Institut für Informatik
Universität zu Köln

Technical Report
March 2014

Single-Commodity Robust Network Design with Finite and Hose Demand Sets*

Valentina Cacchiani[1], Michael Jünger[2], Frauke Liers[3], Andrea Lodi[1], and Daniel R. Schmidt[2]

[1]DEI, University of Bologna, Viale Risorgimento 2, I-40136, Bologna, Italy, {valentina.cacchiani,andrea.lodi}@unibo.it
[2]Institut für Informatik, Universität zu Köln, Albertus-Magnus-Platz, 50923 Köln, Germany, {mjuenger,schmidt}@informatik.uni-koeln.de
[3]Department Mathematik, Friedrich-Alexander Universität Erlangen-Nürnberg, Cauerstraße 11, 91058 Erlangen, Germany, frauke.liers@math.uni-erlangen.de

March 17th, 2014
This preprint also appeared as technical report OR-14-11 at the University of Bologna.

We study a single-commodity Robust Network Design problem (sRND) defined on an undirected graph. Our goal is to determine minimum cost capacities such that any traffic demand from a given uncertainty set can be satisfied by a feasible single-commodity flow. We consider two ways of representing the uncertainty set, either as a finite list of scenarios or as a polytope. We propose a branch-and-cut algorithm to derive optimal solutions to sRND, built on a capacity-based integer linear programming formulation. It is strenghtened with valid inequalities derived as $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts. Since the formulation contains exponentially many constraints, we provide practical separation algorithms. Extensive computational experiments show that our approach is effective, in comparison to existing approaches from the literature as well as to solving a flow based formulation by a general purpose solver.

*We thank the Ateneo Italo-Tedesco VIGONI programme 2011-2012 for its financial support.

1 Introduction

We consider a single-commodity network design problem (sRND) that is robust in the sense of Ben-Tal and Nemirowski [11]: Given an undirected graph $G = (V,E)$ and an (bounded) uncertainty set of traffic demands $\mathfrak{D} \subseteq \mathbb{R}^V$, we can install multiples of a unit capacity on the edges of G. Installing one unit of capacity on $e \in E$ incurs a cost of c_e. The goal is to find minimum cost capacities u such that for all traffic demand vectors $b \in \mathfrak{D}$, there exists a feasible directed single-commodity flow in (G,u) that routes b. Problems of this type arise in the design process of different kind of networks, e.g., transportation and telecommunication networks, but also in energy and gas distribution planning. As even small changes in the traffic demands can cause congestion and network failures, practical solutions should be robust against traffic demands that fluctuate over time or cannot be known with arbitrary precision. This is where *robust* network design comes in.

Most network design models go back to a problem definition by Gomory and Hu [26] where there is a traffic requests r_{ij} for each pair i, j of nodes and r_{ij} units of flow have to be sent from i to j. This means that the underlying flow model is a multi-commodity flow and that the assignment of sources to sinks is fixed, which is not always desirable. Imagine for example a network where several identical servers can answer all the traffic requests of the clients: There, cheaper solutions can be obtained when the optimization process is allowed to map clients to servers instead of using a fixed mapping from the problem input. In that case, the underlying flow model should be a single-commodity flow. One example of such networks would be a movie streaming network [16] or a network of servers for mirrored software distribution. Gas and energy distribution networks also ship a single commodity, but since we do not want to model the complex physical properties of these networks here, we focus on communication networks as our application.

The above single-commodity model is due to Buchheim, Liers and Sanità [41, 16] who additionally assume that $\mathfrak{D}$ is a finite set in the spirit of Minoux [36]. They propose an integer linear programming formulation that is based on arc-flow variables and strengthen it with certain general cutting planes called *target cuts*. The model allows to compute a different routing for each $b \in \mathfrak{D}$. We call this way of routing a *dynamic routing*, as opposed to *static routing* schemes that route all scenarios on the same fixed set of paths. As a consequence, one set of arc-flow variables is needed for each $b \in \mathfrak{D}$. In a previous joint work [4] with Álvarez-Miranda and Parriani, the authors of this article present a linear programming based heuristic for this model. Here, however, we are interested in solving the problem with an exact algorithm and show a different integer programming formulation whose size does not depend on $|\mathfrak{D}|$. Parts of these results appeared in [3] as an extended abstract. Our alternative formulation is based on cut-set and 3-partition inequalities. Both types originally appeared in non-robust network design, see [31] for the original application of these inequalities and [9, 15, 14] for examples of advanced cut-set based branch-and-cut algorithms. Additionally, Atamtürk [6] and Raack, Koster and Wessäly [40] give extensive surveys of the use of these inequalities in network design. Avello, Mattia and Sassano [7] derive a branch-and-cut algorithm for robust multi-commodity network design with a finite demand set that is based on the more general *metric inequalities*.

Moreover, we apply the robustification approach by Ben-Tal and Nemirowski [11] to the above model. In their approach, the uncertainty set $\mathfrak{D}$ is given by a polytope in a linear description. Several prior applications of the approach to multi-commodity network design exist and many of them are again succesfully using cut-set inequalities: Ben-Ameur and Kerivin [10] consider the multicommodity network design problem by [26] with a general demand polytope and static routing. There is an extension to dynamic routing by Mudchanatongsuk, Ordóñez and Liu [37]. Koster, Kutschka and Raack apply the Γ-robustness approach by Bertsimas and Sim [13], using again static routing. Altın, Amaldi, Belotti and Pınar instead consider the Hose uncertainty model that was proposed by Fingerhut, Suri and Turner [23] and Duffield et al. [22]. They also consider a combination with the Γ-robustness model. Mattia [34] considers the Hose model with dynamic routing. We show a natural adaption of the Hose model to single-commodity flows in the model by Buchheim et al. [16] and solve it again with a cut-set model. Pesenti, Rinaldi and Ukovich [39] solve the related Minimum Cost Network Containment problem using cut-set inequalities (see Section 4).

Our contribution. In this paper, we consider the sRND problem and distinguish two ways of representing the uncertainty set $\mathfrak{D}$ in the input: it can be given as a finite list of scenarios or as a linear description of a polytope. Our goal is to determine optimum solutions for the sRND by providing an effective branch-and-cut algorithm in both cases. To this aim, we present a capacity-based ILP formulation. The formulation was introduced in [3] for the case of finite scenario list and uses *cut-set inequalities*. We prove that the corresponding polyhedron is full dimensional and define the conditions under which a cut-set inequality defines a facet. The size of the model depends only on the size of the network, but not on the number of scenarios, as opposed to the flow-based model by Buchheim et al. [16]. On the other hand, it contains exponentially many constraints. We provide a polynomial time algorithm for the separation of cut-set inequalities for the case that $\mathfrak{D}$ is finite. We prove that the separation problem is NP-hard when $\mathfrak{D}$ is given as a polytope, even when $\mathfrak{D}$ is based on an adaption of the Hose model [22]. Still, in this case we propose a practical separation algorithm using a simple mixed integer program (MIP). We strengthen our formulation with 3-partition inequalities and show how to separate them as $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts, as defined by [17]. Extensive computational experiments show that our approach is effective, in comparison to existing approaches from the literature as well as to solving a flow based formulation by a general purpose solver.

General notation and problem definition. For $a \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $n,m \in \mathbb{N}$, we say that $a^T x^* - b$ is the slack of the inequality $a^T x \geq b$ with respect to a vector $x^* \in \mathbb{R}^n$. We say that $a^T x \geq b$ is *binding* for x^* if its slack with respect to x^* is zero and we say that $a^T x \geq b$ is *violated* by x^* if its slack with respect to x^* is negative.

Given an undirected graph $G = (V,E)$ with capacities $u : E \to \mathbb{Z}_{\geq 0}$ and a node balance vector $b \in \mathbb{R}^V$, a directed, single-commodity b-flow is commonly defined as a directed flow $f \in \mathbb{R}^E$

that satisfies the following two conditions.

1. For all nodes $i \in V$, we require that $\sum_{j\in\delta(i)}(f_{ij} - f_{ji}) = b_i$ and call this condition the *flow balance condition*.

2. For all edges $\{i, j\} \in E$, we have $f_{ij} + f_{ji} \leq u_{ij}$. We also say that f *respects* u.

Using this definition we can define the *Single-Commodity Robust Network Design Problem (sRND)* as follows. The input is an undirected graph $G = (V,E)$, an uncertainty set $\mathfrak{D} \subseteq \mathbb{R}^V$ of balance vectors and a cost function $c : E \to \mathbb{R}$ such that installing one unit of capacity on edge e costs c_e. The task is to determine integral capacities $u : E \to \mathbb{Z}_{\geq 0}$ such that for all balance vectors $b \in \mathfrak{D}$ there exists a directed single-commodity b-flow in G that satisfies the capacity conditions with respect to u and minimizes the total capacity installation cost $\sum_{e\in E} c_e u_e$.

Thus, we need to design a network that supports a certain b-flow, but due to uncertain information, we cannot know b exactly. Therefore, we create a set $\mathfrak{D}$ that contains all possible realizations of b and guarantee that no matter what $b \in \mathfrak{D}$ is actually realized, we can route it. We refer to the vectors in $\mathfrak{D}$ as *scenarios* for this reason. If $\mathfrak{D}$ finite, we call the corresponding problem a *finite* sRND problem. If $\mathfrak{D}$ is a polytope, we refer to the underlying problem as the *polytopal* sRND problem.

Finite Versus Polyhedral Uncertainty Sets. The finite and the polytopal sRND problem are equivalent in the following sense: Any finite uncertainty set can be replaced by the polytope defined by its convex hull and any polytopal (i.e., bounded) uncertainty set can be replaced by the finite set of its vertices. Both reductions do not change the set of feasible flows. However, they *do* change the size of the problem input. In general, its size can grow exponentially (see Section 5). Therefore, in any given application, the suitable model needs to be chosen carefully: The finite sRND model should be preferred when the extreme points of the uncertainty set are known and if their number is small. On the other hand, the polytopal model is suited better when the uncertainty set has a small linear description. Thus, we will consider both cases in the scope of this article in spite of their apparent equivalence. Nonetheless, all results that we prove for finite scenario sets also hold for a polytopal scenario set (and vice-versa), as far as they do not concern computational complexity.

Organisation of the article. The paper is organized as follows. In Section 2 we present results that concern both the finite and the polytopal case. Specialized results for both cases follow in Section 3 and Section 4, respectively. We conclude in Section 5 with a branch-and-cut algorithm and computational results.

2 Integer Programming Formulations and Polyhedral Results

We start our considerations with results that concern both the finite and the polyhedral case.

2.1 A Capacity-Based Integer Programming Formulation

In order to obtain a cut-based formulation for the sRND problem, consider some subset $S \subseteq V$ of a graph's node set. We denote the set of edges that have one end-node in S by $\delta(S)$ and call $\delta(S)$ an (edge) cut in G. Consequently, we also call S a cut-set. We can compute the *maximum total balance* of S as $R_S := \max_{b\in\mathscr{D}} \left|\sum_{i\in S} b_i\right|$ and we observe that R_S is exactly the amount of flow that cannot be balanced out within S. At least R_S units of flow must cross the cut $\delta(S)$ and therefore, for any $S \subseteq V$, the capacity of $\delta(S)$ must be at least R_S. This gives rise to the concept of a cut-set inequality.

Definition 1. *Let $G = (V,E)$ be an undirected graph, let $S \subseteq V$ and assume that $\mathfrak{D}$ is a finite or a polyhedral uncertainty set. We then call the inequality*

$$\sum_{\{i,j\}\in\delta(S)} u_{ij} \geq \max_{b\in\mathfrak{D}} \left|\sum_{i\in S} b_i\right| \tag{CS$_S$}$$

the cut-set-inequality *induced by S. We use (CS_S) as a short-hand notation for the inequality and we denote its right hand side by R_S.*

Writing down the cut-set inequalities for all node subsets, we obtain the following integer linear programming problem that will turn out to be a cut-based formulation for the sRND problem:

$$\begin{aligned} \min \quad & \sum_{\{i,j\}\in E} c_{ij} u_{ij} \\ \text{s.t.} \quad & \sum_{\{i,j\}\in\delta(S)} u_{ij} \geq \max_{b\in\mathfrak{D}} \left|\sum_{i\in S} b_i\right| && \text{for all } S \subseteq V \\ & u_{ij} \in \mathbb{Z}_{\geq 0} && \text{for all } \{i,j\} \in E \end{aligned} \tag{IP-CS}$$

Denote by $\mathfrak{P}^f_{\text{sRND}}(G,\mathfrak{D})$ the set of all, possibly fractional, capacity vectors $u \in \mathbb{R}^E_{\geq 0}$ that permit sending a b-flow on G for all scenarios $b \in \mathfrak{D}$ and by $\mathfrak{P}_{\text{sRND}}(G,\mathfrak{D})$ the convex hull of all integer points in $\mathfrak{P}^f_{\text{sRND}}(G,\mathfrak{D})$ (both sets are unbounded). Then, the linear programming relaxation of the cut-set formulation (IP-CS) exactly describes $\mathfrak{P}^f_{\text{sRND}}(G,\mathfrak{D})$, as we show in Theorem 2.

Theorem 2. *A vector $u \in \mathbb{R}^E_{\geq 0}$ is feasible for the linear programming relaxation of the integer program* (IP-CS) *if and only if there exists a feasible b-flow that respects the capacities u in G for all scenarios $b \in \mathfrak{D}$.*

Proof. For the first part, let $u \in \mathfrak{P}^f_{\text{sRND}}(G,\mathfrak{D})$. We consider an arbitrary subset $S \subseteq V$ of the nodes and any scenario $b \in \mathfrak{D}$. By our assumption, there exists a feasible b-flow f on G that respects u. Adding up the flow-conservation conditions for all $i \in S$ yields the well-known result that:

$$\Big|\sum_{i\in S} b_i\Big| = \Big|\sum_{i\in S}\sum_{j\in\delta(i)}(f_{ij}-f_{ji})\Big| = \Big|\underbrace{\sum_{i\in S}\sum_{j\in\delta(i)\cap S}(f_{ij}-f_{ji})}_{=0} + \sum_{i\in S}\sum_{j\in\delta(i)\cap V\setminus S}(f_{ij}-f_{ji})\Big|$$

$$= \Big|\sum_{\{i,j\}\in\delta(S)}(f_{ij}-f_{ji})\Big| \overset{f\geq 0}{\leq} \Big|\sum_{\{i,j\}\in\delta(S)}(f_{ij}+f_{ji})\Big| \leq \sum_{\{i,j\}\in\delta(S)} u_{ij}.$$

This shows that u satisfies all cut-set inequalities.

On the other hand, let u be a feasible vector for (IP-CS) and let again $b \in \mathfrak{D}$ be some arbitrary scenario. To show that $u \in \mathfrak{P}^f_{\text{sRND}}(G,\mathfrak{D})$, we need to show that for every $b \in \mathfrak{D}$, there exists a feasible b-flow under u. Yet, iff for some $b \in \mathfrak{D}$, no feasible b-flow exists in (G,u), then by the MaxFlow-MinCut theorem [24] there exists some cut S with a capacity $\sum_{\{i,j\}\in\delta(S)} u_{ij}$ strictly less than $\left|\sum_{i\in S} b_i\right|$. This is a contradiction to the assumption that u satisfies all cut-set-inequalities. □

In particular, the (non-relaxed) cut-set formulation (IP-CS) exactly characterizes all (infinitely many) integral points in $\mathfrak{P}_{\text{sRND}}(G,\mathfrak{D})$.

Corollary 3. *A capacity vector $u \in \mathbb{Z}^E_{\geq 0}$ is feasible for the sRND instance $(G,\mathfrak{D})$ if and only if it is feasible for the cut-set formulation* (IP-CS). □

The exponential size of the cut-set formulation (IP-CS) naturally raises the question of cut-set constraint separation which we postpone to Sections 3 and 4. To conclude this section, we state that $\mathfrak{P}_{\text{sRND}}(G,\mathfrak{D})$ has full dimension and that the cut-set inequalities are facet-defining for $\mathfrak{P}_{\text{sRND}}(G,\mathfrak{D})$. Both results were already known for the non-robust multi-commodity network design problem [31] since the 90's, before Mattia [34] gave a much shorter proof for the mRND in 2010. Finally, Dorneth observed in his diploma thesis [21] that Mattia's proofs only need small adaptations for the finite sRND. We repeat a more concise version here in order to make the adapted proofs available and to extend them to the polyhedral demand case. As before we define $R_S := \max_{b\in\mathfrak{D}}\left|\sum_{i\in S} b_i\right|$. We let $B^* := \max_{b\in\mathfrak{D}}\sum_{i\in V}|b_i|$ be an upper bound for the capacity on any edge.

Theorem 4 ([34, 21])**.** *For any network $G = (V,E)$ and any (finite or polyhedral) demand set $\mathfrak{D}$, the polyhedron $\mathfrak{P}_{\text{sRND}}(G,\mathfrak{D})$ is full dimensional, i.e. its dimension is $|E|$.*

Proof. We define a capacity vector u^e for every edge $e \in E$ in the following way:

$$u^e_{e'} := \begin{cases} B^*+1, & \text{if } e' = e \\ B^*, & \text{otherwise} \end{cases} \qquad \text{for all } e' \in E$$

and additionally define $U \equiv B^*$ as the capacity vector whose entries are all equal to B^*. Since we have to install at most a capacity of B^* on any edge, we have $U \in \mathfrak{P}_{\text{sRND}}(G,\mathfrak{D})$ and $u^e \in \mathfrak{P}_{\text{sRND}}(G,\mathfrak{D})$ for all $e \in E$. Also, we obtain $|E|$ distinct unit vectors by looking at $u^e - U$ for all $e \in E$. Thus, $\dim(\mathfrak{P}_{\text{sRND}}(G,\mathfrak{D})) = |E|$. □

Theorem 5 ([34, 21])**.** *For any cut $S \subseteq V$, (CS_S) defines a facet of $\mathfrak{P}_{\mathrm{sRND}}(G,\mathfrak{D})$ if and only if $R_S > 0$ and the subgraphs induced by S and $V \setminus S$ are connected.*

Proof. If $R_S = 0$, (CS_S) cannot be stronger than the trivial inequalities $u_e \geq 0$ for $e \in \delta(S)$. Also, if S (or likewise, $V \setminus S$) decomposes into several connected components $S_1,\ldots,S_k$, then summing up the inequalities we get from $S_1,\ldots,S_k$ yields the same left hand side as we get from S; yet, the right-hand side of $(\mathrm{CS}_{S_1}) + \cdots + (\mathrm{CS}_{S_k})$ can only be stronger than the one of (CS_S) by the triangle inequality.

Finally, in order to show that (CS_S) defines a facet of $\mathfrak{P}_{\mathrm{sRND}}(G,\mathfrak{D})$ we define a vector u^e for every edge $e \in E$ in the way suggested by Mattia [34, Theorem 3.14]. In doing so, our choice depends on whether e lies in $\delta(S)$. For all $e \in \delta(S)$, define u^e as

$$u^e_{e'} := \begin{cases} R_S & \text{if } e' \in \delta(S), e' = e \\ 0 & \text{if } e' \in \delta(S),\, e' \neq e \\ B^* & \text{if } e' \notin \delta(S) \end{cases} \qquad \text{for all } e' \in E$$

Now, for all $e \notin \delta(S)$ and some fixed $h \in \delta(S)$ choose u^e as

$$u^e_{e'} := \begin{cases} R_S & \text{if } e' \in \delta(S), e' = h \\ 0 & \text{if } e' \in \delta(S), e' \neq h \\ B^* + 1 & \text{if } e' \notin \delta(S), e' = e \\ B^* & \text{if } e' \notin \delta(S), e' \neq e \end{cases} \qquad \text{for all } e' \in E$$

Because we have $R_S \neq 0$, the vectors $u^e, e \in E$, are linearly independent. This is easily verified by considering the upper triangular matrix with the rows u^e for $e \in \delta(S)$ followed by the rows $u^e - u^h$ for $e \notin \delta(S)$. For all $e \in \delta(S)$, the vector u^e satisfies (CS_S) with equality since $\sum_{e' \in \delta(S)} u^e_{e'} = u^e_e = R_S$ by the definition of u^e. If $e \notin \delta(S)$, we have instead $\sum_{e' \in \delta(S)} u^e_{e'} = u^e_h = R_S$ and again, (CS_S) is satisfied with equality.

It remains to show that $u^e \in \mathfrak{P}_{\mathrm{sRND}}(G,\mathfrak{D})$ for all $e \in E$. We fix an arbitrary cut $X \subseteq V$ such that the subgraphs $G[X]$ and $G[V \setminus X]$ which are induced by X and $V \setminus X$, respectively, are connected and show that u^e satisfies (CS_X) for all $e \in E$. We can assume that $X \neq S$ and that $X \neq V \setminus S$ since we have already shown validity for those two cases. Thus, if $\delta(X) \subseteq \delta(S)$ was true, then either $G[X]$ or $G[V \setminus X]$ would not be connected and therefore, there exists at least one edge $e^* \in \delta(X) \setminus \delta(S)$. Using this observation for any $e \in E$ we have

$$\sum_{e' \in \delta(X)} u^e_{e'} \geq \sum_{e' \in \delta(X) \setminus \delta(S)} u^e_{e'} \geq u^e_{e^*} \geq B^* = \max_{b \in \mathfrak{D}} \sum_{i \in V} |b_i| \geq \max_{b \in \mathfrak{D}} \Big| \sum_{i \in X} b_i \Big|$$

which tells us that u^e satisfies (CS_X). We conclude that (CS_S) defines a face of dimension $|E| - 1$ and, therefore, is a facet of $\mathfrak{P}_{\mathrm{sRND}}(G,\mathfrak{D})$. $\square$

2.2 Valid 3-Partition Inequalities Derived from Chvátal-Gomory Cuts

The cut-set inequalities (CS_S) give a lower bound on the amount of capacity that is needed along the cut that separates a 2-partition $S \subseteq V$ and $V \setminus S$. In general, however, one can ask for lower bounds on the capacity between any k-partition, $k \geq 2$, of the graph. This leads to the definition of k-partition inequalities, an idea that was e.g. explored by [1]. We will see that 3-partition inequalities can be separated as $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts as defined by [17] and elaborate on the details in this subsection. A similar result has been obtained by Magnanti, Michandani and Vachani [32] for a non-robust multi-commodity network design problem.

For any given linear program $Ax \geq b$ with a constraint matrix $A = (a_{ij}) \in \mathbb{Z}^{m\times n}$ and vectors $x \in \mathbb{R}^n, b \in \mathbb{Z}^m$ one can generate a valid inequality for $Ax \geq b$ by selecting some subset $I \subseteq \{1,\dots,m\}$ of the constraints and computing the inequality $\frac{1}{2} \cdot \sum_{j=1}^{n} \sum_{i\in I} a_{ij}x_j \geq \frac{1}{2} \sum_{i\in I} b_i$. If the coefficients $\sum_{i\in I} \frac{1}{2} a_{ij}$ are integral for all $j = 1,\dots,n$, we can round up the right hand side of the inequality and thus obtain a 0-$\frac{1}{2}$-cut [17]. The problem is, of course, to select a suitable set I that generates integral coefficients. Due to the structure of the cut-set inequalities, we can solve this problem if we restrict to $|I| = 3,4$. Indeed, observe that for two non-empty sets $S,T \subsetneq V$ and any vector $u \in \mathbb{R}^E_{\geq 0}$, we have by a counting argument

$$\sum_{e\in\delta(S)} u_e + \sum_{e\in\delta(T)} u_e + \sum_{e\in\delta(S\cup T)} u_e + \sum_{e\in\delta(S\cap T)} u_e = 2\sum_{e\in\delta(S\cup T)} u_e + 2\sum_{e\in(S:T)} u_e + 2\sum_{e\in\delta(S\cap T)} u_e.$$

where $(S : T)$ is defined as the set of edges $\delta(S) \cap \delta(T)$ having one end node in S and one end node in T. Therefore, given cut-set inequalities (CS_S) and (CS_T), we obtain a valid zero-half cut by adding up $\frac{1}{2}((\text{CS}_S) + (\text{CS}_T) + (\text{CS}_{S\cup T}) + (\text{CS}_{S\cap T}))$ to

$$\sum_{e\in\delta(S\cup T)} u_e + \sum_{e\in(S:T)} u_e + \sum_{e\in\delta(S\cap T)} u_e \geq \left\lceil \frac{1}{2}(R_S + R_T + R_{S\cup T} + R_{S\cap T}) \right\rceil. \qquad (\text{ZH}_{S,T})$$

If $R_S + R_T + R_{S\cup T} + R_{S\cap T}$ is odd, the violation of $(\text{ZH}_{S,T})$ with respect to a solution u^* is maximum if $(\text{CS}_S), (\text{CS}_T), (\text{CS}_{S\cup T})$ and $(\text{CS}_{S\cap T})$ are binding for u^*. Therefore, we should select sets S,T where the corresponding cut-set inequalities have small slack.

This observation implies a simple separation algorithm `EnumZH`: We iterate over all pairs $(\text{CS}_S), (\text{CS}_T)$ of binding cut-set inequalities in our constraint set. We then build the corresponding zero-half cut $(\text{ZH}_{S,T})$ and check if it is violated. The running time of the algorithm is quadratic in the number of binding cut-set constraints. While these can be exponentially many (see Figure 1), our experiments show that it pays off to use the algorithm at the root node of the branch and cut tree, see Section 5.

We can replace $(\text{CS}_{S\cup T})$ by $(\text{CS}_{V\setminus(S\cup T)})$ in the above construction without changing $(\text{ZH}_{S,T})$. Then, if S and T are disjoint sets, an edge $e \in E$ has a non-zero coefficient in $(\text{ZH}_{S,T})$ if and only if it is contained in $(S : T)$, $(S : V \setminus (S \cup T))$ or $(T : V \setminus (S \cup T))$. Thus, $(\text{ZH}_{S,T})$ defines a 3-partition inequality for the partitions S, T and $V \setminus (S \cup T)$. In this way, `EnumZH` is a separation heuristic for 3-partition inequalities.

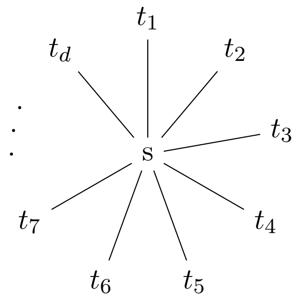

Figure 1: An instance that has a high number of binding cut-set inequalities at the optimum. We define a single scenario: node s has a supply of d and all other nodes $t_1,\dots,t_d$ have a demand of 1. By setting $u^*_e = 1$ for all edges e we obtain a feasible capacity vector and we notice that u^* is a vertex $\mathfrak{P}_{\text{sRND}}$. With respect to u^*, any set $\{s\} \cup X$ with $X \subset \{t_1,\dots,t_d\}$ defines a binding cut-set inequality. Since there are 2^d possible choices for X, we have 2^d binding cut-set inequalities at the basic solution u^*.

3 Robust Network Design with a Finite Scenario List

3.1 A Flow-Based Integer Linear Programming Formulation

When the uncertainty set $\mathfrak{D} = \{b^1,\dots,b^k\}$ is finite, there is a natural, flow-based integer linear programming formulation of the sRND problem. It contains a set of flow variables for each scenario together with the corresponding flow-conservation and capacity constraints [16]:

$$
\begin{aligned}
\min \quad & \sum_{\{i,j\}\in E} c_{ij} u_{ij} & \\
\text{s.t.} \quad & \sum_{\{i,j\}\in E} (f^q_{ij} - f^q_{ji}) = b^q_i & \text{for all } i \in V, q = 1,\dots,k \\
& f^q_{ij} + f^q_{ji} \le u_{ij} & \text{for all } \{i,j\} \in E, q = 1,\dots,k \\
& f^q_{ij}, f^q_{ji} \ge 0 & \text{for all } \{i,j\} \in E, q = 1,\dots,k \\
& u_{ij} \in \mathbb{Z}_{\ge 0} & \text{for all } \{i,j\} \in E
\end{aligned}
\tag{IP-F}
$$

This formulation is similar to classical integer multicommodity flow (MCF) formulations. The only difference is that in the robust context the b-flows $f^1,\dots,f^k$ are not simultaneous and thus do not share the edge capacities. Like for the MCF problem, the finite sRND with integral capacities is NP-hard [41], while its fractional variant can be solved in polynomial time (as is proven by the above compact linear programming formulation). Another property is shared with the MCF problem. While the size of the scenario-expanded formulation (IP-F) is polynomial in the input size, it grows impractically large when the number of scenarios (or commodities, in the MCF case) is high. We therefore concentrate on the cut based formulation for the rest of this article. We notice, however, that both formulations are equivalent in the sense of the following corollary of Theorem 6. In particular, (IP-CS) can be seen as an orthogonal projection of (IP-F).

Corollary 6. *A vector $u \in \mathbb{R}^E_{\geq 0}$ is feasible for the linear programming relaxation of the capacity formulation* (IP-CS) *iff there exist flows $f^1, \dots, f^k$ such that $(f^1, \dots, f^k, u)$ is feasible for the linear programming relaxation of the flow formulation* (IP-F). □

In the non-robust case, the capacity formulation can be obtained by applying Benders' decomposition [12] to the flow formulation, see e.g. [33], and although Benders' original decomposition technique yields a slightly weaker version of (IP-CS), the same principle applies here.

3.2 Polynomial Time Separation of Cut-Set Inequalities

In order to use formulation (IP-CS) in practice, we need a fast separation algorithm for its constraints, i.e., we need to decide if a given capacity vector u^* violates any cut-set constraints on a network $G = (V, E)$ with uncertainty set $\mathfrak{D}$. We show in this section how this can be achieved when $\mathfrak{D}$ is finite.

To this end, we define an auxiliary graph $\hat{G} = (V \cup \{s\}, \hat{E})$ with

$$\hat{E} := E \cup \{(s, \tau) \mid \tau \in V\}.$$

We now iterate over all scenarios in $\mathfrak{D}$. For some fixed scenario $b \in \mathfrak{D}$, we obtain a cost function for the edges of $\hat{G}$ by extending u^* to $\hat{E}$:

$$\hat{u}^*_e := \begin{cases} -b_\tau, & \text{if } e = \{s, \tau\} \\ u^*_e, & \text{otherwise.} \end{cases}$$

Then, we can rewrite the value $b(X \cup \{s\})$ of any minimum s-cut $X \cup \{s\}$ in $\hat{G}$ as

$$\mathrm{val}_b(X \cup \{s\}) = \sum_{e \in \delta_{\hat{G}}(X \cup \{s\})} \hat{u}^*_e = \sum_{\substack{e \in \delta_{\hat{G}}(X \cup \{s\}) \\ s \notin e}} \hat{u}^*_e + \sum_{\substack{e \in \delta_{\hat{G}}(X \cup \{s\}) \\ s \in e}} \hat{u}^*_e = \sum_{e \in \delta_G(X)} u^*_e - \sum_{i \in V \setminus X} b_i.$$

Therefore, any minimum s-cut $X \cup \{s\}$ satisfies that $\sum_{i \in V \setminus X} b_i \geq 0$ – as otherwise, $(V \setminus X) \cup \{s\}$ has a better objective value. As a consequence, the value of $X \cup \{s\}$ is exactly the slack of the cut-set inequality that X would induce if b was the only scenario. The slack of the true cut-set inequality induced by X can only be smaller and therefore we know that if $\mathrm{val}_b(X \cup \{s\}) < 0$, then also

$$0 > \sum_{e \in \delta_G(X)} u^*_e - \sum_{i \in V \setminus X} b_i \geq \sum_{e \in \delta_G(X)} u^*_e - \max_{b \in \mathfrak{D}} \Big| \sum_{i \in V \setminus X} b_i \Big|$$

and X defines a violated cut-set inequality in G. On the other hand, if some $X \subseteq V$ induces a violated cut-set inequality, then there is a scenario $b^* \in \mathfrak{D}$ such that

$$0 > \sum_{e \in \delta_G(X)} u^*_e - \max_{b \in \mathfrak{D}} \Big| \sum_{i \in V \setminus X} b_i \Big| = \sum_{e \in \delta_G(X)} u^*_e - \sum_{i \in V \setminus X} b^*_i = \mathrm{val}_{b^*}(X \cup \{s\})$$

since we can again assume w.l.o.g. that $\sum_{i \in V \setminus X} b_i^* \geq 0$. Thus, by computing a minimum cut on $\hat{G}$ for each scenario, we can find up to $|\mathfrak{D}|$ violated cut-set inequalities or decide that none exist.

In the construction of $\hat{G}$, the signs of the used edge weights are mixed (i.e., positive and negative). In general, the problem of finding a minimum cut in an arbitrary graph with mixed weights is NP-hard. In our case, however, all edges with negative weight are incident to s. This allows us to use a construction for *star-negative graphs* by McCormick, Rao and Rinaldi [35] which reduces the problem to an ordinary minimum s-t-cut problem with non-negative weights. Since this construction changes the size of G by a constant only, we obtain the main theorem of this section.

Theorem 7. *Let $(V,E,\mathfrak{D})$ be an instance of the sRND problem and let $u^* \in \mathbb{R}^E_{\geq 0}$. Then, we can find a cut-set inequality that is violated by u^* or decide that no such inequality exists in time $O(|\mathfrak{D}| \cdot T_{mincut})$, where T_{mincut} denotes the time need to compute a minimum cut in $G = (V,E)$.* $\square$

Any maximum flow algorithm can be used to compute a minimum s-t-cut. We implemented the preflow-push algorithm by Goldberg and Tarjan [25, 19] with the highest label strategy and the gap heuristic. We stop the algorithm when a maximum *pre*flow is found and thus omit its second stage. This results in an overall runtime of $\Theta(|\mathfrak{D}| \cdot |V|^2 \cdot \sqrt{|E|})$ for the separation procedure.

3.3 Separating 3-Partition Inequalities more Efficiently

The assumption that $\mathfrak{D}$ is finite does not only help us to find an efficient separation procedure for cut-set inequalities; it also enables us to find a more efficient alternative to the general 3-partition separation algorithm from Section 2. There, we observed that we can obtain valid 3-partition inequalities by combining two cut-set inequalities with small slack. Instead of enumerating all pairs of binding cut-set inequalities as in Section 2, however, we can now develop an algorithm whose runtime is linear in the number of binding cut-set inequalities.

The key observation for this more efficient algorithm is the following: Our cut-set separation algorithm yields an inequality with maximum violation. Thus, if we try to separate a point u^* that already satisfies all cut-set inequalities, it returns an inequality with *minimum slack*. We use this fact to search for candidates for the zero-half cut generation in our algorithm `MinCutZH`: For each binding cut-set inequality (CS_S) in the current LP relaxation, we call the cut-set separation from the previous subsection on the subgraph $G[S]$ that is induced by S. This yields up to $|\mathfrak{D}|$ cut-sets $T_1 \ldots, T_k \subset S$. By adding up (CS_{T_i}), $(\mathrm{CS}_{S \setminus T_i})$ and $(\mathrm{CS}_{T_i \cup S \setminus T_i}) = (\mathrm{CS}_S)$ we thus obtain one 3-partition inequality for each $i = 1, \ldots, k$. This algorithm mhas a running time of $O(C \cdot |\mathfrak{D}| \cdot T_{mincut})$ where C is the number of binding cut-set inequalities in the current LP relaxation and T_{mincut} again denotes the time needed to compute a minimum s-t-cut in G. It thus depends linearly on the number of binding cut-set inequalities.

Apart from the running time, the algorithm has another advantage over `EnumZH`: There might be good candidate cut-set inequalities that are not part of the current LP solution – and these can only be found by `MinCutZH`. On the other hand, we cannot guarantee that the right hand side of $(\mathrm{CS}_S)+(\mathrm{CS}_T)+(\mathrm{CS}_S)$ is odd and therefore it can happen that `MinCutZH` does not find a violated 3-partition inequality even though one exists (as is the case with `EnumZH`).

4 Robust Network Design with Polyhedral Demand Uncertainties

Duffield et al. [22] propagate the *Hose* demand polytope for multi-commodity network design. Rather than specifying demands for all pairs of nodes (which can be impractical in large networks), they propose to define two bounds for each node i that limit how much flow in total the node i can send to (or receive from, respectively) all other nodes. This is a natural model as these bounds can stem from technical specifications, legal contracts or educated guesses by experienced engineers.

Pesenti, Rinaldi and Ukovich [39] propose a similar model for single-commodity flows: They start from the multi-commodity model and limit the traffic demand r_{ij} for each pair of nodes by an individual upper and lower bound, r_{ij}^{max} and r_{ij}^{min}. Given any such matrix $r=(r_{ij})_{i,j\in V}$ with $r_{ij}^{min}\leq r_{ij}\leq r_{ij}^{max}$, they aggregate the commodities to a demand vector $(b_i)_{i\in V}:=(\sum_{j\in V} r_{ij}-r_{ji})_{i\in V}$. Any demand vector that can be obtained in this fashion is a scenario that needs to be considered in the optimization. This problem is called the *Network Containment Problem* in the literature. Pesenti, Rinaldi and Ukovich subsequently propose to solve the problem with a branch-and-cut algorithm based on a cut-set formulation and a separation MIP.

We propose a different adaptation of the Hose model that is simpler and does not have a point-to-point traffic component. For each node $i\in V$, we define an upper bound b_i^{max} and a lower bound b_i^{min}. We then say that any supply- and demand vector that obeys these bounds *while remaining balanced* is a possible scenario for our optimization. The resulting uncertainty set is the polytope

$$\mathfrak{H}(V,b^{min},b^{max}):=\left\{b\in R^V \;\middle|\; b_i\in[b_i^{min},b_i^{max}] \text{ for all } i\in V \text{ and } \sum_{i\in V} b_i=0\right\}.$$

Due to its similarity to the Hose uncertainty set that is used for multi-commodity network design problems, we call it the *single commodity Hose polytope*. In the following, we assume that our uncertainty set $\mathfrak{D}$ is the polytope $\mathfrak{H}(V,b^{min},b^{max})$ and denote the corresponding (sRND) problem by (sRND-Hose).

4.1 Complexity of Robust Network Design with Single Commodity Hose Demands

Finding an optimum integer solution for (sRND-Hose) is NP-hard, as the problem contains Steiner Tree as a special case (see [41] for a similar reduction for finite $\mathfrak{D}$).

Theorem 8. *The (sRND-Hose) problem is NP-hard.*

Proof. Let $I=(V_I,E_I,c_I,\mathfrak{T})$ be an input for the Steiner Tree problem, i.e., suppose that $G_I=(V_I,E_I)$ is an undirected graph with edge weights c_I and that $\emptyset\subsetneq\mathfrak{T}\subseteq V_I$ is a set of terminals that need to be connected at minimum cost. Steiner Tree is NP-hard [29]. Then, finding an optimum solution for I is equivalent to finding an optimum solution for the following sRND instance J: Select some arbitrary node $s \in \mathfrak{T}$. We set $\hat{b}_s^{min} = 0$ and $\hat{b}_s^{max} = 1$. For all other nodes $i \in \mathfrak{T} \setminus \{s\}$, set $\hat{b}_i^{min} = -1$ and $\hat{b}_i^{max} = 0$. Now, the vertices of $\mathfrak{H}(V_I, \hat{b}^{min}, \hat{b}^{max})$ are exactly the scenarios b where $b_s = 1$ and $b_i = -1$ for some node $i \in \mathfrak{T}$. This means that in any feasible solution for J, there must be a path of capacity 1 from s to all terminals $i \in \mathfrak{T} \setminus \{s\}$. Also, if the support of any feasible integer solution for J contains a cycle, then one edge of the cycle can be deleted. Thus, any optimum solution for J induces a Steiner Tree and any Steiner Tree solution for I defines a solution for J; moreover, the costs of the solutions are identical in both cases. Thus, when $\mathfrak{D} = \mathfrak{H}(V, b^{min}, b^{max})$, solving sRND is at least as hard as solving Steiner Tree. □

We shall see in the remainder of the section that the separation problem for cut-set inequalities is also NP-hard for (sRND-Hose). This proves that (sRND-Hose) remains hard even if we relax the integrality requirement.

4.2 Separating Cut Set Inequalities over $\mathfrak{H}(V, b^{min}, b^{max})$

Finding optimum solutions for the sRND problem in practice becomes significantly harder when the uncertainty set is the polytope $\mathfrak{H}(V, b^{min}, b^{max})$. Following our previous approach, we want to to solve the linear programming relaxation of the capacity-based formulation (IP-CS) in order to generate dual bounds in a branch-and-bound algorithm. As opposed to the case that $\mathfrak{D}$ is finite, however, finding a cut-set inequality with maximum violation will turn out to be NP-hard when $\mathfrak{D} = \mathfrak{H}(V, b^{min}, b^{max})$. The NP-hardness of this problem is somewhat surprising: We could expect to solve the separation problem for (IP-CS) with a minimum cut algorithm. Here, however, the main obstacle is to compute the correct right hand side for a given cut S *inside* of the minimum cut computation. When $\mathfrak{D}$ is finite, we can simply enumerate all possible scenarios b and interpret b as linear node costs that are easily integrated into any minimum cut algorithm. When $\mathfrak{D} = \mathfrak{H}(V, b^{min}, b^{max})$, however, this is no longer possible, as a more sophisticated optimization problem needs to be solved to obtain the correct right hand side for the cut-set inequalities. This is true even though computing the correct right-hand side for a *fixed S* is possible in polynomial time; the difficulty lies in computing it *while computing a minimum cut*.

Summarizing, our problem is that (IP-CS) contains a non-trivial optimization problem on the right hand side. Still, solving such formulations is at the core of robust optimization and several ideas from the literature can be applied here. We observe, for instance, that if we interpret the b_i on the right hand side of (IP-CS) as variables, we obtain a bi-level optimization problem. It minimizes the capacities on the outer level and maximizes the total demands on the its inner level, i.e., the right hand side of each of the cut-set inequalities. Now, if the right hand side optimization problem was a minimization problem, we could collapse (IP-CS) a single level – hoping to obtain inequalities that can be separated more easily. Thus, we only need to replace the linear program $\max_{b \in D} |\sum_{i \in S}|$ by its dual. This technique has been applied successfully to the multi-commodity robust network design problem by Ben-Ameur and Kerivin [10] in the case of static routing. In their case, it results in a separable linear formulation. Applying the same technique to the multi-commodity robust network design problem with *dynamic routing* leads to a non-convex quadratic separation problem, as was shown by Mattia [34]. The same is true in our case. However, when the underlying network flow has several commodities, Mattia observes that linearizing the separation problem yields a mixed integer linear program with big-M constraints. Ben-Tal and Nemirovski [11] give a general solution algorithm for robust linear programs, requiring only that the uncertainty set is compact and that separation over it is possible. They show that any linear program with row-wise uncertainty of this type can be optimized by solving an auxilliary linear program for each row of its deterministic (i.e., non-robust) counterpart. Potentially, each auxilliary problem yields a valid cutting plane for the robust formulation. While we can certainly separate over $\mathfrak{H}(V, b^{min}, b^{max})$, the deterministic counterpart of (IP-CS) unfortunately has an exponential number of rows. We would therefore require an oracle that gives us a row for which the auxilliary problem yields a valid cutting plane. Finding such an oracle is equivalent to solving our original separation problem. Another alternative could be to use a polynomially sized flow-based formulation as deterministic counterpart, but short of introducing a full set of flow-variables for each vertex of $\mathfrak{H}(V, b^{min}, b^{max})$, it is not clear how to robustify the flow-conservation equalities of such a formulation. We conclude that we need to find an alternative to these standard-techniques if we want to solve our separation problem.

Our first step to a practical separation algorithm is to actually write down the separation problem: The following bi-level program will give us a separating hyperplane for any $u^* \notin \mathfrak{P}_{\text{sRND}}(G, \mathfrak{H})$. Since $S \subseteq V$ is variable here, the formulation is not a linear or quadratic program in the strict sense. It can, however, be transformed into a bi-level quadratic program. For now, we stick to the more abstract formulation to benefit from the easier notation. Solving

$$\min_{S \subseteq V} \sum_{e \in \delta(S)} u_e^* - \max_{b \in \mathfrak{H}} \sum_{i \in S} b_i \tag{H-SEP}$$

yields a cut-set inequality that is violated if and only if the optimum objective value of (H-SEP) is negative. As in the finite case, we do not need to take the absolute value of the second sum, as we can assume w.l.o.g. that the *total balance* $\sum_{i \in S} b_i$ is non-negative in an optimum solution (S, b). Moreover, we say that S is a *hose source set* iff $\sum_{i \in S} b_i^{max} \geq 0$ and $\sum_{i \in V \setminus S} b_i^{min} \leq 0$. We only consider hose source sets in the following. If S is not a hose source set, then either $\mathfrak{H}(V, b^{min}, b^{max})$ is empty or $\sum_{i \in S} b_i < 0$ for all $b \in \mathfrak{H}(V, b^{min}, b^{max})$. Finally, we say that a hose source set S is limiting, if $\sum_{i \in S} b_i^{max} \leq -\sum_{i \in V \setminus S} b_i^{min}$. Otherwise, we say that $V \setminus S$ is limiting. We will show next that we can re-write the inner level

$$B_S := \max_{b \in \mathfrak{H}} \sum_{i \in S} b_i \qquad \text{for a fixed } S \subseteq V \tag{MAX-B}$$

such that (H-SEP) reduces to a single-level mixed integer linear program.

We proceed in two steps: First, we give an algorithm that both functions as a scenario separation and proves that there exists a solution of a certain value for (MAX-B). At the same time, we will see that we can compute the value of the solution with a closed formula without actually running the algorithm. This will enable us to integrate the solution value into (H-SEP). Secondly, we prove that our solution maximizes (MAX-B).

To get a better intuition for the algorithm, suppose that $0 \in \mathfrak{H}(V, b^{min}, b^{max})$ and consider the following preliminary method to find an optimum solution for (MAX-B). We start with the vector $b \equiv 0 \in \mathfrak{H}(V, b^{min}, b^{max})$ and our aim is to install as much supply as possible in S. Equivalently, we could try to install as much demand as possible in S, but since we assumed w.l.o.g. that the maximum total balance of S is non-negative, we rather stick to the maximum supply case. We now select an arbitrary node $i \in S$ with $b_i < b_i^{max}$ and another arbitrary node $j \in V \setminus S$ with $b_j > b_j^{min}$. If no such nodes can be found, the algorithm stops. Finally, we increase b_i by one unit and, at the same time, decrease b_j by one unit to maintain a balanced vector.

To analyse the algorithm, we observe that it maintains $\sum_{i \in S} b \leq \sum_{i \in S} b_i^{max}$ and $\sum_{i \in S} b_i = -\sum_{i \in V \setminus S} b_i \leq -\sum_{i \in V \setminus S} b_i^{min}$. The algorithm stops as soon as equality holds in one of the conditions. Thus, if b is the vector that we obtain once the algorithm stops, we have $\sum_{i \in S} b_i = \min\{\sum_{i \in S} b_i^{max}, -\sum_{i \in V \setminus S} b_i^{min}\}$ and we realize that we can compute the value of this solution without actually running the algorithm. Also, increasing the objective value of b further would make b necessarily imbalanced.

The idea of our preliminary algorithm was to start from a feasible vector and to then increase its objective value. We follow the same idea in the case that $0 \in [b_i^{min}, b_i^{max}]$ for all $i \in V$, however, we need a slightly more involved algorithm to do so. The problem is that the starting vector $b \equiv 0$ might be infeasible. More verbosely, the node bounds can force us to install supply on a node in $V \setminus S$ or to install demand on a node in S and thereby change the amount of imbalance that we have to distribute. The bounds can also force us to install a minimum amount of supply or demand on some nodes in S or $V \setminus S$ – which is a problem if we already distributed all the imbalance before reaching such nodes. Both problems can be solved by starting from a different vector. This is why, in contrast to the preliminary algorithm, we start with a vector b that simply satisfies $b_i \in [b_i^{min}, b_i^{max}]$ for all $i \in V$ and then make sure that $\sum_{i \in V} b_i = 0$ in a second phase.

Additionally, the runnning time of our previous algorithm is only pseudopolynomial, as the algorithm needs $\min\{\sum_{i \in S} b_i^{max}, \sum_{i \in V \setminus S} b_i^{min}\}$ many iterations. We overcome this second problem by increasing the b values by as much possible in every iteration. To know this amount, it is necessary to precompute which of the two bounds is reached first, i.e., whether S or $V \setminus S$ is the limiting set. If S has more limiting bounds than $V \setminus S$, we set $b_i = b_i^{max}$ for all $i \in S$; otherwise, we set $b_i = b_i^{min}$ for all $i \in V \setminus S$. In both cases, it only remains to distribute the inbalance of b among the nodes in the non-limiting set. To do this, we iterate over all nodes i in the non-limiting set in arbitrary order and decrease or increase b_i as much as possible in the first and second case, respectively. See Algorithm 1 for the pseudo-code of this procedure. When the algorithm stops with a balanced vector b, we obtain again a solution b of value $\min\{\sum_{i \in S} b_i^{max}, -\sum_{i \in V \setminus S} b_i^{min}\}$.

Lemma 9. *Given a hose source set $\emptyset \subsetneq S \subsetneq V$, Algorithm 1 computes a scenario $b \in \mathfrak{H}(V, b^{min}, b^{max})$ with*

$$\sum_{i \in S} b_i = \min\{\sum_{i \in S} b_i^{max}, -\sum_{i \in V \setminus S} b_i^{min}\}$$

or it correctly decides that $\mathfrak{H}(V, b^{min}, b^{max})$ is empty.

Proof. Let $L = S$ or $L = V \setminus S$ be the set that limits how much supply we can install in S. We prove the correctness of the algorithm by showing that Lines 13–27 maintain two invariants: (1) At all times, b respects all bounds, i.e., $b_i \in [b_i^{min}, b_i^{max}]$ for all $i \in V$. (2) At all times, r stores the balance of our current b vector, i.e. $r = \sum_{i\in V} b_i$.

We establish Invariant 1 in lines 3–9 and 10/11 for $i \in L$ and $i \in F$, respectively. Line 12 establishes Invariant 2. Suppose now that $r < 0$ in line 13 (the other case works analogeously). We already know that both invariants hold before the first iteration of the loop in lines 14–19 and we assume by induction that the same is true before the j-th iteration, for some $j \geq 2$. Suppose that the j-th iteration considers $i \in F$. Then, b_i is at most increased to $b_i + b_i^{max} - b_i = b_i^{max}$, i.e. Invariant 1 is maintained. Also, r is changed by the same value as b_i and thus still stores the current balance of b. This means that Invariant 2 still holds. When the algorithm stops with $r = 0$, we have found a scenario $b \in \mathfrak{H}(V, b^{min}, b^{max})$. Also, by our choice in lines 3–9, we have $\sum_{i\in S} b_i = \sum_{i\in S} b_i^{max}$ if S is limiting and $\sum_{i\in S} b_i = -\sum_{i\in V\setminus S} = -\sum_{i\in V\setminus S} b_i^{min}$ otherwise. If the algorithm stops with $r < 0$, then $m = b_i^{max} - b_i$ in all iterations and thus, $b_i = b_i^{max}$ for all $i \in F$ where $b_i^{max} > 0$. From line 11, we know that $b_i = b_i^{max}$ for all $i \in F$ with $b_i^{max} < 0$ and our initialization guarantees $0 = b_i = b_i^{max}$ for all the $i \in F$ with $b_i^{max} = 0$. We conclude that $0 > r = \sum_{i\in F} b_i = \sum_{i\in F} b_i^{max}$. If $F = S$, we directly have a contradiction to S being a hose source set. If $F = V \setminus S$ instead, we also have $b_i = b_i^{max}$ for all $i \in L$. It follows that $\sum_{i\in V} b_i = \sum_{i\in V} b_i^{max} < 0$. Now, let $b' \in \mathfrak{H}(V, b^{min}, b^{max})$. Then, $\sum_{i\in V} b'_i \leq \sum_{i\in V} b_i^{max} < 0$ which is a contradiction to $\sum_{i\in V} b'_i = 0$. Consequently, $\mathfrak{H}(V, b^{min}, b^{max}) = \emptyset$. □

It remains to show that Algorithm 1 computes an optimum scenario for (MAX-B).

Theorem 10. *Let $S \subseteq V$ be a hose source set. Then*

$$B_S = \min\Big\{\sum_{i\in S} b_i^{max}, -\sum_{i\in V\setminus S} b_i^{min}\Big\}.$$

Proof. For $i = 1, \dots, |V|$, introduce dual variables υ_i, λ_i for the upper/lower bound constraints of b_i, repectively, and define a dual variable β for the balance constraint. This gives us the dual (MAX-B*) of (MAX-B)

$$\begin{aligned} \min \sum_{i\in V} b_i^{max}\upsilon_i - \sum_{i\in V} b_i^{min}\lambda_i& \\ \upsilon_i - \lambda_i + \beta \geq 1 &\quad \text{for all } i \in S \\ \upsilon_i - \lambda_i + \beta \geq 0 &\quad \text{for all } i \in V \setminus S \\ \upsilon_i, \lambda_i \geq 0 &\quad \text{for all } i \in V \end{aligned} \tag{MAX-B*}$$

If $\sum_{i\in S} b_i^{max} \leq -\sum_{i\in V\setminus S} b_i^{min}$, running Algorithm 1 gives us a scenario b with $\sum_{i\in S} b_i = \sum_{i\in S} b_i^{max}$. We choose $\upsilon_i = 1$ for all $i \in S$, $\upsilon_i = 0$ for all $i \in V \setminus S$, $\lambda_i = 0$ for all $i \in V$ and finally $\beta = 0$.

Our choice (υ,λ,β) is feasible for (H_S^*) and satisfies complementary slackness with b. Otherwise, we suppose $\sum_{i\in S} b_i^{max} > -\sum_{i\in V\setminus S} b_i^{min}$ and Algorithm 1 yields a scenario b with $\sum_{i\in S} = -\sum_{i\in V\setminus S} b_i^{min}$. Choosing $\lambda_i = 1$ for all $i \in V\setminus S$, $\lambda_i = 0$ for all $i \in S$, $\upsilon_i = 0$ for all $i \in V$ and $\beta = 1$ is a feasible solution for (H_S^*) and b satisfies complementary slackness with (υ,λ,β). □

Theorem 10 tells us that we can write (H-SEP) as the much easier problem

$$\min_{S\subseteq V} \sum_{e\in\delta(S)} u_e^* - \min\Big\{\sum_{i\in S} b_i^{max}, -\sum_{i\in V\setminus S} b_i^{min}\Big\}. \qquad \text{(H-SEP')}$$

In this formulation, we omitted the constraint that $\sum_{i\in S} b_i \geq 0$, because the inner part of the problem yields a negative value if it is violated. Thus, omitting the constraint does not produce more optimum solutions. We can now write (H-SEP') as a MIP that is a maximum cut problem with additional constraints:

$$\begin{aligned}
\min \sum_{\{i,j\}\in E} & u_{ij}^* y_{ij} - B \\
B &\leq \sum_{i\in V} x_i b_i^{max} \\
B &\leq -\sum_{i\in V} (1-x_i) b_i^{min} \\
x_i - x_j &\leq y_{ij} && \text{for all } \{i,j\}\in E \\
x_j - x_i &\leq y_{ij} && \text{for all } \{i,j\}\in E \\
x_i &\in \{0,1\} && \text{for all } i\in V \\
y_{ij} &\in \{0,1\} && \text{for all } \{i,j\}\in E
\end{aligned} \qquad \text{(IP-H-SEP)}$$

The MIP will not give us an actual worst-case scenario; however, we can easily call Algorithm 1 on the set $S := \{i \in V \mid x_i = 1\}$ to obtain one. If we want more than one worst-case scenario, we can even call it several times while permuting the order in which it considers the nodes.

In contrast to the finite case, separating cut-set inequalities in the polyhedral case is NP-hard, as we show in the following theorem by a reduction from *minimum expansion*. Chekuri, Oriolo, Scutellà and Shepherd [18] show the same result for the multi-commodity case. In fact, they also use a (more complicated) reduction from minimum expansion.

Theorem 11. *Given an instance $(G,\mathfrak{H}(V,b^{min},b^{max}))$ of the (sRND-Hose) problem and a fractional capacity vector $u \in \mathbb{R}^E$, the problem of finding a cut-set inequality that is violated by u is NP-hard. In particular, the feasibility test for u is co-NP-complete.*

Proof. Minimum expansion is defined in the following way: Given an undirected graph $G = (V,E)$ and edge capacities u_e for all edges $e \in E$, find a set $\emptyset \subsetneq S \subsetneq V$ with $|S| \leq |V|/2$ that minimizes the expansion $\sum_{e\in\delta(S)} u_e/|S|$ of G. Minimum expansion is NP-hard [30, Section 3.2].

If we have an input (V, E, u) for minimum expansion, we can define an instance for the cut-set separation problem on the same graph $G = (V, E)$. Set $\hat{b}_i^{max} := 1$ and $\hat{b}_i^{min} = -1$ for all $i \in V$. We claim that G has an expansion of strictly less than 1 if and only if there is a violated cut-set inequality with respect to u and $\mathfrak{H}(V, \hat{b}^{min}, \hat{b}^{max})$.

By our definition of $\hat{b}^{max}$ and $\hat{b}^{min}$, we obtain from Theorem 10 that $B_S = |S|$ for any $S \subseteq V$ with $|S| \leq |V|/2$. Thus, there is a violated cut-set inequality in G iff for some S, $\sum_{e \in \delta(S)} u_e < B_S = |S|$. This is equivalent to $\sum_{e \in \delta(S)} u_e / |S| < 1$. $\square$

5 Computational results

In this section we describe the outcome of our extensive computational campaign conducted to assert the effectiveness of the cut-set formulation within a classical branch-and-cut framework for both the finite and Hose cases. The branch-and-cut algorithm is implemented in C++ within the ABACUS 3.2U2 framework [28] and run on an Intel XEON 5410 2.3 GHz with 3 GB RAM, and Cplex 12.1 is used as an LP solver inside our branch and cut. The main ingredients to enhance the basic scheme are described in the following, while Section 5.1 and Section 5.2 report the results on the finite and Hose cases, respectively.

Preprocessing. We partition the graph into its biconnected components as suggested in [16]. It is straight-forward to generalize the approach to the polytopal case.

Cutting Plane Separation. We use the cut-set separation both for the finite and the Hose case, as described in Sections 3 and 4, respectively. In the Hose case, after the exact separation from Section 4 is invoked, we repeatedly call Algorithm 1 to obtain a list of 10 non-routable scenarios. Then, the polynomial separation of Section 3 is called until the 10 scenarios can be routed. The separation algorithm `EnumZH` is called at the root node only, as well as the $\{0, \frac{1}{2}\}$-cut separation code by Andreello, Caprara and Fischetti [5] which is called at the root node when the other algorithms fail.

Primal Heuristics. We execute several fast primal heuristics at each branch-and-cut node when no more cut-set inequality is violated. They are all based on *rounding* operations. More precisely,

- The *Simple rounding* rounds up all fractional values to produce a feasible solution.
- The *Cycle rounding* looks for a cycle C with only fractional edges by a depth first search. On C the heuristic rounds down the edge with the smallest fractional value, say, p and increases the capacity of all other edges on C by p. When no more cycles are found, all remaining fractional capacities are rounded up.

- The *Shortest Path rounding* works in the same way as the Cycle rounding, but it obtains the cycle C by removing an edge with a smallest fractional value and connecting its end-nodes by a shortest path of fractional edges.

In the finite case, we also use the *rounding heuristic* described in [16]. Finally, we use the large neighborhood search heuristic introduced in [3, 4], but only at the root node, in the finite case and with a time limit of 120 CPU seconds.

Settings. Very few special settings have been used within the branch-and-cut framework provided by ABACUS. Namely, we used *aggressive strong branching*, the branch-and-cut tree is traversed in *best-first order*, and we removed *non-binding* cutting planes after 10 iterations.

5.1 Experiments with Finite Uncertainty Sets

Testbed. We consider four different classes of instances for our experiments. (These instances, as well as those for the Hose case (Section 5.2), are available upon request from the authors.) Each instance consists of a *network topology* and a *scenario set*.

- `BLS`: The instances have been used in [16] and are based on realistic network topologies introduced in [2].
- `JMP`: The instances are generated according to the method in [27] with zero-one balances as proposed in [4].
- `SNDLib`: The SNDLib [38] is an established standard benchmark set for real-world network topologies. We augmented the real-world topologies with random balances to adapt the instances to our specific problems.
- `PA`: The *preferential attachment* model [8] defines a standard way to create realistic networks: For some parameter $a \geq 2$, one starts with a complete graph on a nodes and iteratively adds more nodes and edges to the network. When a new node v is inserted, it connects to exactly a existing nodes. In this way, the parameter a controls the density of the graph. The probability that v connects to an existing node w is proportional to the degree of w. Again, we augmented the resulting network topologies with random balances.

Comparison with the flow formulation. Our experiments for the finite case compare the cut-set formulation (IP-CS) with the flow formulation introduced in [16]. For the `BLS` instances the comparison is performed with the algorithm in [16] that solved the flow formulation by enhancing it through *target cuts* (see, Section 1). We have access to the original computational data by [16] and conducted the experiments on the same machines, making direct comparison possible. For the other set of instances, instead, the flow formulation has been solved as a black-box MIP through Cplex 12.1 by using default settings and in single-thread mode. This is to provide a fair comparison with the sequential ABACUS implementation. Finally, the time limit for each instance was set to 4 hours of CPU time.

Description of the tables. In Tables 1–4 we show instances that could be solved to optimality by both of the compared methods and averages over sub-classes of instances for each table entry. Computing times are expressed in CPU seconds. We first show the instance size and the percentage gap between the optimum fractional and integer solution values. Recall that the flow formulation and the cut-set one are proven to be equivalent in terms of LP relaxation bound. For each method we show the number of instances that could be solved to optimality within 14,400 CPU seconds (4 hours) and in brackets number of instances that stopped due to memory limit of 3 GB. Then, we report the average CPU time over all instances that could be solved to optimality by both methods and the corresponding number of branch-and-bound nodes. The root gap reported is the average percentage gap of the dual root bound (after all cuts were added) with respect to the optimum integer solution value. Finally, we report the time that is needed to solve the LP-relaxation. For the cut-set formulation only, we also report the overall separation time and the overall heuristic time. For the PA instances (Table 4) the results for each size are average over $a \in \{2,3,4,5,6,7\}$.

Results. Table 1 shows that our branch-and-cut algorithm based on the cut-set formulation is superior to the branch-and-cut algorithm (also ABACUS-based) in [16] both in terms of the number of solved instances and the CPU time. In particular, these instances turn out to be rather easy for our algorithm that only has some issues due to memory limits. Specifically, the memory limit prevents us from finding the optimum solution of 10 out of 1,156 instances.

Instances JMP (Table 2) turn out to be much more challenging and the comparison with the flow formulation solved by Cplex is interesting. Until $|V| = 35$ both methods can solve all instances (in roughly the same computing time) and we can observe that the cut-set formulation amended by $\{0, \frac{1}{2}\}$-cuts gives a better bound than the flow formulation with Cplex cuts. On larger instances $|V| \geq 40$ both algorithms start to suffer and the algorithm based on the cut-set formulation frequently reaches the memory limit. Instead, when Cplex is unable to solve the problem it is because of the time limit (14,400 CPU seconds), which is a clear indication that the formulation became too big. As the bound at the root node is better for our algorithm, this behavior seems to indicate that the memory limit reached by our algorithm is likely a software limit (essentially due to the less sophisticated implementation of ABACUS with respect to Cplex) and not a problem of the formulation, whose LP size is always kept under control through cut purging.

The above analysis is confirmed by the results on Tables 3 and 4 for the classes SNDlib and PA, respectively. Especially on the PA instances one can start too appreciate that, for large values of $|V|$ and many scenarios, the cut-set formulation becomes more effective while the flow formulation is too large. That can be expected as the separation limits the size of the cut-set LP to the needed cuts. For Tables 3 and 4 the numbers in brackets for the “#solved(m)” column of the flow formulation refer to the number of times the *memory* limit is reached. So, it is worth mentioning that for PA instances, cplex reaches both the time and the memory limit (the number of solved instances plus those not solved due to the memory limit is sometimes smaller than 180), showing that the size of the flow formulation gives rise to all sort of issues. The two numbers instead almost always sum to 180 in the cut-set formulation case, thus confirming that the management of the tree of ABACUS is likely to be the issue. Nevertheless, for large instances with many scenarios our algorithm can solve many more instances in significantly shorter computing times.

5.2 Experiments with the Hose Uncertainty Set

In order to obtain a flow formulation in the Hose uncertainty case, we would have to convert the linear description of the Hose polytope $\mathfrak{H}(V, b^{min}, b^{max})$ into a list of its vertices. This can be done with a software like PORTA [20]. Table 5 shows that this approach is not practical: Already for small instances, we cannot rely on being able to convert the description within 1800 seconds and, additionally, the list of vertices can easily become too large to be useful. Therefore, we cannot present a comparison with the flow formulation in the Hose case.

Testbed. To limit the space needed to present our results, we only report the results on the most general instances, i.e., the `SNDLib` and `PA` topologies. The Hose uncertainty sets have been generated according to three different distributions:

- `geometric`: The width of the demand intervals is chosen with a geometric distribution, i.e., there are many nodes with narrow demand intervals and few nodes with broad intervals. The center of the intervals is chosen uniformly at random.
- `uniform`: Both the width and the center of the intervals are chosen uniformly at random for each node.
- `zero-one`: All intervals are $[-1, 1]$.

Description of the tables. In Tables 6 and 7, we report the CPU time and number of solved instances for the random Hose instances, grouped by network topology in the `SNDLib` case and according to the density parameter a in the `PA` case, respectively. We also show the number of times that the separation MIP needs to be solved on average over all separation calls. Again, we use a time limit of 4 hours.

Results. The results in Table 6 show that the branch-and-cut algorithm based on the flow formulation is effective in the Hose uncertainty case. More precisely, very few of the `SNDlib` instances cannot be solved to optimality and both the computing times and the number of branch-and-bound nodes are small on average. The same holds for the `PA` instances (Table 7) where the difficulty grows with the value of a. In terms of the difference of the random distribution, the behavior on `geometric` and `uniform` instances is quite similar, while the `zero-one` case turns out to be rather easy, except for 9 instances with $|V| = 100$ and $a = 6$ and one instance with $|V| = 100$ and $a = 7$.

In Table 8 we report a disaggregated picture of our cut-set based algorithm. We consider the PA instances for the three distributions and $a = 6$. In addition to the previous information, we show the time ("ip-sep-time") needed to solve the exact separation MIP (see Section 4), and the corresponding number of calls, "ip sepcalls (in %)". The results for other values of parameter a are comparable. The disaggregated results in Table 8 allow us to assert that the quality of both the LP and root lower bounds is very high. However, the difficulty of the instances with respect to the finite uncertainty case seems to be associated with closing the small gap within the time limit. Indeed, all unsolved PA instances reach the time limit (not a memory limit). This is due to the size of the resulting problems: The LPs start to be time consuming as well as the separation time, especially due to exact separation MIP.

Algorithm 1: Computing a worst-case scenario for a fixed S.

input : Vectors b^{min}, b^{max}, a hose source set $S \subseteq V$
output: A worst-case scenario b for S.

1 let $F := \emptyset$
2 let $b \equiv 0$
3 **if** $\sum_{i \in S} b^{max} \leq -\sum_{i \in V \setminus S} b^{min}$ **then**
4   set $F := V \setminus S$
5   **for** $i \in S$ **do** set $b_i := b_i^{max}$
6 **else**
7   set $F := S$
8   **for** $i \in V \setminus S$ **do** set $b_i := b_i^{min}$
9 **end**

Determine which of S and $V \setminus S$ is limiting according to our earlier definition. Store the non-limiting set in F. All nodes in the limiting set $V \setminus F$ can be set to one of their bounds.

10 **for** $i \in F$ **with** $b_i^{min} > 0$ **do** set $b_i := b_i^{min}$
11 **for** $i \in F$ **with** $b_i^{max} < 0$ **do** set $b_i := b_i^{max}$

Define b for all nodes $i \in F$. Choose the value from $[b_i^{min}, b_i^{max}]$ that is closest possible to zero

12 let $r := \sum_{i \in V} b_i$
13 **if** $r < 0$ **then**
14   **for** $i \in F$ **with** $b_i^{max} > 0$ **do**
15   let $m := \min\{b_i^{max} - b_i, -r\}$
16   set $b_i := b_i + m$
17   set $r := r + m$
18   **if** $r == 0$ **then return** b
19   **end**

Distribute the imbalance r among the nodes in F. If the imbalance is negative, we only consider nodes that can take positive b values. All other nodes cannot reduce the imbalance (due to our choice in lines 10/11).

20 **else if** $r > 0$ **then**
21   **for** $i \in F$ **with** $b_i^{min} < 0$ **do**
22   let $m := \max\{b_i^{min} - b_i, -r\}$
23   set $b_i := b_i + m$
24   set $r := r + m$
25   **if** $r == 0$ **then return** b
26   **end**
27 **end**
28 **return** "$\mathfrak{H}(V, b^{min}, b^{max})$ is empty."

Distribute the imbalance r among the nodes in F. If the imbalance is positive, we only consider nodes that can take negative b values. All other nodes cannot reduce the imbalance (due to our choice in lines 10/11).

				Cut-Set formulation (CS)							BLS [16]	
$\lvert V\rvert$	$\lvert\mathfrak{D}\rvert$	#inst	lp-gap	#solved (m)	cputime	#nodes	root-gap	relax-time (m)	sep-time	heur-time	#solved (m)	cputime
$0 \le \lvert V\rvert \le 149$	2	153	0.02%	153 (0)	2	111	0.00%	0 (0)	0	0	153	0.7
$0 \le \lvert V\rvert \le 149$	3	153	0.03%	152 (1)	7	265	0.00%	0 (0)	0	0	152	1.1
$0 \le \lvert V\rvert \le 149$	4	153	0.03%	151 (2)	2	105	0.00%	0 (0)	0	0	150	4.8
$0 \le \lvert V\rvert \le 149$	5	185	0.02%	182 (3)	0	127	0.00%	0 (0)	0	0	183	5.9
$150 \le \lvert V\rvert \le 299$	2	68	0.00%	67 (1)	2	78	0.00%	0 (0)	1	0	66	85.6
$150 \le \lvert V\rvert \le 299$	3	68	0.01%	68 (0)	45	205	0.00%	0 (0)	8	0	61	4.9
$150 \le \lvert V\rvert \le 299$	4	68	0.00%	66 (2)	2	95	0.00%	0 (0)	1	0	63	27.3
$150 \le \lvert V\rvert \le 299$	5	68	0.00%	67 (1)	82	186	0.00%	0 (0)	11	0	62	141.2
$300 \le \lvert V\rvert \le 499$	2	60	0.00%	60 (0)	0	197	0.00%	0 (0)	0	0	60	81.3
$300 \le \lvert V\rvert \le 499$	3	60	0.00%	60 (0)	0	169	0.00%	0 (0)	0	0	60	103.4
$300 \le \lvert V\rvert \le 499$	4	60	0.00%	60 (0)	0	221	0.00%	0 (0)	0	0	59	129.8
$300 \le \lvert V\rvert \le 499$	5	60	0.00%	60 (0)	0	547	0.00%	0 (0)	0	0	55	166.8

Table 1: Comparison to [16] on the `BLS` class. We use the same grouping and the same machines as the original authors.

				Cut-Set formulation (CS)							Flow formulation (CPLEX)				
$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathfrak{D}\rvert$	lp-gap in %	#solved (m)	cputime	#nodes	root-gap in %	relax-time (t)	sep-time	heur-time	#solved (m)	cputime	#nodes	root-gap in %	relax-time (t)
25	104	5	13.3	3 (0)	1	46	2.9	0 (0)	0	0	3 (0)	0	410	7.7	0 (0)
25	104	10	17.1	3 (0)	24	2016	7.1	0 (0)	3	0	3 (0)	26	2701	12.2	0 (0)
30	121	5	10.6	3 (0)	7	436	2.5	0 (0)	1	0	3 (0)	5	1175	5.6	0 (0)
30	121	10	14.3	3 (0)	125	6875	6.6	0 (0)	15	1	3 (0)	123	12661	9.5	0 (0)
35	155	5	12.3	3 (0)	75	6157	5.3	0 (0)	7	0	3 (0)	9	1808	7.1	0 (0)
35	155	10	12.3	3 (0)	1196	47858	6.2	0 (0)	115	20	3 (0)	597	31355	9.2	0 (0)
40	182	5	17.2	2 (1)	51	1886	6.8	0 (0)	8	0	3 (0)	6	1121	12.0	0 (0)
40	182	10	—	0 (3)	—	—	—	0 (0)	—	—	3 (0)	—	—	—	0 (0)
45	223	5	16.1	1 (2)	15	243	5.6	0 (0)	6	0	3 (0)	10	1106	8.4	0 (0)
45	223	10	—	0 (3)	—	—	—	0 (0)	—	—	1 (0)	—	—	—	0 (0)
50	254	5	—	0 (3)	—	—	—	0 (0)	—	—	2 (0)	—	—	—	0 (0)
50	274	10	—	0 (3)	—	—	—	0 (0)	—	—	0 (0)	—	—	—	0 (0)

Table 2: Computational results for the instances of the `JMP` class. We consider 3 instances for each pair $(|V|, |\mathfrak{D}|)$.

					Cut-Set formulation (CS)							Flow-formulation (CPLEX)				
	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathfrak{D}\rvert$	lp-gap in %	#solved (m)	cputime	#nodes	root-gap in %	relax-time (t)	sep-time	heur-time	#solved (m)	cputime	#nodes	root-gap in %	relax-time (t)
pdh	11	34	5	20.6	30 (0)	1	495	10.5	0 (0)	0	0	30 (0)	0	206	17.6	0 (0)
	11	34	10	29.4	30 (0)	45	13173	21.9	0 (0)	3	0	30 (0)	9	7150	28.6	0 (0)
	11	34	15	27.9	30 (0)	1518	65333	20.8	0 (0)	22	5	30 (0)	59	29047	27.0	0 (0)
	11	34	20	29.2	30 (0)	1486	74290	22.5	0 (0)	35	8	30 (0)	166	37476	28.6	0 (0)
	11	34	30	23.4	20 (10)	113	11587	19.3	0 (0)	13	2	30 (0)	112	5485	22.8	0 (0)
	11	34	40	22.1	20 (10)	63	6050	18.1	0 (0)	9	1	30 (0)	109	2963	21.0	0 (0)
	11	34	50	22.0	20 (10)	60	4531	18.1	0 (0)	10	1	30 (0)	142	2569	21.1	0 (0)
	11	34	75	19.8	20 (10)	41	2902	15.8	0 (0)	10	1	20 (0)	500	5708	18.8	0 (0)
	11	34	100	18.5	20 (10)	75	5699	14.7	0 (0)	22	3	20 (0)	1441	11260	18.5	0 (0)
newyork	16	49	5	15.7	30 (0)	11	2001	10.1	0 (0)	0	0	30 (0)	3	2460	14.1	0 (0)
	16	49	10	12.2	20 (10)	102	8510	9.2	0 (0)	7	0	30 (0)	57	6658	9.8	0 (0)
	16	49	15	12.2	20 (10)	568	38671	9.3	0 (0)	41	6	20 (0)	200	13906	10.7	0 (0)
	16	49	20	11.4	20 (10)	281	19674	9.0	0 (0)	29	3	20 (0)	226	8238	10.5	0 (0)
	16	49	30	12.5	20 (10)	84	5675	9.6	0 (0)	14	1	20 (0)	282	4747	12.0	0 (0)
	16	49	40	9.7	20 (10)	251	15611	7.1	0 (0)	45	5	20 (0)	1131	13451	9.1	0 (0)
	16	49	50	13.8	20 (10)	1611	58553	11.5	0 (0)	197	31	20 (0)	3730	28587	12.8	0 (0)
	16	49	75	12.0	20 (10)	628	29130	9.2	0 (0)	152	22	20 (0)	2851	10278	11.3	1 (0)
	16	49	100	1.6	20 (10)	1	20	1.6	0 (0)	0	0	10 (0)	5	0	1.0	1 (0)
ta1	24	55	5	9.2	30 (0)	0	220	3.9	0 (0)	0	0	30 (0)	0	188	5.7	0 (0)
	24	55	10	11.0	30 (0)	10	1450	6.3	0 (0)	1	0	30 (0)	2	524	8.7	0 (0)
	24	55	15	5.6	20 (10)	4	522	3.2	0 (0)	0	0	30 (0)	3	626	5.6	0 (0)
	24	55	20	5.6	20 (10)	4	541	3.2	0 (0)	1	0	30 (0)	6	343	5.1	0 (0)
	24	55	30	11.8	30 (0)	201	14939	6.9	0 (0)	34	4	30 (0)	67	2673	8.6	0 (0)
	24	55	40	11.5	30 (0)	85	7119	6.6	0 (0)	22	2	30 (0)	88	2254	8.5	0 (0)
	24	55	50	5.4	20 (10)	3	290	3.1	0 (0)	1	0	30 (0)	15	209	3.7	0 (0)
	24	55	75	9.6	30 (0)	57	3925	5.7	0 (0)	22	3	30 (0)	239	1743	7.9	0 (0)
	24	55	100	8.8	30 (0)	46	3094	5.5	1 (0)	22	4	30 (0)	169	565	7.7	1 (0)
france	25	45	5	16.1	30 (0)	7	2680	9.6	0 (0)	0	0	30 (0)	0	905	12.3	0 (0)
	25	45	10	12.3	30 (0)	25	6762	7.0	0 (0)	3	0	30 (0)	3	1234	9.2	0 (0)
	25	45	15	11.1	30 (0)	23	5171	6.4	0 (0)	4	0	30 (0)	10	1960	8.4	0 (0)
	25	45	20	12.0	30 (0)	134	17488	7.4	0 (0)	16	2	30 (0)	24	3682	9.7	0 (0)
	25	45	30	10.5	30 (0)	174	21039	6.7	0 (0)	29	4	30 (0)	93	4917	7.7	0 (0)
	25	45	40	9.7	30 (0)	23	3817	5.9	0 (0)	7	0	30 (0)	31	1220	7.9	0 (0)
	25	45	50	8.4	30 (0)	7	1111	4.8	0 (0)	3	0	30 (0)	18	313	6.3	0 (0)
	25	45	75	8.0	30 (0)	3	432	4.5	0 (0)	1	0	30 (0)	23	218	5.5	0 (0)
	25	45	100	9.3	30 (0)	32	3883	5.6	0 (0)	17	2	30 (0)	377	3949	7.4	0 (0)
norway	27	51	5	10.2	30 (0)	3	454	6.2	0 (0)	0	0	30 (0)	0	292	8.4	0 (0)
	27	51	10	14.1	30 (0)	105	11186	8.7	0 (0)	14	1	30 (0)	14	2699	11.2	0 (0)
	27	51	15	12.4	30 (0)	294	22944	7.2	0 (0)	39	4	30 (0)	48	5065	9.6	0 (0)
	27	51	20	11.1	30 (0)	64	5722	7.4	0 (0)	14	1	30 (0)	45	2668	9.8	0 (0)
	27	51	30	10.4	30 (0)	429	26232	6.3	0 (0)	79	10	30 (0)	394	11838	8.5	0 (0)
	27	51	40	9.6	30 (0)	1625	50713	5.7	0 (0)	188	24	30 (0)	1792	47327	2.4	0 (0)
	27	51	50	9.7	30 (0)	473	24332	6.5	0 (0)	112	15	30 (0)	919	12656	8.5	0 (0)
	27	51	75	9.1	30 (0)	359	18175	5.9	2 (0)	125	17	30 (0)	1658	10226	8.5	1 (0)
	27	51	100	3.8	20 (10)	12	118	1.8	2 (0)	2	9	20 (0)	25	54	3.1	2 (0)
cost266	37	57	5	9.7	30 (0)	18	2245	5.3	0 (0)	2	0	30 (0)	0	530	6.5	0 (0)
	37	57	10	10.3	30 (0)	246	15950	6.0	0 (0)	36	2	30 (0)	18	2704	7.0	0 (0)
	37	57	15	9.8	30 (0)	712	32368	5.7	0 (0)	105	7	30 (0)	25	1434	7.2	0 (0)
	37	57	20	8.9	30 (0)	169	10715	5.1	0 (0)	44	3	30 (0)	29	1094	6.8	0 (0)
	37	57	30	8.5	30 (0)	577	26413	5.5	1 (0)	145	12	30 (0)	135	2523	7.2	0 (0)
	37	57	40	6.7	30 (0)	73	3867	3.6	2 (0)	30	3	30 (0)	197	2417	5.6	0 (0)
	37	57	50	7.4	30 (0)	36	1442	3.9	3 (0)	14	4	30 (0)	68	429	5.9	0 (0)
	37	57	75	6.0	30 (0)	60	2455	4.1	81 (0)	32	9	30 (0)	203	654	5.5	1 (0)
	37	57	100	6.3	30 (0)	105	3948	4.4	15 (0)	62	14	30 (0)	414	1053	5.3	3 (0)
germany50	50	88	5	2.3	10 (20)	1	48	2.3	0 (0)	0	0	30 (0)	0	14	1.8	0 (0)
	50	88	10	1.3	10 (20)	3	90	1.3	0 (0)	0	0	10 (0)	1	19	1.0	0 (0)
	50	88	15	1.4	10 (20)	4	66	1.4	0 (0)	0	0	20 (0)	3	30	1.1	0 (0)
	50	88	20	0.0	10 (20)	3	0	0.0	1 (0)	0	2	20 (0)	0	0	0.0	0 (0)
	50	88	30	3.1	20 (10)	327	5573	2.4	1 (0)	75	7	20 (0)	160	693	2.7	0 (0)
	50	88	40	0.7	10 (20)	11	62	0.7	3 (0)	1	8	20 (0)	11	11	0.7	1 (0)
	50	88	50	3.2	20 (10)	441	6811	2.4	6 (0)	133	15	20 (0)	971	2124	2.9	2 (0)
	50	88	75	2.2	20 (10)	400	5707	1.7	10 (0)	153	29	20 (0)	4685	4734	2.1	7 (0)
	50	88	100	0.7	20 (10)	35	40	0.7	13 (0)	2	31	10 (0)	63	19	0.6	11 (0)
ta2	65	108	5	5.7	20 (10)	292	10274	2.9	0 (0)	37	2	30 (0)	5	868	3.9	0 (0)
	65	108	10	2.4	10 (20)	39	1276	1.7	0 (0)	7	0	30 (0)	4	163	1.6	0 (0)
	65	108	15	2.3	10 (20)	165	5202	2.2	0 (0)	33	2	20 (0)	37	693	2.1	0 (0)
	65	108	20	1.8	10 (20)	89	2758	1.7	1 (0)	23	1	30 (0)	39	471	1.6	0 (0)
	65	108	30	4.1	20 (10)	738	15962	2.6	3 (0)	215	19	30 (0)	195	958	3.3	1 (0)
	65	108	40	4.0	20 (10)	1303	31502	2.4	6 (0)	487	46	30 (0)	376	1396	3.1	3 (0)
	65	108	50	3.6	20 (10)	421	9131	2.0	10 (0)	187	20	30 (0)	350	817	2.6	3 (0)
	65	108	75	3.3	20 (10)	709	11252	2.2	19 (0)	358	67	30 (0)	630	809	2.4	8 (0)
	65	108	100	3.1	20 (10)	352	4776	1.9	57 (0)	181	50	20 (0)	872	562	2.3	23 (0)

Table 3: Computational results for the instances of the SNDlib class. We consider 30 instances for each network topology and for each number of scenarios $|\mathfrak{D}| \in \{5, 10, 15, 20, 30, 40, 50, 75, 100\}$.

				Cut-Set formulation (CS)							Flow formulation (CPLEX)				
$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathfrak{D}\rvert$	lp-gap in %	#solved (m)	cputime	#nodes	root-gap in %	relax-time (t)	sep-time	heur-time	#solved (m)	cputime	#nodes	root-gap in %	relax-time (t)
20	76	5	8.1	180 (0)	0	21	2.5	0 (0)	0	0	180 (0)	0	43	4.6	0 (0)
20	76	10	8.6	180 (0)	0	51	3.2	0 (0)	0	0	180 (0)	0	101	5.5	0 (0)
20	76	30	7.8	180 (0)	0	45	3.0	0 (0)	0	0	180 (0)	3	102	5.0	0 (0)
20	76	50	7.3	180 (0)	0	42	2.7	0 (0)	0	0	180 (0)	7	88	4.8	0 (0)
20	76	75	6.8	180 (0)	0	31	2.4	0 (0)	0	0	180 (0)	15	80	4.4	0 (0)
20	76	100	6.3	180 (0)	0	25	2.0	0 (0)	0	0	180 (0)	19	63	4.0	1 (0)
25	98	5	9.3	180 (0)	1	163	3.7	0 (0)	0	0	180 (0)	0	258	6.0	0 (0)
25	98	10	9.7	180 (0)	5	489	4.4	0 (0)	1	0	180 (0)	7	651	6.8	0 (0)
25	98	30	8.4	180 (0)	6	366	3.9	0 (0)	2	0	180 (0)	57	684	6.2	0 (0)
25	98	50	8.0	180 (0)	7	323	3.7	0 (0)	3	0	180 (0)	140	681	6.0	1 (0)
25	98	75	7.7	180 (0)	9	312	3.6	0 (0)	4	0	180 (0)	306	693	5.8	2 (0)
25	98	100	7.4	180 (0)	10	319	3.4	0 (0)	6	0	180 (0)	497	689	5.6	4 (0)
30	121	5	6.3	180 (0)	3	213	2.5	0 (0)	0	0	180 (0)	1	212	4.0	0 (0)
30	121	10	6.6	180 (0)	9	529	2.8	0 (0)	2	0	180 (0)	15	469	4.5	0 (0)
30	121	30	5.9	180 (0)	8	322	2.6	0 (0)	3	0	180 (0)	122	469	4.1	0 (0)
30	121	50	5.4	180 (0)	12	365	2.4	0 (0)	5	0	180 (0)	399	541	3.9	2 (0)
30	121	75	5.2	180 (0)	10	233	2.4	0 (0)	5	0	178 (0)	544	397	3.8	5 (0)
30	121	100	4.9	180 (0)	9	178	2.3	0 (0)	5	0	172 (0)	792	368	3.6	8 (0)
35	143	5	7.7	180 (0)	8	399	3.2	0 (0)	1	0	180 (0)	2	326	4.9	0 (0)
35	143	10	8.2	180 (0)	36	1610	3.9	0 (0)	6	0	180 (0)	39	967	5.7	0 (0)
35	143	30	7.6	180 (0)	79	2609	3.8	0 (0)	21	2	180 (0)	582	1950	5.7	1 (0)
35	143	50	7.1	180 (0)	57	1539	3.6	0 (0)	21	2	180 (0)	1327	1447	5.3	4 (0)
35	143	75	6.6	180 (0)	45	998	3.3	0 (0)	21	2	174 (0)	2540	1216	5.0	10 (0)
35	143	100	5.6	180 (0)	30	488	2.8	0 (0)	16	2	158 (0)	2906	729	4.2	14 (0)
40	166	5	6.6	180 (0)	6	259	2.4	0 (0)	1	0	180 (0)	2	283	3.8	0 (0)
40	166	10	6.7	180 (0)	15	568	2.8	0 (0)	3	0	180 (0)	24	510	4.2	0 (0)
40	166	30	6.0	180 (0)	28	772	2.5	0 (0)	9	0	180 (0)	338	788	4.0	2 (0)
40	166	50	5.5	180 (0)	26	582	2.4	0 (0)	11	1	179 (0)	944	742	3.8	6 (0)
40	166	75	4.9	180 (0)	18	357	2.1	0 (0)	10	1	169 (0)	1496	550	3.4	13 (0)
40	166	100	4.4	180 (0)	11	187	1.8	0 (0)	7	0	132 (30)	1405	387	3.1	21 (0)
45	188	5	5.8	180 (0)	6	226	2.1	0 (0)	2	0	180 (0)	1	235	3.2	0 (0)
45	188	10	6.2	180 (0)	15	576	2.6	0 (0)	5	0	180 (0)	27	518	4.0	0 (0)
45	188	30	5.2	180 (0)	21	553	2.3	0 (0)	9	0	180 (0)	318	645	3.6	2 (0)
45	188	50	4.8	180 (0)	23	477	2.2	0 (0)	13	0	180 (0)	999	629	3.4	9 (0)
45	188	75	4.3	180 (0)	20	322	2.0	0 (0)	13	1	174 (0)	1985	572	3.2	21 (0)
45	188	100	4.2	180 (0)	21	374	1.9	0 (0)	13	1	89 (90)	1883	645	3.0	26 (30)
50	211	5	6.9	173 (7)	113	3042	2.9	0 (0)	14	0	180 (0)	17	909	4.0	0 (0)
50	211	10	6.2	152 (28)	196	4766	2.8	0 (0)	29	2	180 (0)	102	1372	4.0	0 (0)
50	211	30	5.2	146 (34)	192	3900	2.4	0 (0)	52	5	146 (0)	913	1743	3.5	5 (0)
50	211	50	4.7	143 (37)	167	3083	2.2	0 (0)	61	6	133 (0)	1664	1721	3.2	18 (0)
50	211	75	4.4	153 (27)	92	1672	2.0	0 (0)	46	5	117 (29)	2101	1064	3.0	38 (0)
50	211	100	4.7	152 (28)	58	1099	2.5	0 (0)	33	3	45 (120)	1356	888	3.1	23 (60)
60	256	5	6.3	169 (11)	165	3272	2.7	0 (0)	20	1	180 (0)	20	842	3.7	0 (0)
60	256	10	5.3	133 (47)	269	5525	2.4	0 (0)	43	4	177 (0)	125	1407	3.3	0 (0)
60	256	30	4.5	135 (45)	279	4841	2.2	0 (0)	79	9	142 (0)	1290	1884	3.0	12 (0)
60	256	50	3.9	134 (46)	154	2409	1.9	0 (0)	61	6	119 (0)	2523	1458	2.6	40 (0)
60	256	75	3.8	138 (42)	103	1623	1.8	0 (0)	52	5	68 (90)	2112	1188	2.6	56 (30)
60	256	100	4.0	141 (39)	24	477	1.5	0 (0)	17	1	30 (150)	382	509	2.3	169 (60)
70	301	5	4.5	135 (45)	181	2909	1.9	0 (0)	23	1	180 (0)	21	694	2.5	0 (0)
70	301	10	3.5	109 (71)	228	3921	1.5	0 (0)	40	3	160 (0)	92	919	2.2	1 (0)
70	301	30	3.0	107 (73)	212	2979	1.3	0 (0)	61	5	120 (0)	821	1042	1.9	18 (0)
70	301	50	2.6	110 (70)	219	2399	1.3	0 (0)	77	7	111 (0)	2278	867	1.8	80 (0)
70	301	75	3.3	110 (70)	279	3899	1.4	0 (0)	138	13	44 (120)	1431	1111	2.2	121 (60)
70	301	100	—	107 (72)	—	—	—	0 (0)	—	—	0 (180)	—	—	—	129 (120)
80	346	5	3.3	107 (73)	169	2260	1.2	0 (0)	21	1	180 (0)	10	405	1.8	0 (0)
80	346	10	2.6	91 (89)	148	2649	1.1	0 (0)	28	1	145 (0)	48	440	1.5	2 (0)
80	346	30	2.5	93 (87)	368	3951	1.3	0 (0)	95	7	100 (0)	1215	765	1.6	39 (0)
80	346	50	3.2	93 (85)	366	3652	1.5	0 (0)	131	12	66 (60)	2402	781	2.0	131 (0)
80	346	75	3.6	89 (87)	102	1527	1.3	0 (0)	64	4	30 (150)	512	634	2.0	304 (60)
80	346	100	—	88 (90)	—	—	—	0 (0)	—	—	0 (180)	—	—	—	29 (150)
90	391	5	2.7	88 (92)	330	4393	1.1	0 (0)	41	2	177 (0)	11	381	1.4	0 (0)
90	391	10	1.2	68 (111)	136	2142	0.6	0 (0)	30	1	121 (0)	31	223	0.7	3 (0)
90	391	30	1.0	67 (111)	261	1603	0.7	0 (0)	55	3	85 (0)	2070	640	0.8	66 (0)
90	391	50	1.6	71 (109)	634	7437	0.9	0 (0)	262	21	49 (90)	1544	1814	1.1	141 (30)
90	391	75	2.1	72 (108)	1285	11281	1.2	0 (0)	586	47	23 (150)	1767	1426	1.4	220 (120)
90	391	100	—	72 (107)	—	—	—	—	—	—	0 (180)	—	—	—	—
100	436	5	2.0	81 (99)	186	2042	0.7	0 (0)	25	0	170 (0)	6	230	1.0	0 (0)
100	436	10	1.3	71 (109)	264	3055	0.7	0 (0)	49	2	103 (0)	104	382	0.8	4 (0)
100	436	30	1.2	67 (108)	383	3043	0.7	1 (0)	100	7	63 (0)	1753	774	0.9	104 (0)
100	436	50	1.7	66 (108)	817	6624	1.0	1 (0)	311	23	38 (90)	2010	1425	1.2	221 (30)
100	436	75	—	62 (107)	—	—	—	0 (0)	—	—	0 (180)	—	—	—	20 (150)
100	436	100	—	59 (110)	—	—	—	—	—	—	0 (180)	—	—	—	—

Table 4: Computational results for the PA class. We report aggregated results over all values of $a \in \{2,3,4,5,6,7\}$, thus having 180 instances for each pair $(|V|,|\mathfrak{D}|)$.

$\|V\|$	a	t = 0.25	0.5	0.75	1.0
10	2	8	7	7	6
10	3	9	8	9	6
10	4	7	9	6	5
10	5	7	6	9	9
10	6	6	8	8	5
10	7	7	9	7	7
15	2	8	5	7	4
15	3	6	9	8	8
15	4	7	9	7	7
15	5	6	8	10	8
15	6	8	8	6	8
15	7	7	5	6	8
20	2	9	10	7	8
20	3	6	7	7	6
20	4	8	9	3	7
20	5	7	9	7	8
20	6	4	7	9	9
20	7	8	9	6	7
25	2	7	8	5	5
25	3	8	7	6	6
25	4	6	6	8	1
25	5	7	9	7	5
25	6	8	8	8	6
25	7	5	7	6	6

$\|V\|$	a	t = 0.25	0.5	0.75	1
10	2	1	11	39	153
10	3	1	3	18	117
10	4	1	8	34	89
10	5	1	8	21	144
10	6	1	8	25	92
10	7	1	8	20	207
15	2	2	7	263	2625
15	3	2	36	205	1433
15	4	2	12	95	2051
15	5	2	28	235	1217
15	6	2	15	358	1489
15	7	3	19	71	1443
20	2	7	207	7090	29302
20	3	6	126	594	9668
20	4	6	86	2848	12644
20	5	9	82	4110	72987
20	6	3	118	1323	15300
20	7	6	95	1134	15654
25	2	6	297	8556	49176
25	3	17	307	1355	109225
25	4	10	442	19433	115704
25	5	10	210	4542	84910
25	6	8	808	5126	52224
25	7	9	321	9294	106710

Table 5: Using PORTA to convert the linear description of the instances from the `PA` Hose class, geometric demand distribution. Grouping by the percentage $t \in \{0.25, 0.5, 0.75, 1.0\}$ of terminal nodes. *On the left:* Number of instances that could be converted within 1800 seconds. *On the right:* Resulting average number of vertices of the demand polytope.

			`geometric`			`uniform`			`zero-one`		
	$\|V\|$	$\|E\|$	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes
`pdh`	11	34	39	5	30	40	0	33	40	1	45
`newyork`	16	49	40	19	58	38	61	59	40	0	98
`ta1`	24	55	40	154	74	39	331	73	40	0	70
`france`	25	45	31	13	63	30	38	64	40	0	54
`norway`	27	51	38	189	109	39	34	114	40	0	84
`cost266`	37	57	38	15	183	37	423	217	40	7	203
`germany50`	50	88	31	411	498	30	239	662	40	172	575
`ta2`	65	108	39	558	525	39	38	510	40	0	413

Table 6: Computational results on the `SNDLib` Hose instances. For each of the three distributions, we consider 40 different Hose uncertainty sets per topology.

			$a=2$			$a=3$			$a=4$			$a=5$			$a=6$			$a=7$		
	$\lvert V\rvert$	$\lvert E\rvert$	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes
`geometric`	10	42	40	0	21	40	0	21	40	0	23	40	0	25	40	0	26	40	0	29
	15	77	40	0	17	40	0	40	40	0	41	40	0	56	40	0	54	40	0	52
	20	112	40	0	56	40	0	69	40	0	83	40	0	77	40	0	75	40	0	82
	25	147	40	0	86	40	0	76	40	4	138	40	0	106	40	0	115	40	0	121
	30	182	40	0	97	40	0	116	40	0	114	40	0	148	40	0	161	40	21	216
	35	217	40	0	118	40	0	140	40	3	193	40	0	175	40	3	261	40	5	274
	40	252	40	0	128	40	0	225	40	0	213	40	2	293	40	3	270	40	2	265
	45	287	40	0	185	40	0	161	40	1	299	40	9	277	40	10	369	40	0	253
	50	322	40	0	208	40	5	417	40	1	281	40	33	583	40	344	573	40	224	649
	60	392	40	0	278	40	6	429	40	25	757	40	57	774	39	579	898	39	680	973
	70	462	40	3	361	40	13	920	36	731	2121	40	4	821	37	357	1830	40	46	1400
	80	532	40	1	499	40	11	1035	31	436	2908	40	207	2083	38	547	2418	39	322	2366
	90	602	40	7	824	40	98	1917	40	23	2068	35	785	3212	37	1041	3938	34	677	4799
	100	672	40	56	1410	39	400	3524	40	525	3442	31	968	6508	19	645	7996	19	965	7299
`uniform`	10	42	40	0	23	40	0	24	40	0	23	40	0	29	40	0	29	40	0	28
	15	77	40	0	21	40	0	41	40	0	44	40	0	54	40	0	51	40	0	56
	20	112	40	0	59	40	0	71	40	0	85	40	0	90	40	0	92	40	0	92
	25	147	40	0	92	40	0	92	40	25	166	40	0	117	40	0	116	40	5	130
	30	182	40	0	98	40	0	125	40	0	125	40	0	154	40	1	174	40	84	259
	35	217	40	0	131	40	0	142	40	1	215	40	1	197	40	5	278	40	8	292
	40	252	40	0	144	40	0	220	40	1	242	40	6	322	40	4	328	40	3	264
	45	287	40	1	200	40	0	190	40	3	336	40	1	313	40	10	384	40	0	290
	50	322	40	1	240	40	53	445	40	0	311	40	45	639	40	139	718	40	94	720
	60	392	40	1	297	40	11	492	40	255	862	40	125	962	37	638	988	38	162	1250
	70	462	40	5	436	40	42	1018	28	1808	2539	40	5	818	38	698	2030	40	708	1735
	80	532	40	7	547	40	16	1172	30	1142	3588	37	364	2302	31	459	2628	37	852	2544
	90	602	40	119	977	39	381	2066	40	20	2045	37	1090	3705	34	1081	3823	33	944	5423
	100	672	40	213	1782	35	806	4401	40	163	3770	33	1439	7469	12	1232	7347	18	1730	7706
`zero-one`	10	42	40	0	18	40	0	18	40	0	19	40	0	24	40	0	27	40	0	20
	15	77	40	0	21	40	0	35	40	0	33	40	0	52	40	0	39	40	0	35
	20	112	40	0	67	40	0	62	40	0	93	40	0	80	40	0	82	40	0	90
	25	147	40	0	83	40	0	76	40	1	211	40	0	111	40	0	117	40	0	137
	30	182	40	0	87	40	0	106	40	0	118	40	0	141	40	0	181	40	4	329
	35	217	40	0	115	40	0	118	40	0	163	40	0	119	40	0	245	40	0	247
	40	252	40	0	149	40	0	231	40	0	213	40	0	321	40	0	376	40	1	285
	45	287	40	0	154	40	0	114	40	0	230	40	0	213	40	2	332	40	0	169
	50	322	40	0	184	40	1	372	40	0	211	40	1	506	40	6	600	40	4	565
	60	392	40	0	217	40	0	346	40	2	667	40	14	1063	40	37	1169	40	3	873
	70	462	40	0	315	40	2	681	40	37	2718	40	1	540	40	12	2010	40	7	1245
	80	532	40	0	390	40	3	842	40	19	2896	40	7	1682	40	317	3399	40	13	2352
	90	602	40	1	506	40	4	1062	40	7	1365	40	21	2756	40	19	2546	40	30	3733
	100	672	40	6	1131	40	16	2380	40	16	2391	40	58	6271	31	924	14748	39	100	7632

Table 7: Computational results on the `PA` Hose instances. For each of the three distributions, we consider 40 instances per pair $(|V|,|E|)$ and per $a \in \{2,3,4,5,6,7\}$.

	$\|V\|$	$\|E\|$	lp-gap (in %)	#solved (m)	cputime	#nodes	root-gap (in %)	relax-time (m)	sep-time	ip-sep-time	ipsep calls (in %)	heur-time
geometric	10	39	0.32	40 (0)	0	25	0.00	0 (0)	0	0	14.07	0
	15	69	0.24	40 (0)	0	53	0.03	0 (0)	0	0	10.45	0
	20	99	0.17	40 (0)	0	74	0.03	0 (0)	0	0	10.42	0
	25	129	0.22	40 (0)	0	114	0.03	0 (0)	0	0	9.33	0
	30	159	0.08	40 (0)	0	160	0.02	0 (0)	0	0	10.27	0
	35	189	0.16	40 (0)	3	260	0.08	0 (0)	2	2	12.60	0
	40	219	0.10	40 (0)	3	269	0.04	0 (0)	3	2	9.92	0
	45	249	0.19	40 (0)	10	368	0.09	0 (0)	8	6	13.10	0
	50	279	0.15	40 (0)	344	572	0.10	2 (0)	300	284	16.88	0
	60	339	0.13	39 (0)	579	897	0.11	6 (0)	517	486	16.26	0
	70	399	0.12	37 (0)	357	1829	0.09	15 (0)	330	303	8.31	0
	80	459	0.08	38 (0)	547	2417	0.06	26 (0)	481	440	11.93	0
	90	519	0.13	37 (0)	1041	3937	0.10	43 (0)	897	751	8.74	0
	100	579	0.07	19 (0)	645	7995	0.06	219 (0)	519	373	3.93	0
uniform	10	39	0.28	40 (0)	0	28	0.01	0 (0)	0	0	14.68	0
	15	69	0.31	40 (0)	0	50	0.05	0 (0)	0	0	13.50	0
	20	99	0.32	40 (0)	0	91	0.06	0 (0)	0	0	11.70	0
	25	129	0.16	40 (0)	0	115	0.05	0 (0)	0	0	11.59	0
	30	159	0.37	40 (0)	1	173	0.08	0 (0)	0	0	13.47	0
	35	189	0.20	40 (0)	5	277	0.11	0 (0)	4	3	14.83	0
	40	219	0.19	40 (0)	4	327	0.08	0 (0)	3	2	13.90	0
	45	249	0.11	40 (0)	10	383	0.07	1 (0)	8	7	13.06	0
	50	279	0.16	40 (0)	139	717	0.11	4 (0)	127	120	18.38	0
	60	339	0.13	37 (0)	638	987	0.10	9 (0)	570	543	18.25	0
	70	399	0.11	38 (0)	698	2029	0.09	17 (0)	631	572	12.69	0
	80	459	0.06	31 (0)	459	2627	0.05	38 (0)	413	382	11.66	0
	90	519	0.08	34 (0)	1081	3822	0.07	50 (0)	962	849	10.00	0
	100	579	0.12	12 (0)	1232	7346	0.10	391 (0)	1065	877	4.57	0
zero-one	10	39	1.53	40 (0)	0	26	0.00	0 (0)	0	0	15.81	0
	15	69	2.03	40 (0)	0	38	0.31	0 (0)	0	0	17.27	0
	20	99	1.11	40 (0)	0	81	0.10	0 (0)	0	0	12.87	0
	25	129	0.22	40 (0)	0	116	0.09	0 (0)	0	0	10.31	0
	30	159	0.66	40 (0)	0	180	0.07	0 (0)	0	0	10.64	0
	35	189	0.19	40 (0)	0	244	0.08	0 (0)	0	0	8.17	0
	40	219	0.05	40 (0)	0	375	0.05	0 (0)	0	0	7.13	0
	45	249	0.51	40 (0)	2	331	0.16	0 (0)	2	0	10.17	0
	50	279	0.22	40 (0)	6	599	0.10	2 (0)	5	3	9.70	0
	60	339	0.49	40 (0)	37	1168	0.24	10 (0)	33	25	13.97	0
	70	399	0.07	40 (0)	12	2009	0.00	11 (0)	9	3	3.52	0
	80	459	0.32	40 (0)	317	3398	0.32	47 (0)	293	269	14.46	0
	90	519	0.00	40 (0)	19	2545	0.00	17 (0)	13	2	2.00	0
	100	579	0.26	31 (0)	924	14747	0.17	604 (0)	832	713	6.98	0

Table 8: Detailed computational results for the Hose uncertainty set on the PA instances with $a = 6$.

References

[1] Y. K. Agarwal. k-Partition-based facets of the network design problem. *Networks*, 47(3):123–139, 2006.

[2] A. Altın, E. Amaldi, P. Belotti, and M. Ç. Pınar. Provisioning virtual private networks under traffic uncertainty. *Networks*, 49(1):100–115, 2007.

[3] E. Álvarez-Miranda, V. Cacchiani, T. Dorneth, M. Jünger, F. Liers, A. Lodi, T. Parriani, and D. R. Schmidt. Models and algorithms for robust network design with several traffic scenarios. In A. Ridha Mahjoub, V. Markakis, I. Milis, and V. Th. Paschos, editors, *ISCO 2012, Revised Selected Papers*, volume 7422 of *LNCS*, pages 261–272. Springer, 2012.

[4] E. Álvarez-Miranda, V. Cacchiani, A. Lodi, T. Parriani, and D. R Schmidt. Single-commodity robust network design problem: Complexity, instances and heuristic solutions. Technical Report OR-13-14, University of Bologna, 2013.

[5] G. Andreello, A. Caprara, and M. Fischetti. Embedding {0, 1/2}-cuts in a branch-and-cut framework: A computational study. *INFORMS J. on Computing*, 19:229–238, 2007.

[6] A. Atamtürk. On Capacitated Network Design Cut-Set Polyhedra. *Math. Program. B*, 92(3):425–437, 2002.

[7] P. Avella, S. Mattia, and A. Sassano. Metric Inequalities and the Network Loading Problem. In D. Bienstock and G. Nemhauser, editors, *Proceedings of the IPCO*, volume 3064 of *LNCS*, pages 401–421. Springer, 2004.

[8] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[9] F. Barahona. Network Design Using Cut Inequalities. *SIAM J. on Optimization*, 6(3):823–837, 1996.

[10] W. Ben-Ameur and H. Kerivin. Routing of Uncertain Traffic Demands. *Optimization and Engineering*, 6(3):283–313, 2005.

[11] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Oper. Res. Let.*, 25(1):1–13, 1999.

[12] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[13] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Math. Program. B*, 98:49–71, 2003.

[14] D. Bienstock, S. Chopra, O. Günlük, and C.-Y. Tsai. Minimum cost capacity installation for multicommodity network flows. *Math. Program.*, 81(2):177–199, 1998.

[15] D. Bienstock and O. Günlük. Capacitated Network Design – Polyhedral Structure and Computation. *INFORMS J. on Computing*, 8(3):243–259, 1996.

[16] C. Buchheim, F. Liers, and L. Sanità. An exact algorithm for robust network design. In J. Pahl, T. Reiners, and S. Voß, editors, *Proceedings of the INOC*, INOC'11, pages 7–17. Springer, 2011.

[17] A. Caprara and M. Fischetti. {0, 1/2}-Chvátal-Gomory cuts. *Math. Program.*, 74(3):221–235, 1996.

[18] C. Chekuri, B. F. Shepherd, G. Oriolo, and M. Scutellà. Hardness of robust network design. *Networks*, 50(1):50–54, 2007.

[19] B. V. Cherkassky and A. V. Goldberg. On implementing push-relabel method for the maximum flow problem. In E. Balas and J. Clausen, editors, *Proceedings of the IPCO*, volume 920 of *LNCS*, pages 157–171. Springer, 1995.

[20] T. Christof and A. Löbel. *PORTA — POlyhedron Representation Transformation Algorithm.* `http://typo.zib.de/opt-long_projects/Software/Porta/`, 2008.

[21] T. Dorneth. Ein Branch-and-Cut-Verfahren für robustes Netzwerkdesign. Diplomarbeit, Universität zu Köln, November 2012.

[22] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe. A flexible model for resource management in virtual private networks. In *Proceedings of the SIGCOMM*, SIGCOMM '99, pages 95–108. ACM, 1999.

[23] J. A. Fingerhut, S. Suri, and J. S. Turner. Designing least-cost nonblocking broadband networks. *J. of Algorithms*, 24(2):287 – 309, 1997.

[24] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian J. of Mathematics*, 8:399–404, 1956.

[25] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. of the ACM*, 35(4):921–940, 1988.

[26] R. E. Gomory and T. C. Hu. Synthesis of a Communication Network. *J. of the SIAM*, 12(2):pp. 348–369, 1964.

[27] D. S. Johnson, M. Minkoff, and S. Phillips. The prize collecting steiner tree problem: Theory and practice. In *Proceedings of the SODA*, SODA '00, pages 760–769. SIAM, 2000.

[28] M. Jünger and S. Thienel. The abacus system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Software: Practice and Experience*, 30(11):1325–1352, 2000.

[29] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[30] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. of the ACM*, 46(6):787–832, 1999.

[31] T. L. Magnanti, P. Mirchandani, and R. Vachani. Modeling and solving the capacitated network loading problem. Technical Report OR-239-91, MIT, 1991.

[32] T. L. Magnanti, P. Mirchandani, and R. Vachani. The convex hull of two core capacitated network design problems. *Math. Program.*, 60(1-3):233–250, 1993.

[33] T. L. Magnanti and R. T. Wong. Accelerating Benders Decomposition: Algorithmic Enhancement and Model Selection Criteria. *Oper. Res.*, 29(3):464–484, 1981.

[34] S. Mattia. The robust network loading problem with dynamic routing. *Computational Optimization and Applications*, 54:619–643, 2013.

[35] S. T. McCormick, M. R. Rao, and G. Rinaldi. Easy and difficult objective functions for max-cut. *Math. Program. B*, 94:459–466, 2003.

[36] M. Minoux. Optimum Synthesis of a Network with Non-Simultaneous Multicommodity Flow Requirements. In *Annals of Discrete Mathematics (11) Studies on Graphs and Discrete Programming*, volume 59, pages 269–277. North-Holland, 1981.

[37] S. Mudchanatongsuk, F. Ordóñez, and J. Liu. Robust solutions for network design under transportation cost and demand uncertainty. *J. of the Operational Research Society*, 59(5):652–662, 2008.

[38] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. In *Proceedings of the INOC 2007*, 2007. http://sndlib.zib.de, extended version accepted in Networks, 2009.

[39] R. Pesenti, F. Rinaldi, and W. Ukovich. An exact algorithm for the min-cost network containment problem. *Networks*, 43(2):87–102, 2004.

[40] C. Raack, A. M. C. A. Koster, and R. Wessäly. On the strength of cut-based inequalities for capacitated network design polyhedra. Technical Report 07-08, ZIB, 2007.

[41] L. Sanità. *Robust Network Design*. Ph.D. Thesis. Università La Sapienza, Roma, 2009.