Algorithm to Solve a Chance-Constrained Network Capacity Design Problem with Stochastic Demands with Finite Support

Kathryn M. Schumacher, Richard Li-Yang Chen, Amy E.M. Cohn

March 19, 2014

Abstract

We consider the design problem of determining the capacity to assign to each arc in a given network, subject to uncertainty in the supply and/or demand of each node. This design problem underlies many real-world applications, such as the design of power transmission and telecommunications networks. We first consider the case where a finite set of potential supply/demand scenarios are provided, and we must determine the minimum-cost set of arc capacities such that a feasible flow exists for each of the scenarios. We briefly review existing theoretical approaches to solving this problem and explore implementation strategies to reduce run times. With this as a foundation, our primary focus is on a chance-constrained version of the problem in which $\alpha\%$ of the scenarios must be feasible under the chosen capacity, where α is a user-defined parameter and the specific scenarios to be satisfied are not pre-determined. We present a novel approach for solving this problem which embeds a constraint generation algorithm into a tree-based, parallelizable framework. We present theoretical and computational analysis to evaluate the performance of our proposed approaches.

1 Introduction

In many real-world contexts, such as transportation systems, the power grid, telecommunications networks, and gas pipeline networks, planners need to determine the amount of capacity to build on the arcs in a network. Such decisions must often be made before nodal supplies and demands are known. Furthermore, these supplies and demands may not be static but rather vary over time while the network has to remain fixed. We consider the problem of determining the minimum-cost set of arc capacities to install in a single-commodity network when there is uncertainty in the nodal supplies and demands.

We represent uncertainty as a finite set of possible scenarios, where each scenario is a set of nodal supplies and demands. The set of scenarios could exactly represent a probability distribution with finite support. Alternatively, the scenarios could represent an approximation of a general probability distribution, where scenarios are generated by Monte Carlo sampling techniques. Using techniques such as *Sample Average Approximation*, it has been demonstrated that a finite set of scenarios can be used to find good solutions for the original distribution [10]. As is common in sampling based approaches for generating scenarios, we assume that all scenarios have equal probabilities of realization. If the set of scenarios represents a discrete probability distribution where the scenarios have different rational probabilities, the set of scenarios can be transformed into a set where all scenarios have equal probabilities by making copies of the scenarios in proportion to the probabilities of realization. It is assumed here that these scenarios have been given as an input.

In long term planning, it is common to require feasible operating conditions *almost* all the time, because requiring feasible conditions under absolutely all future scenarios is typically prohibitively expensive. Our goal is to find a minimum-cost set of arc capacities such that there exists a feasible set of flows for $\alpha\%$ of all scenarios. In practice α might be chosen to be a value like 99.5%.

We consider only the costs of installing capacity on arcs, as investment costs tend to be the dominant costs. Additionally, we assume the total cost is a linear function of the capacity installed. This cost structure might arise if the network user must lease arc capacities from the owner of a pre-existing network. Alternatively, this problem might arise when determining how to upgrade the arc capacities in an existing network. For the sake of exposition, we assume that there is no existing capacity on any arc, but the approach presented here can easily be modified to incorporate existing arc capacities.

The rest of this paper is structured as follows. In Section 2, we review the relevant literature on network design problems and approaches for dealing with parameter uncertainty. In Section 3, we describe the *Robust Capacity Design problem (RCD)*, in which a set of feasible flows is required to exist for every scenario in the known set. We review the solution approach for solving RCD first presented in [8], and empirically demonstrate that a decomposition procedure can be used to solve this problem quickly, even when the number of scenarios is large. In Section 4, we build on this work and present a novel algorithm to solve the chance-constrained problem variant in which only some fraction, $\alpha\%$ (typically close to but less than 100%), of the scenarios are required to have a set of feasible flows; this is called the *α-Satisfied Capacity Design problem (αSCD)*. Our proposed algorithm for solving this problem embeds the constraint generation algorithm presented for RCD into a tree-based framework, and we argue that a parallelized implementation can provide significant benefit. Furthermore, we present a greedy algorithm that can quickly solve for a good heuristic solution under certain conditions. Finally, in Section 5 we conclude with a summary and suggestions for future research.

2 Literature Review and Contributions

[8] consider the problem of identifying the minimum cost set of arc capacities such that feasible flows are ensured for a set of different single-commodity demand requirements, each defined by a pair of origin-destination nodes and a demand amount. The authors present a constraint generation algorithm to solve this problem based on network cut-sets. An algorithm for solving the variant of this problem in which capacities are integer-valued is provided in [26].

While the network design formulations presented in [8] and [26] were originally formulated to represent deterministic demand requirements in different time periods, the formulations are nearly identical to the RCD problem defined in Section 3, where the set of scenarios represent uncertainty in the future supplies and demands. The only difference is that we allow any number of nodes to act as sources or sinks in each scenario, as opposed to a single origin-destination pair for each scenario. The constraint generation algorithm presented in [8] is reviewed in Section 3.3.

Many authors have considered variants of this RCD problem. [13] extend the algorithm presented in [8] to solve a multi-commodity variant of the same problem, called a *non-simultanous flows* multi-commodity network design problem. A review of other related work is provided in [14]. More recently, [18] have addressed the same muli-commodity design problem as [13], and empirically compared the performance of two different types of cutting planes within a constraint generation algorithm.

A variety of other robust network design problems have been addressed in the literature.

[6] present a cutting plane algorithm to solve a capacity design problem for a multicommodity network where demands are deterministic, investment costs are minimized, and feasibility is required in the event of any single arc or node failure. [22] consider a robust multi-commodity capacity design problem where the worst case value of a function which penalizes unsatisfied demand is minimized over the set of demand scenarios. A cutting plane solution approach is proposed. [19] consider a similar problem as [22] but instead use an *Affinely Adjustable Robust Counterpart* to compute tighter bounds. [15] prove that the robust multi-commodity capacity design problem with a polyhedral uncertainty set for the demand is NP-hard.

[21] consider the capacity design problem in which demands are expressed in terms of *bilateral contracts*, i.e., agreements between node pairs in which a supplier agrees to meet a customer's demand for any amount within a fixed range; to solve this problem the authors use a cutting plane algorithm. [3] consider the robust capacity design problem where demand is assumed to belong to a budget uncertainty set, and the goal is to minimize the worst case total of investment and routing costs. The authors present a procedure for obtaining bounds, and computational results for several special problem instances. [16] address a similar problem where routing costs are additionally assumed to be uncertain. The authors explore polyhedral and ellipsoidal uncertainty sets, and present a column generation procedure to solve a path constrained problem variant. [17] present a tractable conic LP formulation of a robust capacity design problem in which demands and travel times belong to polyhedral uncertainty sets, worst case travel time is minimized, and investment costs are constrained to be less than a specified budget.

A challenge with robust network design problems in general is that it is difficult to define an uncertainty set that appropriately controls the conservatism of the optimal solution. The focus of this paper is the chance-constrained αSCD problem, where a fraction of the scenarios are allowed to be infeasible. An advantage of this formulation is that it allows the conservatism of the optimal solution to be controlled with a single parameter α.

There are many different approaches in the literature for solving chance-constrained problems. Several authors including [1], [9] and [11] present mixed-integer formulations of chance-constrained problems and describe several types of valid inequalities that can be added to strengthen these formulations. [12] present a branch-and-cut decomposition algorithm for solving the mixed-integer formulation of the chance-constrained problem.

Other authors such as [4], [7], [23] and [24] suggest methods that utilize *p-level efficient points* of discrete distributions to develop equivalent formulations of probabilistic constraints. Alternatively, [27] propose an algorithm based on *progressive hedging* which can solve for a heuristic solution quickly, relative to other methods, even for large problems. These published methods have each been developed to solve a fairly general class of chance-constrained problems. We propose a method for solving a very particular chance-constrained problem and we are able to exploit the problem structure to our advantage.

3 Robust Capacity Design Problem

In this section we review the problem of determining the minimum cost set of arc capacities to install in a given network such that there exists feasible flows for all scenarios in the given set. We first present a traditional network-flow based formulation, and then we review the constraint generation approach proposed by [8]. We discuss implementation details to augment the theoretical discussions in [8] and other works.

We assume that in each scenario the total supply equals the total demand. Flows are defined to be feasible for a particular network if they satisfy flow balance constraints and arc

capacity constraints.

3.1 Notation

Sets

N set of all nodes in the network.
A set of all arcs in the network. For each element $(i, j) \in A,\ i, j \in N,\ i \neq j$.
Ω set of all scenarios, where each scenario is a set of supplies/demands for all nodes.

Parameters

c_{ij} cost of installing a unit of capacity on arc (i, j), for all $(i, j) \in A$.
b_i^ω supply at node i in scenario ω, for all $i \in N,\ \omega \in \Omega$.
Demand is represented as negative supply.

Variables

f_{ij}^ω flow on arc (i, j) in scenario ω, for all $(i, j) \in A,\ \omega \in \Omega$.
u_{ij} capacity to be installed on arc (i, j), for all $(i, j) \in A$.

3.2 Network Flow Formulation

We present here formulations for a directed network, but these formulations could easily be modified to model an undirected network. RCD can be modeled with a traditional network-flow formulation as follows:

$$\min \sum_{(i,j)\in A} c_{ij} u_{ij} \tag{1a}$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in A} f_{ij}^\omega - \sum_{j:(j,i)\in A} f_{ji}^\omega = b_i^\omega \quad \forall i \in N,\ \omega \in \Omega \tag{1b}$$

$$f_{ij}^\omega - u_{ij} \leq 0 \quad \forall (i, j) \in A,\ \omega \in \Omega \tag{1c}$$

$$f_{ij}^\omega \geq 0 \quad \forall (i, j) \in A,\ \omega \in \Omega \tag{1d}$$

$$u_{ij} \geq 0 \quad \forall (i, j) \in A \tag{1e}$$

Objective (1a) states that the total cost of installing capacity is to be minimized. For each scenario $\omega \in \Omega$, constraint (1b) enforces that flow is conserved at each node and constraint (1c) enforces that arc flow cannot exceed capacity. Because the number of flow variables f and the number of constraints depend on the number of scenarios, this LP may be intractable if the number of scenarios $|\Omega|$ is large.

3.3 Cut-set Solution Approach

In the network design problem addressed here, we only consider the cost of investing in new capacity. This suggests the potential value of a formulation that depends only on the arc capacity variables u, as these variables do not depend on the number of scenarios and are the only variables in the objective function.

Let Θ be the set of all nonempty proper subsets of nodes in N, i.e., $\Theta = \{\theta \subset N : \theta \neq \emptyset\}$. $|\Theta| = 2^{|N|} - 2$. The following LP can also be used to solve for the minimum-cost arc capacity

assignment such that a set of feasible flows exists for all scenarios.

$$\min \sum_{(i,j)\in A} c_{ij}u_{ij} \tag{2a}$$

$$\text{s.t.} \quad \sum_{i\in\theta, j\notin\theta} u_{ij} \geq \sum_{i\in\theta} b_i^\omega \quad \forall\theta\in\Theta, \omega\in\Omega \tag{2b}$$

$$u_{ij} \geq 0 \quad \forall(i,j)\in A \tag{2c}$$

Constraint set (2b) states that for any network cut-set defined by the partition of nodes θ and $N\backslash\theta$, the total capacity of the arcs contained in the cut-set is at least equal to the total net supply for the nodes in θ, for each scenario. These cut-set constraints (2b) are both a necessary and a sufficient condition for the existence of a feasible flow for every scenario (Theorem 6.12, pp. 196, [2]).

The number of constraints in (2b) can be reduced to exactly one constraint per node subset by recognizing that for all $|\Omega|$ cut-set constraints for a given θ, one constraint dominates all other constraints. For a given θ, if the constraint with the greatest right hand side is satisfied, all other constraints will immediately be satisfied. Let $M(\theta) \equiv \max_{\omega\in\Omega}\{\sum_{i\in\theta} b_i^\omega\}$. Constraint set (2b) can be replaced by the following constraint set:

$$\sum_{i\in\theta, j\notin\theta} u_{ij} \geq M(\theta) \quad \forall\theta\in\Theta. \tag{3}$$

The number of constraints in (3) is determined entirely by the number of node subsets $|\Theta|$, which is a function only of the number of nodes $|N|$. Thus, the size of this linear program is completely independent of the number of scenarios $|\Omega|$. However, there are an exponential number of constraints in the set (3).

As proposed in [8], we suggest that the cut-set LP (2) with constraint set (3) be solved via a constraint generation procedure which will here be referred to as the Cut-set Constraint Generation (CSCG) algorithm.

Let the *Master Problem (MP)* be a relaxation of the cut-set LP. In each iteration, MP is solved, and for each scenario in the set Ω, a subproblem is solved to identify constraints from the set (3) that are violated for the current MP solution. The subproblem identifies a minimum cut-set for the current capacity assignment and the given scenario. There exist several methods for formulating and solving the min cut-max flow problem, and any one of these methods may be used to identify the minimum cut-set. The node subset θ corresponding to this minimum cut-set is used to identify a constraint from the set (3) that is not satisfied, and this constraint is then added to the MP. Note that while a particular scenario ω is used to identify the minimum cut-set, the corresponding constraint added to the MP is valid for all scenarios. The right hand side of the cut-set constraint is $M(\theta)$, which ensures that sufficient capacity is installed on this cut-set for all scenarios.

The procedure of iteratively solving the MP and adding violated constraints is repeated until, for all scenarios, the subproblem identifies that all constraints in the set (3) are satisfied. When this occurs, the algorithm exits with the optimal capacity assignment that is feasible for all scenarios.

3.4 Implementation Details

To provide a practical assessment of the theoretical concepts provided by earlier authors, we present here an analysis of the algorithm's performance in implementation. In this section we discuss the implementation details which have a significant impact on the algorithm's rate of convergence. We

found that the three most important issues are (i) the choice of initial MP constraints, (ii) the rules for selecting a scenario from the list Ω to define the subproblem, and (iii) how many identified violated constraints are added to the MP per iteration.

On the first issue, we experienced our best run times when we initialized the MP with one constraint per node requiring that the total capacity on the arcs directed out of a node be at least the maximum supply at the node across all scenarios (if a supply node) or that the total capacity on the arcs directed into a node must be at least the maximum demand at the node (if a demand node). We assume that each node acts only as a supply node or as a demand node in all scenarios, but these initializing constraints could also be used if a single node acts both as a demand node and as a supply node in different scenarios.

On the second issue, we found that the best policy was to maintain two mutually exclusive and collectively exhaustive lists of scenarios: the current list and the set-aside list. The current list is a list of scenarios that is currently being screened. When a scenario is identified to be feasible for the current set of arc capacities, it is moved from the current list to the set-aside list. The current list is then shorter the next time it is looped through, and generally contains the more "difficult" scenarios, while "easy" scenarios get moved to the set-aside list. In order to guarantee that the algorithm exits with an optimal solution, when the current list becomes empty it is necessary to loop through the entire list of set-aside scenarios to confirm that all scenarios are feasible. If all scenarios are not feasible, the process is repeated. The idea is that this policy avoids needlessly and repeatedly checking "easy" scenarios that have been found to be feasible in an early iteration and are likely to remain feasible in later iterations. Instead, the algorithm can focus on the "difficult" scenarios. Our computational results indicate that this policy results in the fewest total subproblems solved on average, which results in the fastest run time.

Finally, on the third issue, we recommend that scenario subproblems from the current list are solved only until a cut-set with insufficient capacity is identified. At this time one violated constraint is added to the MP and the MP is solved again. We have found that this approach of adding one constraint to the MP per iteration causes the algorithm to solve faster than when multiple constraints are added to the MP during each iteration.

3.5 Run Time Results

Using the implementation options described in Section 3.4, we present computational experiments that indicate how the performance of the CSCG algorithm scales with the number of scenarios.

Our first test network is an IEEE 118 node test system ([5]), which is representative of a portion of the U.S. power grid. Supply/demand scenarios were generated by perturbing and scaling the nominal power generation and consumption levels provided for each of these nodes in the IEEE test system. The perturbation was done by adding a normally distributed random variable to the nominal supply/demand at each node, where the normal distribution has mean of 0 and standard deviation set so as to make the coefficient of variation (COV) equal to 0.25, if the nominal supply is positive, or equal to 0.75 if the nominal supply is negative. These COV values were chosen because, in a conventional power grid, the demand for power is typically more variable than the generation of power. Scaling was done by multiplying the perturbed supplies/demands by a random value, which is drawn from uniform distribution with minimum 0.1 and maximum 2.0. An additional dummy node was added to make the total supply equal the total demand in each scenario by supplying or demanding whatever is excess (i.e., $b^{\omega}_{119} = -\sum_{i=1}^{118} b^{\omega}_{i}\ \forall \omega \in \Omega$). An arc was added to and from each of the 118 nodes to this dummy node to ensure feasibility, changing the number of arcs from 358 to 594.

While the IEEE118 test network is a useful example of a real world network, it is a fairly

sparse network. To test the performance of the CSCG algorithm on a dense network, for our second test instance we constructed a network with 30 nodes that is exhaustively dense, i.e., there exists an arc between every node pair. Let the test instance be called exh30. Supplies in each scenario were randomly generated from a continuous uniform distribution $[-10, 10]$. A dummy node was added and its supply/demand was set in each scenario to make the total supply equal the total demand.

For all test instances presented in this paper, the arc costs c_{ij} were randomly generated from a discrete uniform distribution $[1, \ldots, 50]$. All algorithms were implemented in C++ which called CPLEX v12.4 to solve all linear programs, on a computer with a 2.3 GHz processor and 4G RAM. In Figure 1, the run time results shown are averaged over 5 trials and the error bars indicate one standard deviation in these run times.

Figure 1 shows the average run time of the CSCG algorithm over a range of different numbers of scenarios up to 40,000 for both the IEEE118 and exh30 test systems. For both of these networks, the run time increases approximately linearly as the number of scenarios increases. The coefficient of determination (R^2) of the linear regression is 0.99 for both test systems.

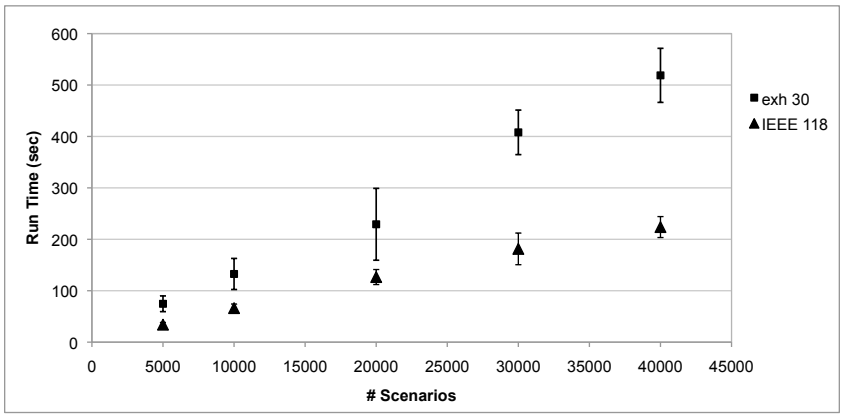

Figure 1: CSCG Algorithm Run Time for IEEE118 and exh30 test systems

IEEE118 has more nodes than exh30, and so the number of cut-set constraints in constraint set (3) is larger. The number of cut-sets for a 118 node network is on the order of 10^{35}, compared to 10^9 for a 30 node network. However, it seems that the dominant factor in determining the runtime is not the number of total cut-sets but the number of arcs. The denser exh30 has 870 arcs, compared to 594 arcs for the IEEE118. The number of variables in the master problem is equal to the number of arcs. This difference appears to drive the difference in runtimes shown in Figure 1. The CSCG algorithm solves the IEEE 118 node test case with 40,000 scenarios in less than 4 min, and the exhaustive 30 node test instance with 40,000 scenarios in about 8 and a half minutes.

4 α-Satisfied Capacity Design Problem

Having reviewed a successful approach for solving the RCD problem, we now explore the more challenging chance-constrained problem where some percentage α (typically close to but less than 100%) of all the scenarios in the set Ω are required to have feasible flows.

We first consider a Mixed Integer Program (MIP) formulation of the αSCD problem, and then discuss the difficulties of solving this formulation directly. The difference between the αSCD problem and the RCD problem is that the set of scenarios for which feasibility is required is a

decision for αSCD, whereas it is given for RCD. The RCD problem is relatively easy to solve, so we propose an approach to solve αSCD that employs a combinatorial tree-based framework for exploring subsets of scenarios for which feasibility could be required. We present an algorithm which embeds the CSCG algorithm within this tree to find the optimal solution. Finally, we present a greedy variation of this approach that can be used to solve for a heuristic solution, and we analyze the solution quality.

4.1 MIP Formulation

We introduce a set of binary indicator variables I_ω for all $\omega \in \Omega$ which enable us to determine which scenarios have feasible network flow solutions. Let I_ω equal 1 if there exists a set of feasible flows for scenario ω, and equal 0 otherwise. The MIP formulation of the αSCD problem is as follows.

$$\min \sum_{(i,j)\in A} c_{ij} u_{ij} \tag{4a}$$

$$\sum_{i\in\theta, j\notin\theta} u_{ij} - \left(\sum_{i\in\theta} b_i^\omega\right) I_\omega \geq 0 \quad \forall \theta \in \Theta, \omega \in \Omega \tag{4b}$$

$$\sum_{\omega\in\Omega} I_\omega \geq \left(\frac{\alpha}{100}\right) |\Omega| \tag{4c}$$

$$u_{ij} \geq 0 \quad \forall (i,j) \in A \tag{4d}$$

$$I_\omega \in \{0,1\} \quad \forall \omega \in \Omega \tag{4e}$$

In this formulation, constraint set (4b) is analogous to the cut-set constraint set (2b); if $I_\omega = 1$, the constraints require that demand must be fully satisfied in scenario ω, or if $I_\omega = 0$, the constraint is not restrictive. Constraint (4c) states that at least $\left(\frac{\alpha}{100}\right)|\Omega|$ binary variables must be equal to 1, indicating that at least $\alpha\%$ of all scenarios in Ω must be feasible.

Due to the binary variables, formulation (4) is difficult to solve. If the set of scenarios Ω is large, (4) contains a large number of binary variables which have significant incentive to be fractional. For example, it is typically much cheaper to install half as much capacity as is needed on each cut-set for two different scenarios than to fully satisfy all cut-set constraints for one scenario. Thus, at the optimal solution of the LP relaxation of (4), the I_ω variables will have fractional values and a branch-and-bound algorithm would require a lot of branching to fix the indicator variables to integer values.

Additionally, because the left hand side of each constraint in the set (4b) contains the scenario-specific variable I_ω, the number of constraints cannot be reduce to 1 constraint per node subset θ as was done in the formulation of the RCD problem. Every cut-set constraint is scenario-specific. Essentially, the Benders' feasibility cuts in the set (4b) are much weaker than the feasibility cuts for the RCD problem (3). A decomposition procedure for solving (4) is not promising due to the combination of weak Benders' cuts and a master problem which is difficult to solve.

There are several other approaches for solving mixed integer formulations of general chance-constrained programs, including enumerating p-efficient points as in [4], generating valid inequalities as in [25], or the branch-and-cut algorithm described in [12]. In this paper, we offer an alternative approach that leverages the specific structure of αSCD, taking advantage of the fact that l, the number of scenarios allowed to be infeasible, is typically very small, and the fact that if the set of scenarios allowed to be infeasible were known, the resulting RCD problem could be solved quickly with the CSCG algorithm.

4.2 Combinatorial Tree Algorithm

There are a finite number of distinct subsets of Ω of cardinality $\lceil \left(\frac{\alpha}{100}\right) |\Omega| \rceil$, so one approach to solve this problem is to enumerate all such subsets and apply the CSCG algorithm for each subset. One way of organizing all such subsets is to use a tree. At the root node, the set of scenarios required to be feasible is equal to the complete set Ω. At depth d in the tree there are $\binom{\Omega}{d}$ nodes, where at each node the set of scenarios required to be feasible is the complete set Ω minus a unique set of d scenarios. At the bottom level at depth $l = \lfloor \left(1 - \frac{\alpha}{100}\right) |\Omega| \rfloor$ there is a node for every distinct set of scenarios of cardinality $\lceil \left(\frac{\alpha}{100}\right) |\Omega| \rceil$. The optimal solution could be found by applying the CSCG algorithm to each of the nodes in the bottom level and identifying the node with the overall minimum-cost. A tree constructed as described is illustrated in Figure 2, where each tree node is labeled with the set of scenarios that are allowed to be infeasible.

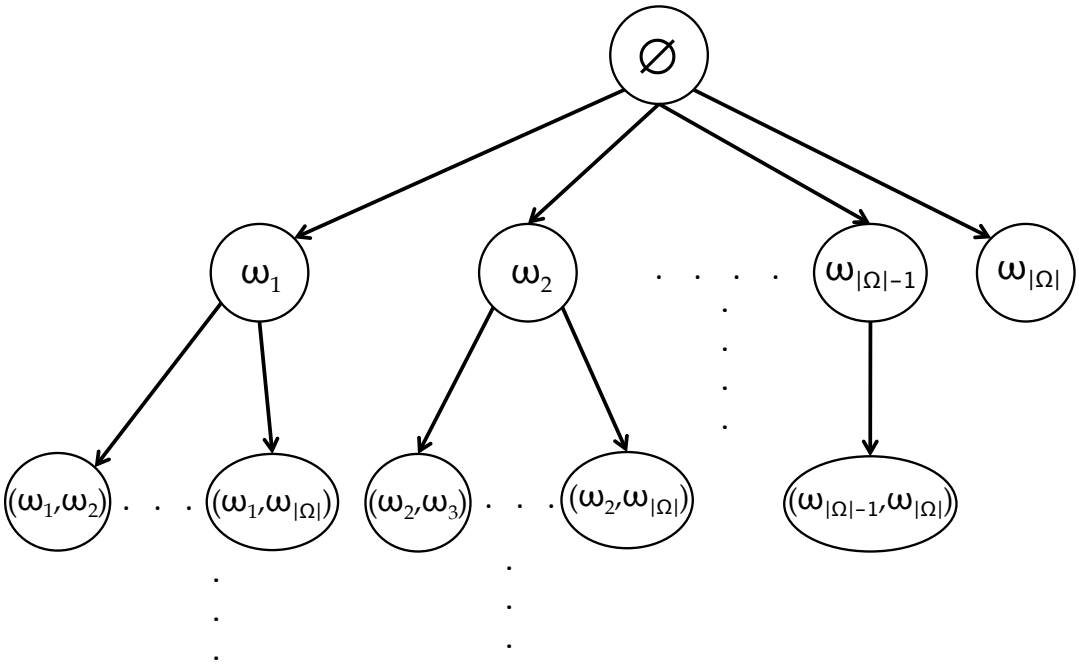

Figure 2: Combinatorial Tree

With the described tree, the number of nodes at bottom level $\binom{\Omega}{l}$ could be very large even if l is relatively small. However, as we build the tree, we can identify some branches that will contain only suboptimal solutions and therefore we can prune these branches.
First we introduce useful terms and notation.

- When the CSCG algorithm terminates for a single RCD problem, the final MP includes a set of constraints that have been generated over the course of the algorithm which we will call *explicit constraints*.
- A subset of the explicit constraints will be tight at the optimal solution to a final MP; we call these *active explicit constraints*.
- For any cut-set constraint with corresponding node subset θ, there are one or more scenarios that define the right hand side by having the greatest net supply over θ. Let any scenario such that $\bar{\omega} = \arg\max_{\omega \in \Omega} \{\sum_{i \in \theta} b_i^{\omega}\}$ be the *dominant scenario* for the given constraint.

- A *binding scenario* is a scenario that is dominant for at least one active explicit constraint.
- Each node in the combinatorial tree is a RCD problem where the set of scenarios required to be feasible is $(\Omega \backslash E)$, where E is the *exclusion set*.
- The set of binding scenarios for a node with exclusion set E is denoted $B(E)$ and the optimal objective value for that node is $V(E)$.
- The goal of αSCD is to both find an optimal set of arc capacities u and corresponding exclusion set E with cardinality $l = \lfloor \left(1 - \frac{\alpha}{100}\right) |\Omega| \rfloor$. Let this optimal set of excluded scenarios be denoted E_l^*.

We propose a branching rule that is based on the idea that, given a current node in the tree with exclusion set E, we can identify some branches that will contain only suboptimal solutions based on what scenarios are not in the set $B(E)$. In particular, a scenario that is not dominant for any active explicit constraints is implicitly dominated by one or more other scenarios that are contained in the set $B(E)$. Thus branching only on scenarios contained in the set $B(E)$ is sufficient. Formal theorems and proofs will now be stated, and then the branching rule will be presented in more precise terms.

4.2.1 Reduced Combinatorial Tree Theorems

Lemma 1: *For any scenario $\hat{\omega} \in \Omega \backslash B(E)$, $V(E \cup \hat{\omega}) = V(E)$.*

In other words, if a scenario is not binding, then excluding that scenario will not have any effect on the feasible region, and thus the optimal objective value will not change. The formal proof for this lemma is as follows.

Proof. The cut-set LP for RCD with exclusion set E could be appropriately modified to solve RCD with exclusion set $(E \cup \hat{\omega})$ by relaxing every constraint in the set (3) whose dominant scenario is $\hat{\omega}$. The CSCG algorithm for solving RCD with exclusion set E exits when the MP contains a set of explicit constraints such that any solution u satisfying these constraints will also satisfy all cut-set constraints that were not explicitly added to MP. Relaxing any of these implicitly satisfied constraints has no effect on the optimal solution. Relaxing any explicit constraint in the final MP that is not tight at the optimal solution will not change the optimal solution. By the choice of $\hat{\omega} \notin B(E)$, there does not exist an active explicit constraint for which scenario $\hat{\omega}$ is dominant. Thus $V(E) = V(E \cup \hat{\omega})$. □

Lemma 2: *There exists an optimal exclusion set for αSCD, E_l^*, that takes the form $\{\omega_1, \omega_2, \ldots, \omega_l\}$ where $\omega_1 \in B(\emptyset)$ and $\omega_i \in B(\{\omega_1, \ldots, \omega_{i-1}\})$ for all $i = 2, \ldots, l$.*

In other words, the optimal exclusion set can be ordered such that each scenario is binding for the exclusion set equal to all lower ordered scenarios. Binding scenarios are the only scenarios whose exclusion has the potential to improve the objective value. So when growing the optimal exclusion set from the empty set, only binding scenarios should be candidates for the next scenario to exclude. The formal proof for this lemma is as follows.

Proof. Suppose, in contradiction, that all optimal exclusion set(s) take the form $E_l = \{\hat{\omega}_1, \ldots, \hat{\omega}_l\}$ where scenarios $\hat{\omega}_1, \ldots, \hat{\omega}_l \in \Omega \backslash B(\emptyset)$. If RCD is solved for exclusion set $\emptyset$, and then, one by one, each scenario in the set E_l is added to the exclusion set, by Lemma 1, the optimal objective value will not change, i.e., $V(\emptyset) = V(E_l)$. RCD with an exclusion set of cardinality l which includes a

scenario $\omega_i \in B(\emptyset)$ will have an objective value less than or equal to the objective value of $V(E_l)$, thus it is not possible that all exclusion sets have the supposed form. There is a contradiction.

Suppose, for some $2 \leq k \leq (l-1)$, that all optimal exclusion set(s) take the form $E_l = \{\hat{\omega}_1, \ldots, \hat{\omega}_l\}$ where $\hat{\omega}_1 \in B(\emptyset)$ and for all $i = 2, \ldots, k$, $\omega_i \in B(\{\hat{\omega}_1, \ldots, \hat{\omega}_{i-1}\})$, but all scenarios $\hat{\omega}_{k+1}, \ldots, \hat{\omega}_l \in \Omega \backslash B(\{\hat{\omega}_1, \ldots, \hat{\omega}_k\})$. If RCD is solved for $E = \{\hat{\omega}_1, \ldots, \hat{\omega}_k\}$, and then, one by one, all other scenarios in the set E_l are also excluded, by Lemma 1, the optimal objective value will not change, i.e., $V(\{\hat{\omega}_1, \ldots, \hat{\omega}_k\}) = V(E_l)$. RCD with an exclusion set of cardinality l which includes scenario $\hat{\omega}_1, \ldots, \hat{\omega}_k$ and includes a scenario $\omega_i \in B(\{\hat{\omega}_1, \ldots, \hat{\omega}_k\})$ will have an objective value less than or equal to the objective value of $V(E_l)$, thus it is not possible that all exclusion sets have the supposed form. There is a contradiction. □

Given the stated lemmas, we present the following algorithm which constructs a tree of RCD problems and finds the optimal set of arc capacities u for optimal exclusion set E_l^*. The tree nodes referred to in this algorithm each represent an RCD problems with different exclusion sets, as illustrated in Figure 2.

Reduced Combinatorial Tree Algorithm:

1. Initialize the Last In First Out (LIFO) queue of tree nodes to contain only the root node, which is an RCD problem with exclusion set $E = \emptyset$. Initialize the current best objective value $v = \infty$ and the current best arc capacities $u = \emptyset$.

2. Pop off a tree node from the LIFO queue with exclusion set E. Solve this RCD problem. If $|E| = l$ and $V(E) < v$, update $v = V(E)$ and let u be the optimal solution to this RCD problem. Otherwise, if $|E| < l$, find the set of binding scenarios $B(E)$. For each scenario $\omega \in B(E)$, add a tree node to the LIFO queue which has exclusion set $\{E \cup \omega\}$ if a tree node with this exclusion set does not already exist in the LIFO queue.

3. If the LIFO queue is not empty, go to step 2. Otherwise, exit with the optimal set of arc capacities u.

Theorem: The Reduced Combinatorial Tree Algorithm will exit with the optimal set of arc capacities u for optimal exclusion set E_l^*.

Proof. The Reduced Combinatorial Tree Algorithm constructs a tree node for every exclusion set E of size l with form $\{\omega_1, \omega_2, \ldots, \omega_l\}$ where $\omega_1 \in B(\emptyset)$ and $\omega_i \in B(\{\omega_1, \ldots, \omega_{i-1}\})$ for all $i = 2, \ldots, l$. By Lemma 2, the tree contains a node for every exclusion set that satisfies the necessary condition to be the optimal exclusion set, and thus includes the optimal exclusion set E_l^*. The algorithm finds the set of arc capacities that has the lowest cost for the RCD problem among these tree nodes with exclusion set of size l, which is the optimal set of arc capacities. □

4.2.2 Reduced Combinatorial Tree Size

In our computational experiments with the Reduced Combinatorial Tree (RCT), we observe that the total number of tree nodes in each level to be significantly reduced from the number of tree nodes in the complete combinatorial tree. Instead of $\binom{|\Omega|}{k}$ tree nodes in the kth level of the tree, the number of tree nodes in each level is a function of the number of binding scenarios in the tree nodes one level above. The set of binding scenarios $B(E)$ for a tree node with exclusion set E is the set of dominant scenarios for the active explicit constraint. The size of this set $|B(E)|$ depends on the number of active explicit constraints in MP, which depends on the number of variables, $|A|$, and on

how close to pareto dominant the set of scenarios Ω is. For example, if all the scenarios were scalar multiples of a base scenario, then the scenario with the largest scalar multiplier would be pareto dominant over all other scenarios, and would define the right hand side for every possible cut-set constraint. Then, regardless of how many active explicit constraints there were in the MP, there would be exactly one binding scenario, and exactly one tree node in each level. For any other type of scenario set, the number of binding scenarios will likely be low if a few scenarios are dominant over all other scenarios, or will likely be higher if each cut-set has a different defining scenario.

To analyze the number of tree nodes required to be solved in the RCT for different problem instances, we performed computational experiments with an IEEE 30 node test system [5]. We generated scenarios using two different methods. One scenario generation method, labeled as "Scaled & Perturbed Supplies" in Table 1, is the same as was described in Section 3.5 for IEEE118. The other method, labeled as "Uniform Supplies" in Table 1, is the same as was described in Section 3.5 for exh30. For both ways of generating the scenarios, a dummy network node was used to make the supplies and demands net to 0 in each scenario, as was also described in Section 3.5.

The numbers of tree nodes created at each level of the RCT are presented in Table 1. From Table 1, it is evident that the scenario distribution has a significant impact on how many nodes must be generated in the tree. The number of tree nodes is significantly greater when the scenarios are drawn from a uniform distribution than when generated by scaling and perturbing a base scenario. When the scenarios are generated from a uniform distribution, the scenarios are very different from each other, and thus the active explicit cut-set constraints are likely to have different dominant scenarios, and the set of binding scenarios is larger. When the scenarios are generated by perturbing and scaling a base scenario, the scenarios are more similar to each other, and the active explicit cut-set constraints are more likely to be have common dominant scenarios, and the set of binding scenarios is smaller.

We note that while the number of tree nodes in each level increases exponentially, all tree nodes in a particular level are independent of each other and could be solved in parallel. If all tree nodes per level were to be parallelized, the time to solve the reduced tree and find the optimal solution is equal to the time to run the CSCG algorithm l times. Given a fixed number of processors p, the time required to solve the tree could be approximated by taking the maximum node solve time for each level, and multiplying this by the number of nodes in that level of the tree divided by p.

4.3 Greedy Algorithm

While utilizing parallelization may allow the RCT to be solved relatively quickly to find the optimal solution for certain problem instances, a greedy algorithm can alternatively be used to generate a heuristic solution more quickly without the need for a parallel implementation. The basic idea behind the algorithm is that scenarios are added one by one to the set of excluded scenarios, greedily choosing the scenario whose exclusion most improves the cost. A similar algorithm, which gradually excludes constraints to solve a chance-constrained problem, is presented in [20]. We demonstrate that the heuristic is close to optimal for the problem instances we tested.
The greedy algorithm is as follows:

1. Solve the RCD problem with the exclusion set $E = \emptyset$. Initialize iteration $d = 0$. Let $B(\emptyset)$ be the current set of binding scenarios.

2. Increase d by 1. If $d < l$, for each scenario ω in the current set of binding scenarios $B(E)$, solve a RCD problem with exclusion set $\{E \cup \omega\}$.

			Uniform Supplies				Scaled & Perturbed Supplies			
Num. Scenarios	α	l	Total Tree Nodes	Level 1	Level 2	Level 3	Total Tree Nodes	Level 1	Level 2	Level 3
67	97%	2	696	34	661	-	250	21	228	-
100	97%	3	15,020	43	940	14,036	2,911	24	299	2,587
300	99 %	3	28,396	50	1,383	26,962	5,200	29	441	4,789
600	99.5 %	3	33,829	55	1,590	32,183	8,229	35	624	7,569

Table 1: Number of Reduced Combinatorial Tree Nodes

3. Among these RCD problems, find the exclusion set E whose optimal objective value $V(E)$ is smallest. Set the current set of binding scenarios equal to $B(E)$. Go to step 2.

The final solution u that this algorithm exits with is a feasible solution, but it is not guaranteed to be an optimal solution. However, computational results indicate that for certain types of problems the solution u is often optimal or close to optimal.

4.4 Heuristic Computational Results

We tested the greedy algorithm on the IEEE30 test system and on an exhaustively dense 20 node test network, constructed in the same way as exh30 described in Section 3.5. In Table 2 the column "Heuristic Obj. Value" is the heuristic objective value returned by the greedy algorithm, and these values are shown next to the optimal objective value. The run times for the greedy algorithm for this set of computational experiments are listed in Table 3.

For almost all of the computational experiments presented in Table 2 the greedy algorithm returns an optimal objective value. The only exception was for IEEE30 Uniform with $\alpha = 97\%$. As discussed in section 4.2.2, if the set of scenarios was pareto dominant, meaning that all scenarios were a scalar multiple of a base scenario, the greedy algorithm will exactly return the optimal solution. Performance should be close to optimal if the distribution of the scenarios is close to pareto dominant, and further from optimal if the scenarios are very different from each other.

In many real world systems, such as the power grid, supplies and demands among the nodes often have similar relationships across different scenarios, e.g. a larger generator will always have greater output than a smaller generator, though their absolute outputs may vary. With this type of scenario distribution, the heuristic should be close to optimal. Interestingly, in our computational experiments in which scenarios were generated from a uniform distribution where scenarios are different from each other and not at all close to pareto dominance, when we expect the heuristic to perform poorly, and the heuristic still returns the optimal objective in most cases.

5 Conclusion

We have considered a robust network capacity design problem where there is uncertainty in the supplies and demands which is represented with a set of discrete scenarios. We review a constraint generation algorithm for this problem when all scenarios are required to have a feasible flow and present practical implementation details that empirically lead to improved algorithmic performance. We develop a novel algorithm to solve the chance-constrained problem when $\alpha\%$ of all scenarios are required to be feasible. This tree-based combinatorial algorithm embeds the previously described constraint generation algorithm to find the optimal arc capacity assignment. Additionally, a greedy algorithm is presented that can often be used to solve for a high quality heuristic solution to this chance-constrained problem.

6 Acknowledgements

This work was supported in part by an NSF Graduate Student Fellowship.

Sandia National Laboratories' Laboratory-Directed Research funded portions of this work. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

			IEEE30 Uniform		IEEE30 Scaled&Perturbed		exh20 Uniform	
Number Scenarios	α	l	Optimal Obj. Value	Heuristic Obj. Value	Optimal Obj. Value	Heuristic Obj. Value	Optimal Obj. Value	Heuristic Obj. Value
67	97%	2	8706	8706	2607	2607	1420	1420
100	97%	3	8030	8053	2864	2864	1704	1704
300	99 %	3	8674	8674	2867	2867	1360	1360
600	99.5 %	3	8950	8950	3621	3621	1475	1475

Table 2: αSCD Heuristic Evaluation

Number Scenarios	α	l	IEEE30 Uniform	IEEE30 Scaled&Perturbed	exh20 Uniform
67	97%	2	8	2	11
100	97%	3	16	6	21
300	99%	3	46	14	45
600	99.5%	3	72	28	110

Table 3: Heuristic Run Times (sec)

References

[1] S. Ahmed and A. Shapiro. Solving chance-constrained stochastic programs via sampling and integer programming. In Z.L. Chen and S. Raghavan, editors, *Tutorials in Operations Research*, pages 261–269. INFORMS, 2008.

[2] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows: theory, algorithms, and applications. 1993.

[3] A. Atamtúrk and M. Zhang. Two-stage robust network flow and design under demand uncertainty. *Oper. Res.*, 55(4):662–673, 2007.

[4] P. Beraldi and A. Ruszczyński. A branch and bound method for stochastic integer problems under probabilistic constraints. *Optim. Methods Software*, 17(3):359–382, 2002.

[5] R. Christie. UW power systems test archive. `http://www.ee.washington.edu/research/pstca/`, 1993.

[6] G. Dahl and M. Stoer. A cutting plane algorithm for multicommodity survivable network design problems. *INFORMS J. Comput.*, 10:1–11, 1998.

[7] D. Dentcheva, A. Prékopa, and A. Ruszczynski. Concavity and efficient points of discrete distributions in probabilistic programming. *Math. Programming*, 89(1):55–77, 2000.

[8] R.E. Gomory and T.C. Hu. An application of generalized linear programming to network flows. *J. SIAM*, 10(2):260–283, 1962.

[9] S. Küçükyavuz. On mixing sets arising in chance-constrained programming. *Math. Programming*, pages 1–26, 2009.

[10] J. Luedtke and S. Ahmed. A sample approximation approach for optimization with probabilistic constraints. *SIAM J. on Optim.*, 19(2):674–699, 2008.

[11] J. Luedtke, S. Ahmed, and G.L. Nemhauser. An integer programming approach for linear programs with probabilistic constraints. *Math. Programming*, 122(2):247–272, 2010.

[12] James Luedtke. A branch-and-cut decomposition algorithm for solving chance-constrained mathematical programs with finite support. *Mathematical Programming*, pages 1–26, 2013.

[13] M. Minoux. Optimum synthesis of a network with non-simultaneous multicommodity flow requirements. *Stud. Graphs Discrete Programming*, 11:269–277, 1981.

[14] M. Minoux. Network synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, 19(3):313–360, 1989.

[15] M. Minoux. Robust network optimization under polyhedral demand uncertainty is np-hard. *Discr. Applied Math.*, 158(5):597–603, 2010.

[16] S. Mudchanatongsuk, F. Ordónez, and J. Liu. Robust solutions for network design under transportation cost and demand uncertainty. *J. Oper. Res. Soc.*, 59(5):652–662, 2008.

[17] F. Ordóñez and J. Zhao. Robust capacity expansion of network flows. *Networks*, 50(2):136–145, 2007.

[18] Adam Ouorou. Robust capacity assignment in telecommunications. *Computational Management Science*, 3(4):285–305, 2006.

[19] Adam Ouorou. Tractable approximations to a robust capacity assignment model in telecommunications under demand uncertainty. *Computers & Operations Research*, 2012.

[20] Bernardo K Pagnoncelli, D Reich, and Marco C Campi. Risk-return trade-off with the scenario approach in practice: a case study in portfolio selection. *Journal of Optimization Theory and Applications*, 155(2):707–722, 2012.

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

[22] Georgios Petrou, Claude Lemaréchal, and Adam Ouorou. An approach to robust network design in telecommunications. *RAIRO-Operations Research*, 41(04):411–426, 2007.

[23] A. Prékopa. Dual method for the solution of a one-stage stochastic programming problem with random RHS obeying a discrete probability distribution. *Math. Methods Oper. Res.*, 34(6):441–461, 1990.

[24] A. Ruszczyński. Probabilistic programming with discrete distributions and precedence constrained knapsack polyhedra. *Math. Programming*, 93(2):195–215, 2002.

[25] Yongjia Song and James R Luedtke. Branch-and-cut approaches for chance-constrained formulations of reliable network design problems. *Mathematical Programming Computation*, 5(4):397–432, 2013.

[26] S. Sridhar and R. Chandrasekaran. Integer solution to synthesis of communication networks. *Math. Oper. Res.*, pages 581–585, 1992.

[27] J.P. Watson, R.J.B. Wets, and D.L. Woodruff. Scalable Heuristics for a Class of Chance-Constrained Stochastic Programs. *INFORMS J. Comput.*, 22(4):543–554, 2010.