes are shown.



(a) All instances of group G for different values of $|K|$ for the RRTLStT problem ($\beta, \epsilon \in \{0.5, 1, 2, 3\}$, $\pi = 10\%$)

(b) Influence of β and ϵ in the algorithm performance for the 250 nodes group of class G ($\pi = 10\%$, $|K| = \{10, 20, 30\}$, RRTLStT problem)

Figure 5 Cumulative percentage of instances with a given gap (%) obtained within the time limit for the RRTLStT problem.

6. Conclusions

RRO is a concept that falls within the framework of 2SRO. In a certain sense, RRO can be viewed as two-stage robust optimization with limited recourse (where the practical recovery action constitutes the limited recourse available to the decision maker). It models the practical contexts where a robust solution is desired, but where it is possible to "recover" the solution appropriately (i.e., make it feasible using the limited set of recourse actions available) once the uncertainty is resolved. While there has been a lot of work on robust optimization, the work on 2SRO and RRO, especially in the discrete optimization context is somewhat limited. Our work contributes to the 2SRO and RRO literature in the context of the TLND problem.

The recoverable robust counterpart of the TLND problem studied in this paper addresses uncertainty in the set of primary nodes, which we modeled by means of a set of discrete scenarios. We showed that when the input instance corresponds to a tree, the problem remains NP-Hard and we propose a MIP formulation with a linear number of variables for this case. For general networks we developed a MIP formulation based on cut-set inequalities, and we designed specialized techniques to solve the problem exactly within a branch-and-cut framework. A Steiner variant of the problem was also considered and the exact approach was suitably adapted.

Our branch-and-cut approach was tested extensively on two classes of instances for both the RRTLND and RRTLStT problems. As noted in our experiments the cost structure of the problem has a significant effect on both the solution structure and the running times. We evaluated the benefit of RRO against a traditional (one-stage) Robust Optimization approach and a Wait-and-See approach using a concept termed the *Gain of Recovery* (used previously by Büsing et al. 2011). Our computational results clearly demonstrate the significant benefits of the RRO approach.

Acknowledgments

This work was done during the research visit of E. Álvarez-Miranda and I. Ljubić to the University of Maryland. Support to E. Álvarez-Miranda from the Institute of Advanced Studies of the Università di Bologna (where he was a PhD fellow), I. Ljubić from the APART Fellowship of the Austrian Academy of Sciences (OEAW), and to S. Raghavan from the Smith School Strategic Research Fund are gratefully acknowledged.

References

Atamtürk, A., M. Zhang. 2007. Two-stage robust network flow and design under demand uncertainty. *Oper. Res.* **55** 662–673.

Balakrishnan, A., T. Magnanti, P. Mirchandani. 1994a. A dual-based algorithm for multi-level network design. *Management Sci.* **40** 567–581.

Balakrishnan, A., T. Magnanti, P. Mirchandani. 1994b. Modeling and heuristic worst-case performance analysis of the two-level network design problem. *Management Sci.* **40** 846–867.

Barabasi, A., R. Albert. 1999. Emergence of scaling in random networks. *Science* **286** 509–512.

Ben-Tal, A., L. El-Ghaoui, A. Nemirovski, eds. 2010. *Robust Optimization*. 1st ed. Princeton University Press.

Ben-Tal, A., A. Goryashko, E. Guslitzer, A. Nemirovski. 2004. Adjustable robust solutions of uncertain linear programs. *Math. Programming, Ser.A* 351–376.

Ben-Tal, A., A. Nemirovski. 2000. Robust solutions of linear programming problems contaminated with uncertain data. *Math. Programming, Ser.B* 411–421.

Bertsimas, D., M. Sim. 2003. Robust discrete optimization and network flows. *Math. Programming, Ser.B* 49–71.

Birge, J., F. Louveaux. 2011. *Introduction to Stochastic Programming*. 2nd ed. Springer.

Büsing, C. 2012. Recoverable robust shortest path problems. *Networks* **59** 181–189.

Büsing, C., A. Koster, M. Kutschka. 2011. Recoverable robust knapsacks: the discrete scenario case. *Optimization Letters* **5** 379–392.

Chamberland, S. 2010. On the design problem of two-level IP networks. *Proceedings of the IEEE 14th Symposium on International Telecommunications Network Strategy and Planning*. 1–6.

Cherkassky, B., A. Goldberg. 1995. On implementing push-relabel method for the maximum flow problem. E. Balas, J. Clausen, eds., *Proceedings of IPCO IV*, *LNCS*, vol. 920. Springer, 157–171.

Chopra, S., C. Tsai. 2002. A branch-and-cut approach for minimum cost multi-level network design. *Discrete Mathematics* **242** 65–92.

Cicerone, S., G. Di Stefano, M. Schachtebeck, A. Schöbel. 2012. Multi-stage recovery robustness for optimization problems: A new concept for planning under disturbances. *Information Sciences* **190** 107–126.

Costa, A., P. França, C. Lyra. 2011. Two-level network design with intermediate facilities: An application to electrical distribution systems. *Omega* **39** 3–13.

Current, J. 1988. Design of a hierarchical transportation network with transshipment facilities. *Transportation Sci.* **22** 270–277.

Current, J., C. ReVelle, J. Cohon. 1986. The hierarchical network design problem. *Eur. J. Oper. Res.* **27** 57–66.

Duin, C., A. Volgenant. 1989. Reducing the hierarchical network design problem. *Eur. J. Oper. Res.* **39** 332–344.

Duin, C., T. Volgenant. 1991. The multi-weighted steiner tree problem. *Annals of Operations Research* **33** 451–469.

Garg, N., V. Vazirani, M. Yannakakis. 1997. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica* **18** 3–20.

Gollowitzer, S., L. Gouveia, I. Ljubić. 2013. Enhanced formulations and branch-and-cut for the two level network design problem with transition facilities. *Eur. J. Oper. Res.* **225** 211–222.

Gouveia, L., J. Telhada. 2008. The multi-weighted steiner tree problem: A reformulation by intersection. *Computers & OR* **35** 3599–3611.

Grötschel, M., C. Monma, M. Stoer. 1992. Facets for polyhedra arising in the design of communication networks with low-connectivity constraints. *SIAM J. Optim.* **2** 474–504.

`igraph` Project, The. 2012. The `igraph` library for complex network research. URL `http://igraph.sourceforge.net/`.

Johnson, D., M. Minkoff, S. Phillips. 2000. The prize collecting Steiner tree problem: theory and practice. *Proceedings of the 11th Symposium on Discrete Algorithms*. 760–769.

Kouvelis, P., G. Yu, eds. 1997. *Robust discrete optimization and its applications*. 1st ed. Kluwer Academic Publishers.

Liebchen, C., M. Lübbecke, R. Möhring, S. Stiller. 2009. The concept of recoverable robustness, linear programming recovery, and railway applications. R. Ahuja, R. Möhring, C. Zaroliagis, eds., *Robust and Online Large-Scale Optimization*, *LNCS*, vol. 5868. Springer, 1–27.

Ljubić, I., R. Weiskircher, U. Pferschy, G. Klau, P. Mutzel, M. Fischetti. 2006. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Math. Programming, Ser.B* 427–449.

Mirchandani, P. 1996. The multi-tier tree problem. *INFORMS J. Comput.* **8** 202–218.

Obreque, C., M. Donoso, G. Gutiérrez, V. Marianov. 2010. A branch and cut algorithm for the hierarchical network design problem. *Eur. J. Oper. Res.* **200** 28–35.

Pirkul, H., J. Current, V. Nagarajan. 1991. The hierarchical network design problem: A new formulation and solution procedures. *Transportation Sci.* **25** 175–182.

Sancho, N. 1995. A suboptimal solution to a hierarchial network design problem using dynamic programming. *Eur. J. Oper. Res.* **83** 237–244.

Thiele, A., T. Terry, M. Epelman. 2009. Robust linear optimization with recourse. *Tech. Report TR09-01, University of Michigan* .

yWorks. 2012. yEd Graph Editor. URL `http://www.yworks.com/`.

Zhao, L., B. Zeng. 2012. An exact algorithm for two-stage robust optimization with mixed integer recourse problems. *Tech. Report - University of South Florida* .