Using mixed-integer programming to solve power grid blackout problems

Daniel Bienstock[1]
Dept. of IEOR
Columbia University
New York, NY 10027

Sara Mattia
DIS
Università "La Sapienza"
Roma

October 2005, revised August 2006

Abstract

We consider optimization problems related to the prevention of large-scale cascading blackouts in power transmission networks subject to multiple scenarios of externally caused damage. We present computation with networks with up to 600 nodes and 827 edges, and many thousands of damage scenarios.

[1] Partially funded by NSF award 0521741.

1 Motivation

During the last decade, several large-scale failures of national power transmission networks took place. The most recent were the blackouts of the U.S. Northeast and Eastern Canada [20] of August 2003, and the September 2003 blackout that affected Italy [23]. In addition, Brazil experienced large blackouts in 1999 (see [18]).

These blackouts affected large numbers of people over wide geographical areas, with substantial economic impact. Had the event lasted more than a few days, the human cost would have been quite large.

The task reducing the likelihood of catastrophic blackouts in a large network is complex, involving engineering, economic and even political issues. However, a reading of any of the recent studies (we recommend [20] for an excellent in-depth analysis) makes it clear that, at the core of the problem, there are significant combinatorial difficulties that need to be addressed.

In this paper we present two different models and algorithmic tools to address this problem.

1.1 A brief overview of transmission networks

For general background on power networks we refer the reader to [2]. Also see [7]-[9]. Broadly speaking, a power grid is made up of three components: generation, transmission and distribution. At one end of the grid there are the generators (power units) that produce power at relatively high voltage. At the other end is consumption, primarily in metropolitan areas. There, power is conveyed at fairly low voltages by means of (relatively) simple sub-networks known as distribution networks. Between generation and consumption lies the transmission network, whose purpose it to convey power from one to the other. Transmission networks operate at fairly high voltages (for efficiently); both generators and distribution networks are connected to transmission networks by means of transformers.

For a number of economic and political reasons, modern transmission networks are large and complex, spanning great distances and conveying power from many generators to many metropolitan areas located far away. The recent blackout events were due to failures of transmission networks.

The reader familiar with e.g. telecommunication networks may expect that one can control how power flows in a network. In fact, this is actually not true – power flows according to the laws of physics and one can only indirectly influence this flow.

Power flows are usually studied using the so-called *AC flow* model. (For convenience we will usually use the standard node, edge graph-theoretic terminology, although we will sometimes use the term "line" to refer to an edge). In this model, the voltage at a node k of the network is represented by a complex number, $U_k\, e^{j\theta_k}$, where $j = \sqrt{-1}$ and θ_k is the *angle* at k. The power flowing from k to q along the (undirected) edge $\{k, q\}$ depends on known parameters g_{kq}, b_{kq}, b^{sh}_{kq} and is expressed as $p_{kq} + jq_{kq}$, where

$$\begin{aligned} p_{kq} &= U_k^2\, g_{kq} - U_k\, U_q\, g_{kq} \cos\theta_{kq} - U_k\, U_q\, b_{kq} \sin\theta_{kq} && (1)\\ q_{kq} &= -U_k^2\, (b_{kq} + b^{sh}_{kq}) + U_k\, U_q\, b_{kq} \cos\theta_{kq} - U_k\, U_q\, g_{kq} \sin\theta_{kq} && (2)\\ \theta_{kq} &\doteq \theta_k - \theta_q. && (3) \end{aligned}$$

The quantity p_{kq} is called the *active* power flow, q_{kq} is the *reactive* power flow. Both quantities have concrete physical interpretations, and can take negative values. Note that this model permits that e.g. $p_{qk} \neq -p_{kq}$. At a node k of the network, the net power injected into the network at k is (approximately) given by the complex number $P_k + jQ_k$, where

$$\begin{aligned} P_k &= \sum_{kq} p_{kq} && \text{(net active power leaving } k) && (4)\\ Q_k &= \sum_{kq} q_{kq} && \text{(net reactive power leaving k).} && (5) \end{aligned}$$

These are standard network flow conservation constraints – we stress that in both of them there is a term for each edge incident with node k. If k is a generator node, then $P_k \geq 0$; in general at a generator node there will be a constraint of the form

$$P_k^{min} \leq P_k \leq P_k^{max} \qquad (6)$$

and similar bounds for the reactive power at k. If k is a *load* (demand) node, $P_k < 0$; at any point in time this represents the negative of the demand at k. If k is neither a generator nor a demand node, then $P_k = Q_k = 0$. For a more thorough treatment, see [23].

The model given by constraints (1)-(5) provides a fairly accurate approximation of the steady-state behavior of a power grid. Nevertheless, it suffers from two shortcomings: first, it can be expensive to solve, and second, the system may have multiple solutions (the solution set may be discrete; less frequently, the system may even be infeasible). Partly in order to remedy the second difficulty, the most popular approaches to computing AC power flows rely on iterative methods, which require an initial "guess" of the solution. Such a guess is relatively easy to arrive at when one is familiar with the network being solved but not so if the network is in an unusual configuration; an incorrect guess can lead to convergence to the "wrong" solution. Human input in this loop is frequently used.

In order to bypass these shortcomings, primarily the speed issue, a linear model is frequently used. This is the so-called DC flow model, which relies on some estimations, primarily that $\theta_{kq} \approx 0$ for each edge $\{k, q\}$ and $U_k \approx 1$ for any node k. The (approximate) active power constraint (1) becomes

$$x_{kq}\, p_{kq} - \theta_k + \theta_q = 0 \qquad \text{for all } \{k, q\}. \qquad (7)$$

where $x_{kq} \doteq -\frac{1}{b_{kq}}$ is the *series reactance* (reactance for short in this paper – it is perhaps unfortunate that the x notation is standard in denoting this parameter). Note that because of (7), we have $p_{qk} = -p_{kq}$ for each edge $\{k, q\}$, in other words the two equations (7) corresponding to $\{k, q\}$ are equivalent. (Alternatively, we can view the network as directed, and use a negative flow value to indicate flow in the direction of the reversed edge. In Section 2 we will use the directed edge notation because it is more convenient in the context of the linearized power flow model, whereas in Section 3 we will use the undirected edge notation). The system made up of equations (7), together with (4), (6) and a fixed value $P_k < 0$ at each demand node k constitutes the DC power flow model. For completeness, we state the following result:

Lemma 1.1 *Suppose we have a connected network with node set V. Then for each choice of values P_k (for each $k \in V$) with $\sum_k P_k = 0$, the system made up of all equations (4) and (7) has a unique solution in the variables p_{kq}.*

It remains a matter for research precisely when the DC flow model provides a good approximation for the AC model. A more salient question is when the DC model provides an approximation that is good enough from the point of view of network design. In any case, the DC model is *vastly* popular in the electric power literature, being the model of choice any time that many power flow computations are needed.

When analyzing a power network, there is an additional, critical, operational requirement. For each edge k, q there is a "capacity" u_{kq}, representing a thermal limit. In the DC flow model, we should have $p_{kq} \leq |u_{kq}|$. This (or the appropriate statement in the AC flow model) is not a constraint that enters into the solution procedure – the power flow values are determined by the physics of the network, whereas the capacity constraint is simply a desirable outcome. Should an edge exceed its capacity, then eventually it will burn up (how long this takes depends on the overload) but normally protection equipment will disconnect the edge when the failure point is approached. We stress that a small overload is tolerable and that the protection equipment will not act immediately in such a case. Note: we will use the term "capacity" because of its familiar interpretation in optimization.

1.2 How large-scale blackouts occur

A cascade in a power grid begins with an externally caused event, such as a fire, or lightning, or even repair work, that causes a power line or lines to be disabled. From a graph-theoretic standpoint, we now have a new network, and the physics of the situation will determine the resulting power flows, which take hold *immediately* (or more precisely, at the speed of light) and may be quite different from the initial flows.

The new power flows may exceed the capacities of some edges. Thus, after some time (possibly minutes) unless appropriate action is taken the more severely overloaded edges will be turned off.

From a graph-theoretic standpoint, this is no different from the initial external event: now we have an even smaller network, with new power flows, which may (again) exceed the capacities of some edges.

Essentially, at this point we have a cycle. In an unlucky situation, this cycle may gather speed and 'snowball' or 'cascade' catastrophically. This can cause a large fraction of the demand to become unserviceable, and in an extreme situation may damage generators. In order to avoid this, the cascade is terminated as soon as it is deemed unrecoverable, and usually this is done by disconnecting much of the demands. This is a blackout. Recovery is, as turning the network 'back on' will simply cause the cascade to resume.

A number of points are useful here. First, the transient effects that occur when the network moves from one set of power flows to the next are not considered to be the cause of edge failures during a cascade. Rather, it is the thermal (edge overload) effects that are responsible, and these take effect over periods of minutes (as opposed to the transient effects, which last for fractions of a second).

Second, in a large power grid, external events that disable some power lines are not uncommon, yet seldom cause a cascade. Nevertheless, the probability that a given line will be disabled by some event (lightning, etc.) is quite small. And the probability that a cascade will occur is extremely small (large-scale blackouts are very rare!). For this reason, in this paper we will take a "worst-case" approach. We will expand on this later.

The prevalent view in the power community is that, once a "cascade" has started, it would be impossible to run a centralized algorithm to attempt to control the cascade (too much information changing too quickly). Whether this is strictly true or not, it clearly makes sense to try to react sooner than later, and to try to design networks so that catastrophic cascades are less likely to occur to begin with. This is the approach followed here.

1.3 This paper

In this paper we consider two optimization models to address the following generic question: given a network, how do we protect it at minimum cost so as to make it (more likely to) survive a potential cascade? In specifying such a model, we need to make three elements concrete: (a) the externally caused damage that may trigger a cascade, (b) how we invest in the network, and (c) what we mean by "survive".

(a) We will assume that we are given a list of *contingencies*, or *scenarios.* Each scenario σ consists of a set of edges $S(\sigma)$ of the network: were scenario σ to be realized, all edges in $S(\sigma)$ are disabled. We will further take a *worst-case* approach: when we protect the network we want it to survive *every* scenario. We further comment on this below.

(b) There are many possible (and reasonable) models on how to invest on a network in order to make it more resilient. For example, we might add edges (in particular, parallel edges). Or, we might upgrade an edge $\{k, q\}$, at a cost, in order to change its parameter x_{kq} (see eq. (7). Or, by investing on any given edge, we might make it immune from the type of external damage being considered.

The purpose of this paper is to demonstrate how a body of techniques (in particular, Benders' decomposition) can prove effective for solving problems of this type. For the sake of conciseness, we consider one particular model:

We will assume that the capacity of any edge $\{k, q\}$ can be increased from u_{kq} to a higher value u^{new}_{kq} by paying a certain cost w_{kq} (possibly $+\infty$), while keeping x_{kq} fixed.

It is important to note that this particular model may (or may not) be a completely accurate representation of what may be feasible in practice – but we pick it, nonetheless, because of its simplicity. What is important here are not the low-level details of the model, but the overall algorithmic approach that we take, which can be adapated in a straightforward fashion to more detailed models.

(c) There remains to specify when we say that a network has survived in a scenario.

In the first model we consider, we insist that, for each scenario σ, the flows p^{σ} in the residual network – the network obtained by deleting the set $S(\sigma)$– satisfy the capacity constraints. That is to say, for each $\{k, q\} \notin S(\sigma)$ we have $p^{\sigma}_{kq} \le |\hat{u}_{kq}|$, where $\hat{u}_{kq} = u^{new}_{kq}$ if $\{k, q\}$ was upgraded, and $\hat{u}_{kq} = u_{kq}$ otherwise. A complete mixed-integer programming formulation, using the DC flow model, is given in Section 2.

This is a fairly conservative model (it requires each potential cascade to be immediately stopped). On the positive side, this model admits a traditional mixed-integer programming formulation, and we are able to solve problems on networks with several hundred edges and with thousands of scenarios in practicable time.

The second model we consider, studied in Section 3, is more flexible, and takes into account the dynamics of the evolution of a cascade. We model the cascade as proceeding in discrete "rounds," building on the models in [15], [11]; this formalizes the ideas in Section 1.2. In each round, a new set of edges is removed from the network. Instead of asking that each cascade be stopped immediately, we allow multiple rounds of edge removals to occur – essentially, we just ask that the cascade "eventually" stop. We also allow a small fraction of the overall demand to be "lost" (for example, when nodes become disconnected from one another). As in the first model, we want to upgrade a minimum-cost set of edges so as to ensure that the network survives every scenario. A more precise definition of this model will be given in Section 3.

The algorithm we develop for this problem does not require the DC flow model – any flow model can be handled. For the sake of expediency, all experiments in this paper use the DC flow model. Extending our experiments to AC flow models, and so as to model *voltage collapse* (see [20] for some background) will be a venue for future research

1.3.1 Scenario generation and the adversarial problem

In this paper we focus on optimization problems while assuming that a set of scenarios of 'interest' have been previously generated as an input to our algorithms.

An alternative approach would be to employ stochastic programming. This is attractive because stochastic programming techniques are able to efficiently handle optimization problems with large numbers of scenarios. In order to use stochastic programming scenarios would be assigned probabilities; and other changes to the model would be needed as well: we would likely need to model a 'cost' of each scenario using a linear function, and the multi-round problem in 3 might need to be collapsed into one or two rounds. The output of a stochastic programming algorithm would be a tight *confidence interval* for the *expected* total cost (cost of investments plus scenario cost).

A fundamental difficulty with the stochastic programming approach is precisely how we would model the scenario probabilities, especially since if the model were accurate the set of scenarios that correspond to blackouts should have extremely small measure. A possible approach involves the concept of *importance sampling* [17]. This methodology still does require a fairly precise model of the scenario probabilities, particularly those of the "interesting" scenarios, which might be problematic. A probabilistic model of cascades is given in [16].

In general, a criticism that can be leveled at an approach based on assigning probabilities to scenarios is that we end up protecting against what appear to be the more likely events, according to a possibly idiosyncratic probabilistic model, while remaining exposed to other, possibly just as significant, events. From the point of view of genuine robustness, a worst-case approach may be better. Of course, when we have an astronomically large number of scenarios the worst-case approach becomes impractical. Consequently, one would need to limit the set of scenarios to some manageable collection (e.g. all subsets with at most 5 edges). It is true that by doing this we are also remaining exposed to events outside the scope of the model, but at least there is an explicit understanding of what we are protecting against. One possible future research venue would be to use a sophisticated probabilistic model to generate a list of (say) the $10,000$ most likely scenarios; and then handle these scenarios in a worst-case fashion using the techniques of this paper. This does not mitigate the issues described above, but simply provides a reasonable way of generating interesting scenarios. It may also be possible that stochastic programming could be adapted to work together with the algorithms we develop in this paper, as a sampling heuristic.

There is a third, alternative approach to the problem, which is interesting on its own. This is the *adversarial* outlook. Given a fixed network, what is the minimum number of edges that an adversary has to delete in order for a catastrophic cascade to occur? There are many variants to this question, including some where the adversary is deleting edges one at a time, and some that allow for reaction, i.e. gaming. The adversarial problem would systematically discover weaknesses of the network, and could be used to generate interesting scenarios both for the worst-case approach we use here and for the stochastic programming approach. It can be shown that the adversarial problem is NP-hard, and (using the DC flow model) it can be formulated as a (large) mixed-integer program.

1.4 Prior work

In [21], a network reinforcement problem is considered, where as above there is a fixed set of scenarios and in each scenario a subset of edges is deleted. The objective is to add to the network a minimum-cost set of power lines (edges), so that in each scenario the power flow in every edge is within its capacity. This is similar to the model we study in Section 2. Using the DC flow model, [21] formulates this problem as a mixed-integer program, with explicit flow and angle variables for each scenario, and 0/1 variables to model edge additions. The model is then solved using commercial mixed-integer programming software. Some heuristics are discussed to handle large cases – the explicit formulation will probably become difficult to use when the network is large and, especially, when the number of scenarios is large. Also see [18].

Several interesting models for and analyses of cascading failures are presented in [11], [14], [15], [16] and other publications by the same group of authors. These models provide a step-by-step recipe for simulating cascades. We will expand on this in Section 3.

In [12], a network design problem without failure scenarios is studied. We are given a network and a set of candidate edges C that can be added to the network, each at a certain cost. In addition, for each demand node k we have a penalty r_k; this is the per-unit cost of demand not met at k. The problem is to find a set of candidate edges so that the total cost (cost of adding edges plus cost of unmet demand) is minimized. If C has been appropriately chosen and if the penalties are large enough, then in the optimal solution all demand will be met. [12] models the problem as a mixed-integer program with 0/1 variables corresponding to the edge additions. The problem is tackled using Benders' decomposition; the "slave" subproblem is that of finding a set of power flows, in a fixed network, that minimizes the cost of unmet demand. [12] also discusses using Gomory cuts in the master problem. A problem on 46 nodes, 66 edges and 237 candidate edges (some parallel) is solved in a few hours of CPU time.

2 The first model

In this section we present the first approach to the problem. Given a network, we want to decide on which lines we need to increase capacity in order to guarantee that in each scenario σ of a given family, all flows are within bounds (capacities) after removing the edges of the set $S(\sigma)$ from the network.

We will first describe a *natural* MIP formulation of the problem (see also [21] and [12]). This formulation can be quite large and impractical when the number of scenarios is large. For this reason we present a (compact) *projected* formulation and an algorithm for solving the problem via this projected formulation.

Our problem can be formally stated as follows. We are given a directed graph $G(V, E)$ representing the network and a set Σ of scenarios. As discussed in Section 1.1, the nodes of the graph are partitioned into three classes: nodes corresponding to power plants (generators, supply nodes), nodes corresponding to loads (demand nodes), and transmission nodes. Let $\mathcal{D} \subseteq V$ be the set of the demand nodes and let $\mathcal{O} \subseteq V$ be the set or the supply nodes. For every node $i \in \mathcal{D}$, there is a demand $D_i > 0$ for power and for every node $j \in \mathcal{O}$ there is a minimum and a maximum amount of power $P_j^{min}, P_j^{max} > 0$ that can be provided by that power plant. In the rest of the section we assume that $P_j^{min} = 0$, although the algorithm can be modified to handle the case where $P_j^{min} > 0$. A more comprehensive model would be one where (in addition to having $P_j^{min} > 0$) the decision maker can *turn off* generator j; this is related to so-called *unit commitment* problems [22]. Such a model cannot be handled, directly, using the machinery we present below – however, we note that such a model handles detailed, short-term, reactive decisions (which generators to turn off in the event of a fault) whereas in this paper we focus on long-term decisions only.

The edges of the graph represent the transmission lines. For every edge $(i, j) \in E$ we are given a value x_{ij} representing the physical parameter (reactance) of the line, and a capacity value u_{ij} which is the maximum amount of power that can safely use the line without burning it. We are also given capacity expansion costs w_{ij}. The capacity cannot be added in arbitrary continuous amounts: it is only possible, by paying the cost w_{ij}, to increase capacity by c_{ij} units, obtaining a total capacity of

$$u_{ij}^{new} = u_{ij} + c_{ij} \tag{8}$$

for the edge.

For every scenario $\sigma \in \Sigma$ a list $S(\sigma)$ of edges is given: this list consists of the edges which are supposed to fail and are disabled in that scenario. We restrict our attention to the case of edge failures, but both the models and the algorithms we describe can be easily extended to the case of node failures, node and edge failures, also when quality of service criteria are given and the demands D_i are scenario-dependent.

2.1 The Natural Formulation

Using the DC power flow model, the state of the system in each fault scenario σ can be completely described by the values of the angles at the nodes and by the power flows on the edges. They are related to each other by linear equations. Let p^σ_{ij} be the power flowing on edge $(i,j) \in E$ and let θ^σ_i be the angle at node $i \in V$, both in scenario σ. The equations describing the behavior of the network are (7) and (4), restated for scenario σ. A natural MIP formulation of the problem is the following:

$$\begin{array}{llll}
\textbf{CAP:} & \min & \sum_{(i,j)\in E} w_{ij} y_{ij} & \\
& \text{s.t.} & \theta^\sigma_i - \theta^\sigma_j = x_{ij} p^\sigma_{ij} \qquad (i,j) \in E - S(\sigma), \sigma \in \Sigma & (9) \\
& & \sum_{(i,j)\in\delta^+(i)} p^\sigma_{ij} - \sum_{(j,i)\in\delta^-(i)} p^\sigma_{ji} = \begin{cases} P^\sigma_i & i \in \mathcal{O} \\ -D_i & i \in \mathcal{D} \\ 0 & i \in V - (\mathcal{D} \cup \mathcal{O}) \end{cases} \qquad \sigma \in \Sigma & (10) \\
& & 0 \le P^\sigma_i \le P^{max}_i \quad \forall\, i \in \mathcal{O},\, \sigma \in \Sigma & (11) \\
& & -(u_{ij} + c_{ij} y_{ij}) \le p^\sigma_{ij} \le u_{ij} + c_{ij} y_{ij} \qquad (i,j) \in E - S(\sigma), \sigma \in \Sigma & (12) \\
& & y \in \{0,1\}^{|E|} &
\end{array}$$

In this formulation, the y_{ij} are the decision variables: if $y_{ij} = 1$ the capacity on edge (i,j) is $u_{ij} + c_{ij}$, otherwise the edge has its original capacity u_{ij}. Corresponding to scenario σ, the p^σ are the power flows and the θ^σ are the angles. Both powers and angles are free variables. As previously explained, a negative value of p^σ_{ij} means that the power is flowing on edge (i,j) from j to i. Constraints (9) correspond to (7), constraints (10) are flow balance constraints corresponding to (4) (and $\delta^+(i)$ denotes the set of edges leaving node i; similarly with $\delta^-(i)$). For each generator node i and scenario σ, variable P^σ_i indicates the output generated by i, bounded by constraints (11). Constraints (12) are variable upper and lower bound constraints for the flows.

This formulation is very similar to that of a standard capacitated network flow problem [1]. However, constraints (9) introduce a significant complexity . These constraints highlight the fact that in power networks we cannot completely control (or predict) how power flows in the network.

2.1.1 Some examples

In this section we present three examples that show how constraints (9) heavily affect the structure of the problem and make optimization difficult.

In the first example we show how constraints (9) can cut-off solutions which would otherwise have been feasible (for the standard flow problem). Consider the network of figure 1. It has three loads 1 (demand 10), 6 (demand 50) and 7 (demand 50). Nodes 2 and 3 are the generators, with $P^{max}_2 = 75$ and $P^{max}_3 = 35$. The labels on the edges are the original capacities. Suppose $x_{ij} = 1$ for all edges (i,j).

Suppose we want to check whether the network is feasible using the given capacities, that is, if it is possible to route all the demands using the existing capacities. Suppose we ignore constraints (9) – then the problem reduces to a standard network flow problem. A feasible solution exists and it is the following:

$$p_{21} = 5, \quad p_{31} = 5, \quad p_{24} = 70, \quad p_{35} = 30, \quad p_{45} = 20, \quad p_{46} = 50, \quad p_{57} = 50.$$

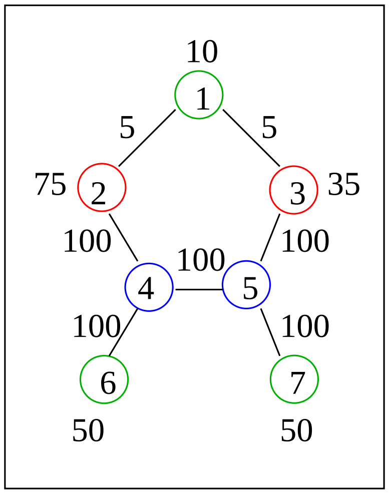

Figure 1: an infeasible network

Let's now add constraints (9) back to the problem. Because of the edge capacity constraints, the only feasible way to serve node 1 is to set $p_{21} = p_{31} = 5$, which completely uses up the capacity of $(2, 1)$ and $(3, 1)$, and thus only those flows with destination 5 can use either of those edges. Furthermore, after serving node 5, the combined (residual) supply of nodes 2 and 3 equals 100 which is also the combined demand of nodes 6 and 7 – as a result, $p_{35} = 30$ and $p_{24} = 70$. Further, since $x_{21} = x_{31} = 1$, constraints (9) impose that $\theta_2 = \theta_3$. But now the fact that $p_{24} > p_{35}$ implies that $\theta_4 < \theta_5$, and so $p_{54} > 0$, and finally $p_{57} < 30 < 50$: therefore node 7 cannot be served completely and the problem is infeasible.

In the second example we see how the introduction of a new edge can have a negative impact on the network due to constraints (9). Consider now the network in figure 2. It has two loads, nodes 4 (demand 20) and 5 (demand 10) and one generator, node 1, with $P_1^{max} = 30$. The labels on the edges are the original capacities. Let $x_{ij} = 1$ for all the edges.

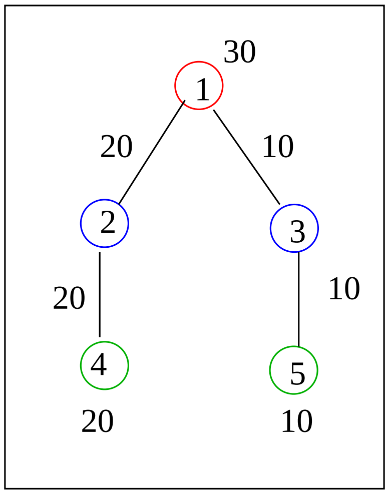

Figure 2: a feasible network

A flow that satisfies all constraints (including (9)) is:

$$p_{12} = 20, \quad p_{13} = 10, \quad p_{24} = 20, \quad p_{35} = 10.$$

Suppose now to add the new edge $(2, 3)$, with capacity 30 and $x_{23} = 1$ (figure 3). In a standard network flow problem, the presence of a new edge would have no effect. However, constraints (9) have a remarkable effect: adding the new edge renders the problem infeasible, as we will argue.

Because of the capacity constraints, in order to serve node 4 we need to have flow $p_{12} = 20$, and to serve node 5 we need $p_{13} = 10$. Since $x_{12} = x_{13}$ and $p_{12} > p_{13}$, then by (9) we have that $\theta_3 > \theta_2$ and therefore $p_{32} > 0$. But then by flow conservation we have that $p_{35} < 10$, in other words the demand at 5 is not fully served. This example may be viewed as an analogue of “Braess' law” [13].

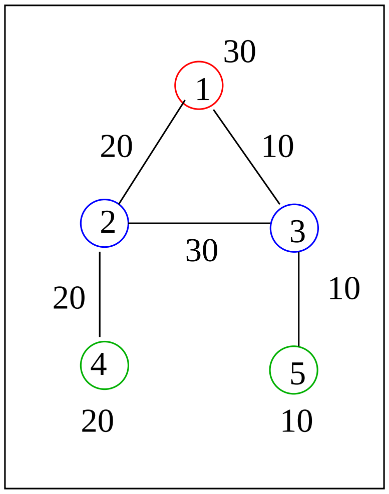

Figure 3: an infeasible network again

The previous two examples highlighted how constraint (9) adds significant complexity to the problem. In the third example we show how constraints (9) impact our optimization problem CAP by preventing the use a simple notion of *dominance* between scenarios.

One difficulty associated with CAP is the potentially large number of scenarios to consider. In particular, given a vector $y \in \{0,1\}^{|E|}$, simply checking feasibility of y requires solving $|\Sigma|$ linear programs. Thus, ideally we would have a simple dominance criterion: given scenarios τ and ρ, if τ “dominates” ρ then we only need to check scenario τ. Ideally, such a dominance criterion should not depend on the particular vector y being checked; a “natural” candidate for such a dominance definition would be to say that τ dominates ρ if $S(\rho) \subseteq S(\tau)$.

However, this particular criterion is not valid. Consider the network of figure 4. It has two loads: node 2, with demand 20, and node 3, with demand 10. There are two generators: node 1 and node 4 with $P_1^{max} = P_4^{max} = 30$. The original capacities are the labels on the edges in the figure. Let $x_{ij} = 1$ for all the edges and let $c_{ij} = u_{ij}$. Suppose we have two scenarios: σ_1, with $S(\sigma_1) = \{(2,3),(2,4),(3,4)\}$, and σ_2 with $S(\sigma_2) = \{(2,4),(3,4)\}$.

The vector y with $y_{13} = 1$ and $y_{ij} = 0$ for all $(i,j) \neq (1,3)$ is feasible for scenario σ_1: by increasing the capacity on edge $(1,3)$ from $u_{13} = 5$ to $u_{13}^{new} = u_{13} + c_{13} = 10$, it is possible to serve node 3 from node 1. We can also serve node 2 from node 1 and it can be checked that this solution satisfies constraints (9). On the other hand, there are no feasible solutions for scenario σ_2. To see that this is the case, consider the network in scenario σ_2, and write $p_{12} = 20 + f$ (possibly $f < 0$). Then $p_{23} = f$, and therefore $p_{13} = 10 - f$. Since $u_{13}^{new} = 10$, it follows that $f \geq 0$. Therefore $\theta_2 \geq \theta_3$ (by (9)), and as a result $20 + f \leq 10 - f$, from which we get $f \leq -5 < 0$, a contradiction.

2.2 The Projected Formulation

The natural formulation described above has $| \Sigma | (| E | + | V | + | \mathcal{O} |)$ variables and $| \Sigma | (3 | E | + | V | +2 | \mathcal{O} |)$ constraints and it can become unwieldy for large instances, i.e. instances involving large networks or a large number of scenarios. In this section we present a projected formulation which only has $| E |$ variables at the cost of potentially a large number of constraints. We use linear programming duality to generate valid inequalities for the projection of the natural formulation to the space of the y variables. Thus, we are essentially using a Benders' decomposition algorithm [6] (properly stated, we are only using the “infeasible subproblem” case

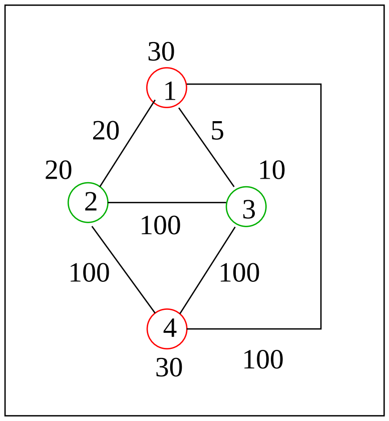

Figure 4: an example with scenarios

of Benders). Similar algorithms have been used in the context of network design in telecommunications, see [10], [4], [3] and references therein.

Let σ be a scenario and let $\bar{y}$ be a possibly fractional capacity assignment vector. Consider the following linear program:

$$
\begin{array}{llll}
\mathrm{FEAS}(\bar{y},\sigma): \quad \mu^* \doteq & \max & \mu & \qquad (13) \\
& (f_{ij}^\sigma) & \theta_i^\sigma - \theta_j^\sigma = x_{ij} p_{ij}^\sigma & (i,j) \in E - S(\sigma) \\
& (z_i^\sigma) & \displaystyle\sum_{(i,j)\in\delta^+(i)} p_{ij}^\sigma - \sum_{(j,i)\in\delta^-(i)} p_{ji}^\sigma = \begin{cases} P_i^\sigma & i \in \mathcal{O} \\ -\mu D_i & i \in \mathcal{D} \\ 0 & i \in V - (\mathcal{D} \cup \mathcal{O}) \end{cases} & \\
& (r_{ij}^\sigma) & 0 \le P_i^\sigma \le P_i^{max} \quad \in \mathcal{O} & \\
& (v_{ij}^\sigma) & p_{ij}^\sigma \le u_{ij} + c_{ij}\bar{y}_{ij} \quad (i,j) \in E - S(\sigma) & \\
& (t_{ij}^\sigma) & p_{ij}^\sigma \ge -(u_{ij} + c_{ij}\bar{y}_{ij}) \quad (i,j) \in E - S(\sigma) &
\end{array}
$$

In this formulation, μ is the fraction of the demand that can be served according to the capacities fixed by $\bar{y}$; we have also indicated the dual variable names next to each corresponding constraint. It is easy to see that if $\mu^* < 1$ then $\bar{y}$ is not a feasible capacity assignment vector for scenario σ, and that if $\mu^* > 1$, then by appropriately scaling the optimal solution we obtain a feasible solution with $\mu = 1$.

Using linear programming duality, the necessary and sufficient condition $\mu^* \ge 1$ becomes the constraint

$$
\sum_{(i,j)\in E-S(\sigma)} (u_{ij} + c_{ij} y_{ij})(v_{ij}^\sigma + t_{ij}^\sigma) + \sum_{i\in\mathcal{O}} P_i^{max} r_i^\sigma \quad \ge \quad 1, \qquad (14)
$$

for every vector $(f^\sigma, z^\sigma, r^\sigma, v^\sigma, t^\sigma)$ feasible for the dual of FEAS$(\bar{y},\sigma)$. If we solve FEAS$(\bar{y},\sigma)$, and $\mu^* < 1$, then the optimal dual variables can be used to generate a violated inequality (14).

Note that if a dual solution has $f_{ij}^\sigma = 0$, then we are essentially ignoring constraints (9) in the primal problem, in which case the primal problem reduces to a standard single-commodity capacitated network design problem. For such problems, necessary and sufficient conditions for feasibility are known: these are given by the well-known *cut inequalities* ([3], [4], [10]). Thus, separation over inequalities (14) includes separation over the cut inequalities.

Using inequalities (14), we can write a projected formulation for problem **CAP**:

$$\begin{array}{lll} \min & \sum_{(i,j)\in E} w_{ij} y_{ij} & \\ \text{s.t.} & \text{constraints (14)} & \text{for all } \sigma \in \Sigma \\ & y \in \{0,1\}^{|E|} & \end{array}$$

2.3 The Solution Method

We propose two different approaches to solve the problem via the projected formulation: a Branch & Cut approach and a heuristic approach. In the next two sections we describe the two approaches and finally we discuss computational results.

2.3.1 The Branch & Cut Approach

The projected formulation (15) has an exponential number of constraints; we solve it via Branch & Cut. We use a working formulation $Ay \geq b$ which initially consists of the bounds $0 \leq y_{ij} \leq 1$; at each iteration we solve the linear program $\min\{w^T y : Ay \geq b\}$ with solution $\bar{y}$. Next we solve FEAS$(\bar{y}, \sigma)$ to detect a violated inequality (14) for some $\sigma \in \Sigma$, if such an inequality is found then we add it to $Ay \geq b$. If not, and $\bar{y}$ is integral, then we are done, and if $\bar{y}$ is fractional, then we branch. The inequalities (14) are post-processed in two ways: first, by scaling and rounding procedure for numerical stability and second, by running a heuristic to find violated cover inequalities, i.e. inequalities of the form $\sum_C y_{ij} \geq 1$, and generalizations, and "rank-2" inequalities $\sum_C y_{ij} \geq 2$.

Procedure 2.1 *BRANCH & CUT ALGORITHM*

Initialize: $Ay \geq b$ consists of the bounds $0 \leq y_{ij} \leq 1$ for all (i,j).

```
1. Let $\bar{y}$ be the solution to the problem $\min\{w^T y : Ay \geq b\}$.

2. If, for some scenario $\sigma$, the solution to FEAS($\bar{y}$,$\sigma$) satisfies $\mu^* < 1$, then
   2.1. Let $\beta^T y \geq \beta_0$ be the corresponding cut (14) violated by $\bar{y}$,
   2.2. Scale and round $\beta^T y \geq \beta_0$ obtaining $\gamma^T y \geq \gamma_0$, which is added to $Ay \geq b$,
   2.3. Attempt to find a violated cover or rank-2 inequality $\rho^T y \geq \rho_0$,
        which is also added to $Ay \geq b$.
   2.4. Go to 1.

3. Otherwise, if $\bar{y}$ is integral, EXIT ($\bar{y}$ is optimal for CAP).
   If not, branch, and continue the procedure at each node of the
   Branch & Cut tree.
```

The algorithm enumerates scenarios in round-robin fashion, e.g. in a fixed cyclical order, and in Step 2 we select as the first scenario to check at the current iteration the scenario which produced $\mu^* < 1$ in the previous iteration.

Our scaling and rounding is as follows: we divide all the coefficients and the right-hand-side of a constraint by the smallest positive coefficient, and then round up to the nearest integer.

The final ingredient in the Branch & Cut algorithm is the separation of cover inequalities [NW88]. Given an inequality $\gamma^T y \geq \gamma_0$, we first complement variables as necessary so as to obtain nonnegative left-hand side coefficients. To simplify notation, let us still write the resulting constraint $\gamma^T y \geq \gamma_0$. We next use a simple knapsack heuristic to find a (violated) cover

$$\sum_{(i,j)\in C} y_{ij} \geq 1, \tag{15}$$

(i.e., $\sum_{(i,j)\notin C} \gamma_{ij} < \gamma_0$ and $\sum_{(i,j)\in C} \bar{y}_{ij} < 1$). In addition, if

$$\hat{\gamma} = \max_{(i,j)\in C} \gamma_{ij},$$

and, for some $(k,q) \notin C$

$$\sum_{(i,j)\notin C} \gamma_{ij} - \gamma_{kq} + \hat{\gamma} \, < \, \gamma_0,$$

then

$$\sum_{(i,j)\, \in \, C \cup (kq)} y_{ij} \geq 2, \tag{16}$$

is valid and implies (15). (16) is our "rank-2" inequality. The separation of these inequalities carries essentially zero computational cost.

2.3.2 Heuristics

We have developed two simple heuristics for problem **CAP**. The first one is a rounding heuristic that is periodically run during the course of the Branch & Cut search algorithm.

First heuristic The first heuristic uses a parameter, τ. When running the heuristic, we use the current (fractional) solution that has been computed by the Branch & Cut algorithm, $\bar{y}$. Then we construct the 0/1 vector $\hat{y}$ as follows: $\hat{y}_{ij} = 1$ iff $\bar{y}_{ij} > \tau$. Next, we check $\hat{y}$ against *each* scenario σ. If $\hat{y}$ is not feasible for σ then the (valid) constraint

$$\sum_{(i,j)\in F^0} y_{ij} \geq 1$$

is added to the formulation, where $F^0 = \{(i,j) \, : \, \hat{y}_{ij} = 0\}$. And if $\hat{y}$ is feasible for every scenario σ, then we have computed a new upper bound for the overall problem **CAP**.

The heuristic is expensive because of the need to check every scenario, and consequently it is run only sporadically. In our implementation, we set $\tau = 0.2$. We note that the value of the heuristic lies not just in computing upper bounds, but in the valid inequalities that it discovers.

Stand-alone heuristic. On large problems the pure Branch & Cut approach can prove time consuming. For this reason we have developed a stand-alone heuristic that allows us to generate good solutions in a reasonable amount of time and appears effective.

Because of constraint (9), we have found it difficult to produce effective heuristics that are purely combinatorial. Our heuristic builds a solution vector y by sequentially solving each problem **CAP** restricted to each scenario. Formally, let $\sigma \in \Sigma$ and a let F be a subset of E. Denote by $P(\sigma, F)$ problem **CAP**, restricted to scenario σ and where in addition we fix $y_{ij} = 1$ for all $(i,j) \in F$.

Procedure 2.2 *STAND-ALONE HEURISTIC*

```
Initialize: $F = \emptyset$
For $\sigma \in \Sigma$ do
        if $P(\sigma, F)$ is not feasible then
                exit : the problem is not feasible
        else let $\bar{y}$ be the solution of $P(\sigma, F)$, and set $F = F \cup \{(i,j) : \bar{y}_{ij} = 1\}$
Output: the solution $y$ having $y_{ij} = 1$ for $(i,j) \in F$ and $y_{ij} = 0$ for $(i,j) \in E - F$
```

The single-scenario problems solved by the heuristic are handled using the Branch & Cut machinery described above. On extremely large problem instances this heuristic could prove effective, for example by running it repeatedly using different orderings of the scenarios. An alternative would be to combine it with a *pruning* algorithm such as the one we will describe in Section 3.2.

2.4 Computational Results

In this section we report on the computational experience with th e Branch & Cut algorithm and with the stand-alone heuristic approach. The computations reported in this section were performed on a machine having a 1.6GHz Pentium M processor and 512MB RAM. We used Cplex 9 [19] as the LP solver and for the Branch & Cut framework.

The test bed is organized as follows. We use three networks: $net1$, $net2$ and $net3$. Networks $net1$ and $net2$, both with 300 nodes and 409 edges, are copies of one of the IEEE "test" cases (available from [24]) simplified and modified to suit the purposes of this paper (in particular, we added capacities). In in $net1$ we used costs equal to 1 for all the edges, while in $net2$ we used randomly generated costs. Network $net3$ is made up of two identical copies A and B of $net1$, plus a matching between a random subset of the nodes in A and their copies in B, with large capacities. Costs are set to 1 for all the edges. Finally, in all the tests we set $c_{ij} = u_{ij}$ (c.f. (8)) for all the edges, that is to say when we invest on an edge we double its capacity.

In table 1 we summarize the network data: $\mid V \mid$ is the number of nodes, $\mid \mathcal{D} \mid$ is the number of loads, $\mid \mathcal{O} \mid$ is the number of generators, $\mid E \mid$ is the number of edges. Table 2 describes the scenario families we used in our tests: $\mid \Sigma \mid$ is the number of scenarios in the set while $\mid S(\sigma) \mid$ is the number of edges which fails in each scenario $\sigma \in \Sigma$. We constructed these scenario sets by starting with a random (large) list (for example, the list of all pairs) and post-processing the list to remove trivial cases.

problem	$\mid V \mid$	$\mid \mathcal{D} \mid$	$\mid \mathcal{O} \mid$	$\mid E \mid$
$net1 - net2$	300	172	49	409
$net3$	600	344	98	827

Table 1: Size of the networks

Σ	$\mid \Sigma \mid$	$\mid S(\sigma) \mid$
$S2$	2312	2
$S5$	181	$2 - 11$
$S7$	355	$2 - 11$
$T1$	105	1
$T2$	5444	2
$T3$	2398	3
$T4$	7947	$1 - 3$

Table 2: size of the scenario families

To measure the effectiveness of our Branch & Cut algorithm, we compared it against Cplex (used to solve the "natural" formulation given in section 2.1). It is clear the natural formulation will prove unusable if the number of scenarios is very large (the linear program will be huge). Thus, two questions are of interest:

(a) on problems with relatively small number of scenarios, how efficient is the Branch & Cut algorithm?

(b) on problems with many scenarios where the natural formulation is simply too large, does the Branch & Cut algorithm still prove effective?

Table 3 summarizes our results. Our algorithm solves all the instances in less than 1 hour, except for the last instance, where a gap of 11% remains after one hour. On the other hand, within 1 hour of CPU time, Cplex can only solve three problems instances: $net1\_S5$, $net2\_S5$ and $net1\_S7$. On instances $net2\_S7$ and $net3\_T1$ it produces an upper and lower bounds with gaps of 14% and 13.5% respectively. On all the other instances it is not even able to solve the LP-relaxation, denoted by

problem	$B\&C$ time	Cplex time
$net1\_S2$	276 secs.	—
$net1\_S5$	35 secs.	647 secs.
$net1\_S7$	38 secs.	1568 secs.
$net2\_S2$	237 secs.	—
$net2\_S5$	30 secs.	593 secs.
$net2\_S7$	79 secs.	gap = 14%
$net3\_T1$	40 secs.	gap = 13.5%
$net3\_T2$	1762 secs.	—
$net3\_T3$	2954 secs.	—
$net3\_T4$	gap = 11%	—

Table 3: results for the Branch & Cut approach

"—".

In Table 4 we compare our stand-alone heuristic to our Branch & Cut approach and to Cplex. It turns out that the heuristic is very effective. In the table, *opt* is the value of the optimum computed by the Branch & Cut algorithm, *B&C_time* is the Branch & Cut runtime, *heur_UB* is the value of the solution found by the stand-alone heuristic procedure, *heur_time* is the time used by the heuristic, *cpx_UB* is the value of the first feasible solution found by Cplex, and *cpx_time* is the time required by Cplex to find this solution. Cplex was used with default settings, but with the "mip-emphasis" parameter set to "feasibility," and with a time limit of 1 hour.

problem	*opt*	*B&C_time*	*heur_UB*	*heur_time*	*cpx_UB*	*cpx_time*
$net1\_S2$	28	276 secs.	31	21 secs.	—	1 hour
$net1\_S5$	22	35 secs.	26	3 secs.	23	45 secs.
$net1\_S7$	30	38 secs.	36	5 secs.	36	237 secs.
$net2\_S2$	1393.67	237 secs.	1471.2	21 secs.	—	1 hour
$net2\_S5$	1006.9	30 secs.	1126.96	3 secs.	1154.8	47 secs.
$net2\_S7$	1766.61	79 secs.	1980.9	6 secs.	1987.2	232 secs.
$net3\_T1$	46	40 secs.	52	5 secs.	54	147 secs.
$net3\_T2$	116	1762 secs.	126	129 secs.	—	1 hour
$net3\_T3$	78	2954 secs.	87	65 secs.	—	1 hour
$net3\_T4$	LB=116.67	1 hour	131	185 secs.	—	1 hour

Table 4: results for the heuristic approach

Finally, Table 5 describes the behavior of the algorithm on the different runs.

Figures 5 and 6 show how running time is affected by the size of the scenario family. Note that many of these problem instances cannot be handled by Cplex.

3 The second model

In this section we present our second optimization model. As in the last section, we invest to reinforce a network by increasing capacities (line limits) of selected edges. The model in this section considers the dynamics of a cascade, and assumes that no action is taken during the course of a cascade. In other words, we seek a reinforcement plan that can passively "ride out" cascades produced by a given set of scenarios.

The starting point for our work is the "fast dynamics" model for a cascade introduced in [11], [15]. In the model given next, G is the initial network and S is the subset of edges to be removed in some scenario.

problem	$B\&C$ nodes	cuts added	sep. time
$net1\_S2$	12	152	262 secs.
$net1\_S5$	11	118	32 secs.
$net1\_S7$	5	152	35 secs.
$net2\_S2$	8	128	211 secs.
$net2\_S5$	9	134	27 secs.
$net2\_S7$	14	234	71 secs.
$net3\_T1$	12	202	37 secs.
$net3\_T2$	10	788	1641 secs.
$net3\_T3$	39	892	2759 secs.
$net3\_T4$	18	813	3590 secs.

Table 5: Statistcs for the Branch & Cut approach

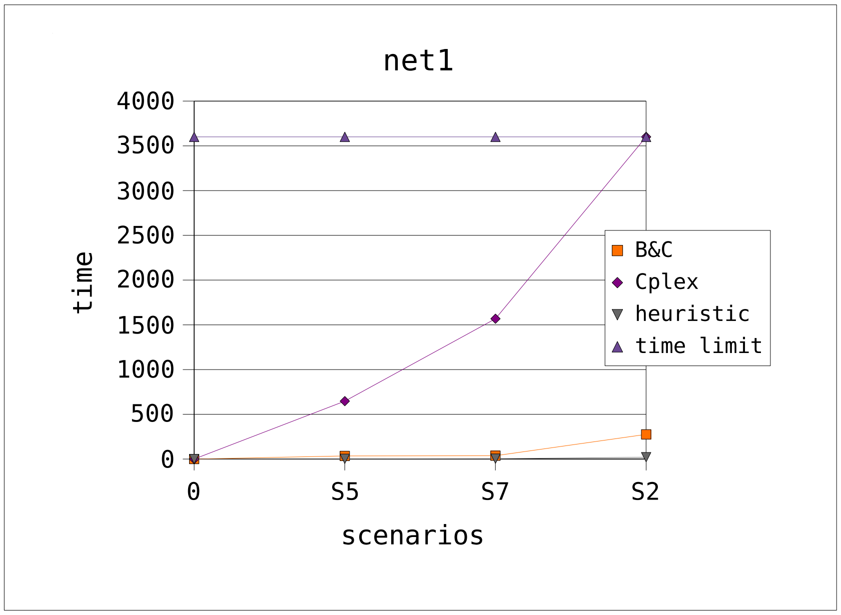

Figure 5: statistics for net1

```
**Procedure 3.1** *GENERIC CASCADE MODEL*

**Input**: a network $G$ and a subset of edges $S$
**Initialize**: $G^0 = G$, $S^1 = S$
**For** $r = 1, 2, \ldots$ **do**
    (comment: simulate round $r$ of the cascade)
    **1. Set** $G^r = G^{r-1} - S^r$.
    **2. Set** $p^r$ = vector of power flows in $G^r$.
    **3. Set** $S^{r+1}$ = set of edges to be removed in round $r+1$.
```

Here, each round is meant to represent a span of time lasting a few minutes or less. The quicker the rounds, the finer the granularity of the model, although the computational complexity of the simulation (and our particular optimization model) will grow.

In order to make this model formal, we need to make precise how Steps 2 and 3 are implemented. This will be done below. For the purposes of this paper, we need to extend this model to specify what we mean when we say that a network has "survived" a scenario. The first ingredient in our extension is that a small amount of lost demand may be tolerable. More precisely, as a cascade evolves it may be the case, for example, that some demand nodes become isolated from generators;

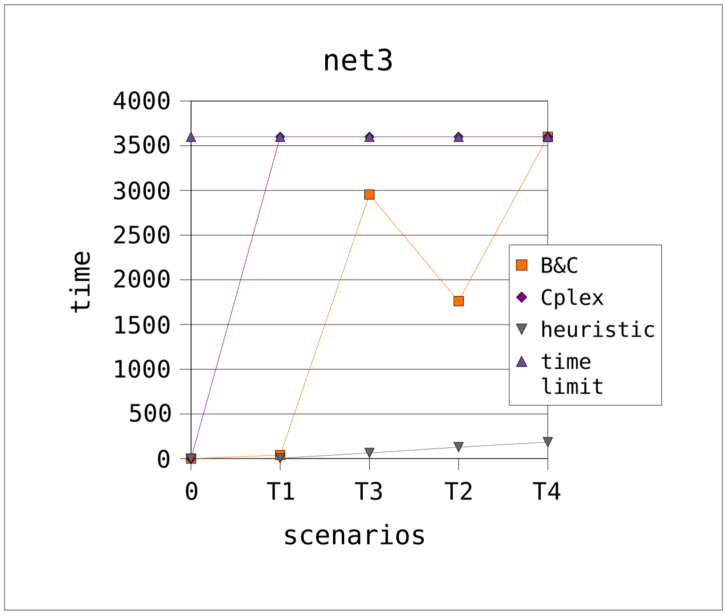

Figure 6: statistics for net3

and this may be acceptable if their demand is not large. The other ingredient is that a cascade that progresses very slowly, i.e., it requires many rounds, may not be viewed as catastrophic.

Notation 3.2 *In what follows, by a "network" we will mean a graph-theoretic network, with demands, bounds (6) on generator nodes, capacities u_{kq} and parameters x_{kq} on the edges.*

Definition 3.3 *Let G be a network and S be a subset of edges of G. Let $0 \leq \mu \leq 1$, and let $R \geq 1$ be an integer. We say that G is (R, μ)-survivable w.r.t. S, if after R rounds of the cascade the fraction of the total demand still being served is at least μ.*

Here, "protecting" could mean increasing the capacity of the edge to a higher value, changing the physical parameters of the edge, duplicating the edge, etc. We will use the terms "protecting" and "reinforcing" interchangeably. Even though in this paper we consider one specific definition of protection, most of our techniques extend directly to the others. We will simply say "survivable" when R and μ are clear from the context.

Notation 3.4 *Let G be a network with m edges, and let $y \in \{0, 1\}^m$. We denote by $G(y)$ the network obtained by protecting those edges e with $y_e = 1$.*

Now suppose we have a network G with m edges. For each edge e, let w_e, the cost of protecting e, be given. Let $R \geq 1$ and $0 \leq \mu \leq 1$ be given. Finally, suppose we have a family of scenarios Σ. As before, for each $\sigma \in \Sigma$ we denote by $S(\sigma)$ the set of edges removed in σ. The main problem we consider is:

Problem 3.5 *Find $y \in \{0, 1\}^m$ of minimum cost, such that $G(y)$ is (R, μ)-survivable w.r.t. $S(\sigma)$, for each $\sigma \in \Sigma$.*

3.0.1 Making Step 3 explicit

Now we return to the task of making more explicit the generic cascade model 3.1. First we will consider how Step 3 is carried out. The starting point for this is the notion of overload: if edge $\{k, q\}$ has capacity u_{kq} and carries flow p_{kq}, then its *overload* is $|p_{kq}|/u_{kq}$. If the overload is very large, the edge will quickly fail (and will be quickly turned off by automatic equipment). If the overload is greater than 1, but not very large, the edge can operate at the overload for some time, but will eventually be turned off as well. These simple rules, which are implemented in practice, are motivated by thermal considerations.

In [11] the following model is proposed. Let the overload of edge $\{k,q\}$ be $\rho_{qk} > 1$. Then edge $\{k,q\}$ is removed with probability $H(\rho_{qk})$, where $H : (1,+\infty) \to (0,1]$ is a monotonely increasing function. Using this model, the generic cascade model becomes a probabilistic process. A possible disadvantage of this approach is the need to choose a specific function H. For the sake of simplicity, in this paper we use a deterministic model given in the next section. This model should provide the same qualitative behavior, e.g. the more overloaded an edge is, the sooner it should be removed from the network.

Our model relies on one additional parameter, $0 \le \alpha \le 1$. For each round r, our model will maintain an additional vector $s^r \in R^m_+$. Furthermore, as an input to the model, there will be an initial power flow vector p^0 – these are the power flows *before* the scenario is realized. We set $s^0_{kq} = |p^0_{kq}|$ for all $\{k,q\}$, and at each round r ($r = 1,2,\ldots$), we perform the following update:

$$s^r_{kq} \;=\; \alpha|p^r_{kq}| + (1-\alpha)s^{r-1}_{kq}, \quad \text{for each edge } e. \tag{17}$$

Then, edge $\{k,q\}$ is removed in round r if $s^r_{kq} > u_{kq}$. Thus, essentially, if $\alpha < 1$ an edge has to remain overloaded for several rounds in order to be removed. If the overload, at a certain round, is very high, the edge might be removed immediately. The system has "memory", which is precisely what happens in the real-life setting from a thermal standpoint.

3.0.2 Making Step 2 explicit

Next, we consider how to implement Step 2 of the generic cascade model 3.1. In general, there may be multiple solutions to the power flow equations, i.e., p^r is not uniquely defined. Suppose that for some $r > 1$, $S^r = \emptyset$, i.e. $G^r = G^{r-1}$. Then it is reasonable to insist that $p^r = p^{r-1}$. If $S^r \neq \emptyset$, however, then the network changes and some rule must be employed to choose the power flows. One possibility is to ask that that the amount of power produced by each generator (see eqs. (4 and (5)) remains constant. This rule will in general produce (trivially) infeasible problems. Moreover, the rule is overly constraining – in practice, generators can adjust their output (subject to e.g. (6)) according to demands, and without much centralized control. In addition, we should allow for some of the demand to be lost (this might be required, for example, if G^r is disconnected). Finally, we want our model to assume a minimum amount of centralized control – essentially, we want to assume that it is the physics that determine the new power flows.

One idea would be to pick p^r so as to minimize $||p^r - p^{r-1}||_2$, subject to some constraints, i.e., pick a "minimum energy change". For computational expediency, we pick a linearized model. This is where we use the DC flow model (in fact, it is our only use of the DC flow model).

In order to motivate our approach, consider the following simple example. Suppose we have a network with three generators: these are nodes 1, 2 and 3, with $P^{min}_i = 0$ and $P^{max}_i = 20$ for $1 \le i \le 3$. Suppose that nodes $4-7$ are demand nodes, with demands (respectively) 8, 15, 14 and 5. Suppose that after a round of failures, the network has split into three components: one containing nodes 1, 2, 4, 5 and 6, one containing node 3 (and no demand nodes) and one containing node 7 (and no generator nodes). Thus, the demand from node 7 is lost. However, in the first component we have a total supply of 40 units and a total demand of 37. It is not unreasonable to expect that all 37 units of demand will be served – each node will continue demanding the same amount over a short span of time (the alternative, so-called "load shedding" approach, is viewed as undesirable). In general, if a component has total supply Ω and total demand Γ, then the demand served in this component will be $\min\{P,\Gamma\}$. By Lemma 1.1, we know that for each fixed choice of Ω_1 and Ω_2 with $\Omega_1 + \Omega_2 = 37$ there will be a unique set of power flows that meet the demand. But how do we pick Ω_1 and Ω_2? The choice that we make is that for which the new power flow vector is *closest* to the old one, in a precise sense made clear next.

The approach that we follow is as follows. Let $\mathcal{D}$ be the set of demand nodes; for $k \in \mathcal{D}$ let D^0_k denote the demand at k at the start of the cascade. For $r \ge 1$, let

$$\mathcal{C}^r \;=\; \text{the set of connected components of } G^r. \tag{18}$$

For each $K \in \mathcal{C}^r$, let

$$\begin{aligned}
\Omega_K &= \sum\{P_k^{max} : k \in K,\ k \text{ a generator}\} && (19)\\
\Gamma_K &= \sum\{D_k^0 : k \in K \cap \mathcal{D}\} \qquad \text{(total demand in } K\text{)} && (20)\\
\mu_k &= \min\{\Omega_K, \Gamma_K\} && (21)
\end{aligned}$$

Our model **stipulates** that the total amount of demand served in component K equals μ_k. Due to the lower limits P_k^{min} on the generators this may be infeasible (see below). But otherwise, by Lemma 1.1 there is a feasible flow vector that delivers a total demand of μ_k in component K. Finally, let μ^r denote the faction of all demand still being served at the end of round r, i.e.

$$\mu^r = \frac{\sum_{K \in \mathcal{C}^r} \mu_k}{\sum_{k \in \mathcal{D}} D_k^0}. \tag{22}$$

Our approach solves a linear program which uses a variable θ_k^r for each node k (the angle at k) and variables p_{kq}^r for each arc $\{k,q\}$:

$$\begin{aligned}
\min\ & \sum_k |\theta_k^r - \theta_k^{r-1}| && (23)\\
\text{s.t.}\ \ & x_{kq}\, p_{kq}^r - \theta_k^r + \theta_q^r = 0 \quad \forall \{k,q\} \notin \bigcup_{i=0}^r S^i && (24)\\
& p_{kq}^r = 0 \quad \forall \{k,q\} \in \bigcup_{i=0}^r S^i && (25)\\
& P_k^{min} \le \sum_{kq} p_{kq}^r \le P_k^{max} \quad \forall \text{ generator node } k \ \ \text{(generator limit)} && (26)\\
& \sum_{kq} p_{qk}^r - D_k^r = 0 \qquad \forall\, k \in \mathcal{D} && (27)\\
& 0 \le D_k^r \le D_k^0 \qquad \forall\, k \in \mathcal{D} && (28)\\
& \sum_{k \in \mathcal{D}} D_k^r \ge \mu^r \sum_{k \in \mathcal{D}} D_k^0 && (29)\\
& \sum_{kq} p_{kq}^r = 0 \quad \forall \text{ non-generator, non-demand node } k && (30)
\end{aligned}$$

Here, the quantities θ_k^{r-1} are constants (determined in the previous round). Constraints (24)-(25) describe the operational constraints of the network – recall that S^i is the set of edges removed in round i. Constraint (29) is used to imply that precisely a fraction μ^r of the demands is served (if we remove this constraint we might have a feasible solution with $\sum_{k \in \mathcal{D}} D_k^r < \mu^r \sum_{k \in \mathcal{D}} D_k^0$). If, for a certain component $K \in \mathcal{C}^r$, we have

$$\sum\{P_k^{min} : k \in K,\ k \text{ a generator}\} > \Gamma_K$$

the linear program is infeasible, precisely because of constraint (29).

Thus, the new linear program will pick a new power flow whose corresponding angle vector is closest (in the L_1-metric sense) to the old one. Roughly speaking, we mean to model a smallest possible change in the configuration of the network that still delivers the right amount of demands. To some degree, this models a minimum amount of operator control over the network. We stress, however, that what we have described is simply one of many "reasonable" ways of implementing Step 2 of the generic cascade model 3.1 (another one is given in [11]). We have, in fact, implemented other models, without drastic qualitative impact on the overall optimization problem (3.5).

3.0.3 Putting the model together

Now we can provide our complete cascade model.

```
Procedure 3.6 EXTENDED CASCADE MODEL

Inputs: a network $G$ and a subset of edges $S$,
        a (feasible) power flow vector $p$ and angle vector $\theta$,
        parameters: integer $R \geq 1$, $0 \leq \alpha \leq 1$, $0 \leq \mu \leq 1$.
Initialize: $G^0 = G$, $S^1 = S$, $p^0 = p$, $s^0 = |p|$ (componentwise) and $\theta^0 = \theta$.
For $r = 1, 2, \ldots$ do
    (comment: simulate round $r$ of the cascade)
    0. If $r > 1$, set $S^r = \{\{k,q\} : s_{kq}^{r-1} > u_{kq}\}$.
    1. Set $G^r = G^{r-1} - S^r$
        1a. Determine $\mu^r$ using (18)-(22).
        1b. If $\mu^r < \mu$, STOP: $G$ is not survivable w.r.t. $S$.
        1c. Otherwise, if $r = R$, STOP: $G$ is survivable w.r.t. $S$.
    2. Let $p^r, \theta^r$ be solution to the L.P. (23)-(30).
    3. Determine $s^r$ from $p^r$ and $s^{r-1}$ using equation (17).
```

Two critical parameters in this model are α, used in equation (17) to set the vectors s^r, and R, the number of rounds. Using larger values for R allows us to model shorter round-to-round times, i.e. a we model the cascade with smaller time granularity. Using larger values for α makes the evolution of the network more abrupt. Hence, the richer models will be those with larger values for α and larger values for R.

Below we will describe an algorithm for solving Problem 3.5 using the extended cascade model.

3.0.4 Why the problem is difficult

Clearly, when we have many scenarios the problem will be computational challenging: verifying that a vector y is feasible entails solving, in the worst case, R linear programs for each scenario.

However, there is a much more significant difficulty which impacts the combinatorial aspect of the problem. Consider the network in Figure 7. Here we have one generator (node 0, with 18 units of capacity) and two demand nodes (nodes 3 and 5, with 9 units of demand each). We have one scenario; in this scenario edge $\{0,4\}$ is removed. We have $\alpha = 1.0$ (c.f. (17)), that is, there is no "memory"; $R = 2$ and $\mu = 0.6$. Suppose that when we protect an edge we **double** its capacity.

Consider first the vector y^1 with $y^1_{02} = 1$ and $y^1_{kq} = 0$ for all other $\{k,q\}$ (i.e., we reinforce $\{0,2\}$ only). Then, in the first round, $\{0,1\}$ and $\{0,5\}$ are removed: because x_{02} is large, most of the 18 units of demand flow on $\{0,1\}$ and $\{0,5\}$, in equal amounts (in fact, $p^1_{01} = p^1_{05} \approx 8.97$). However, in the second round no edges are removed: all demand flows on $\{0,2\}$ and is evenly split on the paths $0,2,1,3$ and $0,2,5,4$. Thus, y^1 is feasible.

On the other hand, consider now the vector y^2 with $y^2_{01} = y^2_{02} = 1$ and $y^2_{kq} = 0$ for all other $\{k,q\}$. Then, in the first round, again we have that $p^1_{01} = p^1_{05} \approx 8.97$ – but now only $\{0,5\}$ is removed.

In the second round, we will have that $p^2_{01} \approx 17.794$, $p^2_{13} \approx 17.765$ and $p^2_{34} \approx 8.765$. Thus, $\{0,1\}$, $\{1,3\}$ and $\{3,4\}$ are removed; consequently node 3 becomes isolated and we lose 50% of the demand. So y^2 is infeasible.

In summary: we can make an infeasible vector y, feasible, by unprotecting an edge (that is protected under y). This non-monotonicity is similar to what is given in the examples in Section 2.1.1; it arises in many forms and in all reasonable versions of the extended cascade model. Further, it significantly impacts the search for effective cutting-planes in our algorithm.

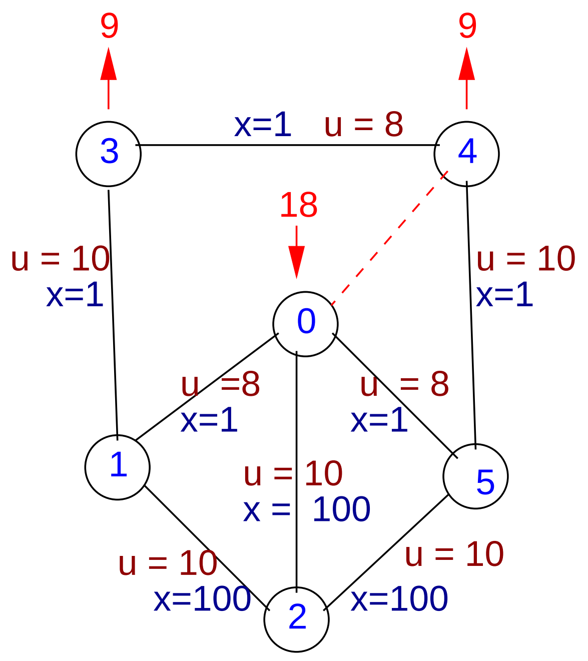

Figure 7: A pathological case

3.1 A cutting-plane algorithm

Let $\Phi \subseteq \{0,1\}^m$ be the set of 0/1 vectors y such that $G(y)$ is (R,μ)-survivable w.r.t. $S(\sigma)$, for each $\sigma \in \Sigma$. Our problem can simply be stated as: $\min\{w^T y \,:\, y \in conv(\Phi)\}$.

For this problem we propose a classical cutting-plane algorithm. This algorithm works with the vector of 0/1-variables y; at any iteration we will have a working formulation $Ay \geq b$, initially empty. At each iteration the algorithm solves a simpler optimization problem to determine a certain 0/1 vector y, if this vector y is feasible then we are done, and otherwise we find an inequality that separates y from $conv(\Phi)$.

Procedure 3.7 *CUTTING-PLANE ALGORITHM*

Initialize: $L = 0$.

```
    1. Let $y^* \in \{0,1\}^m$ be the solution to the problem $\min\{w^T y \,:\, Ay \geq b\}$.
    2. If $y^* \in \Phi$, then STOP: $y^*$ is optimal.
    3. Otherwise, let $\beta^T y \geq \beta_0$ be an inequality valid for $\Phi$ which is violated by $y^*$.
    4. Add $\beta^T y \geq \beta_0$ to $Ay \geq b$, update $L \leftarrow w^T y^*$, and go to 1.
```

At any point of the procedure, L is a lower bound on the optimal cost. The feasibility check in step 2 is implemented using the extended cascade model 3.6. Clearly, Step 3 is critical. In addition, we should add as an additional step the periodic execution of heuristics: each run of a heuristic would produce a 0/1 vector y; if $y \in \Phi$ then we can update an upper bound U on the value of the problem, and otherwise we can add a valid inequality as in Step 3. Finally, we may need to modify step 2 in case the optimization problem in that step becomes too difficult. We take up these issues below.

3.1.1 A combinatorial inequality

Suppose a vector $y^* \in \{0,1\}^m$ is such that $G(y^*)$ is not survivable w.r.t. some scenario σ. Let $I_k = \{\text{edges } e : y^*_e = k\}$, $k = 0, 1$. Then the trivial inequality

$$\sum_{e \in I_0} y_e + \sum_{e \in I_1} (1 - y_e) \quad \geq \quad 1, \tag{31}$$

is valid and certainly cuts off y^*. Of course, this inequality is also trivially weak. Next we present a combinatorial inequality that is stronger. To describe the precise form of this inequality, we will consider the specific model of "protection" mentioned above, namely: protecting an edge $\{k, q\}$ means increasing its capacity from u_{kq} to a strictly higher value u^{new}_{kq}. The inequality discussed below can easily be adapted to other models of protection.

To motivate the inequality, we consider a simple example. Figures 8 and 9 show in outline the evolution of a cascade as per the extended cascade model. In the example, all demands and generator capacities have been scaled so that the total demand and the total generator capacity are both equal to 1.0. Further we assume that $\mu = 0.75$, i.e., we want to satisfy at least 75% of the demand. The sets I_0 and I_1 are not shown. Round 1 is not shown, but it is assumed that the removal of initial set $\mathcal{F}^1 = S(\sigma)$ did not disconnect the network.

Figure 8 shows an outline of the network at the start of round 2. Here, the edges labeled 1-7 (shown in red) are the set $\mathcal{F}^2$, i.e. the set of edges removed at the start of round 2. Thus, during round 2 we have five components (labeled $K_1 - K_5$). Using the notation of equations (18)-(22) and using the shorthand $\mu_i = \mu_{K_i}$, we have that $\mu_1 = 0.5$, $\mu_2 = 0.3$, and $\mu_k = 0$ for $3 \leq k \leq 5$ (these quantities shown towards the bottom of each component in Figure 8). Thus, $\mu^2 = 0.8$, i.e. we satisfy 80% of the demand, and we go to the third round.

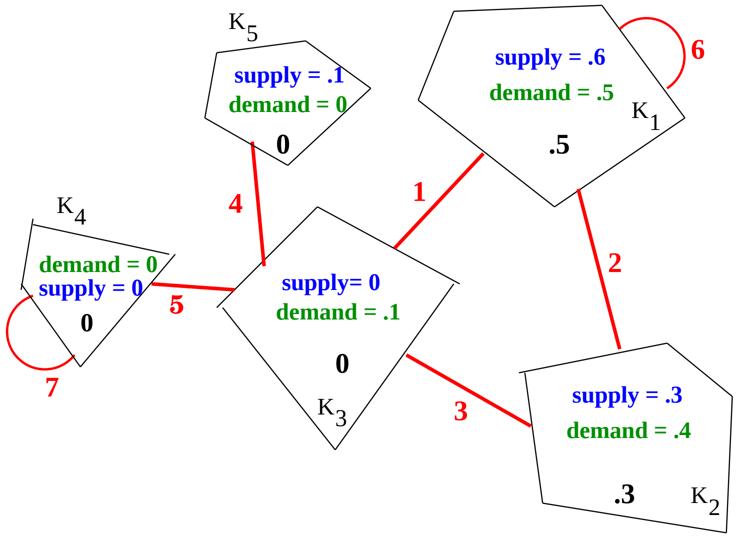

Figure 8: Round 2

Figure 9 shows that six additional edges (labeled 8-13) are removed at the start of round 3, and we now have ten components, labeled $K_6 - K_{15}$ (The outlines of the round 2 components are also shown). The supply of component 6 is .1, its demand is .2, and so $\mu_6 = .1$; the supply of component 7 is .5, its demand is .3, so $\mu_7 = .3$; the supply of component 8 is .29, its demand is .2, so $\mu_8 = .2$; and the supply of component 9 is .01, its demand is .2, so $\mu_9 = .01$. For all other round 3 components either the demand or supply is zero. In this case we have that $\mu^3 = .1 + .3 + .2 + .01 = .61$ which is less than the desired value, 0.75, and so the network has not survived the cascade.

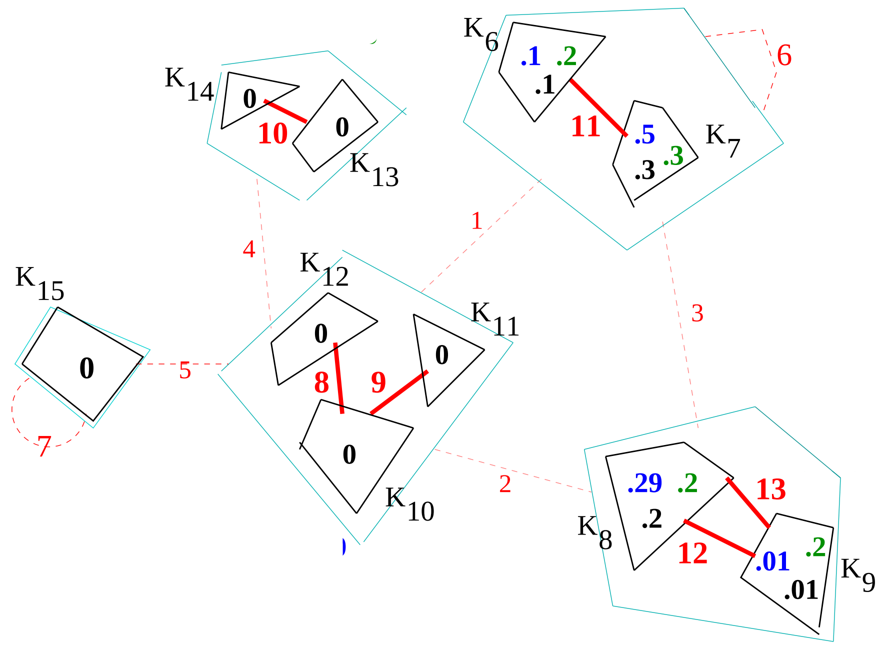

Figure 9: Round 3

In order to drive our example, in the following analysis we assume that $y^*_e = 0$ for $1 \leq e \leq 11$. With this assumption, the inequality

$$\sum_{e=1}^{11} y_e + \sum_{e \in I_1} (1 - y_e) \quad \geq \quad 1, \tag{32}$$

is valid (and violated by y^*) and is an improvement over (31). To see that (32) is valid, suppose that for some 0/1 vector y the left-hand side of (32) equals zero. Then, by construction of the cascade model, during round 1 the power flows in $G(y)$ will be the same as in $G(y^*)$. Since $y_e = y^*_e$ for $1 \leq e \leq 7$, and for $e \in I_1$, the sets $\mathcal{F}^1$ will also be the same in $G(y)$ as in $G(y^*)$, i.e. edges $1 - 7$ are removed at the start of round 2. So during round 2 we will have exactly the same components in $G(y)$ as in $G(y^*)$ – not just vertex-wise, but edge-wise as well, because of the second term in the left-hand side of (32). Thus, the power flows in each of those components will be the same in $G(y)$ as in $G(y^*)$, and by a similar argument as before edges $8 - 11$ will be removed at the start of round 3; the network fails at that point proving that y is infeasible.

But we can do more: we claim that

$$\sum_{e=1}^{7} y_e + y_{11} + y_{12} + y_{13} + \sum_{e \in I_1} (1 - y_e) \quad \geq \quad 1, \tag{33}$$

is valid. The reason for this is that if the left-hand side of (33) were equal to zero, then after round 2 we would still have that the same components K_6, K_7, K_8, K_9, and, in the best case, three other components, all of which have either 0 demand or 0 supply; altogether the value of μ^2 would still equal .61.

Inequality (33) can itself be improved. We claim that

$$y_1 + y_2 + y_3 + y_6 + y_{11} + y_{12} + y_{13} + \sum_{e \in I_1} (1 - y_e) \quad \geq \quad 1, \tag{34}$$

is valid. If the left-hand side of (34) were equal to zero, then after during round 2 we would have component K_1, component K_2, and in the best case just one more component with supply and demand equal to .1. Then during round 3 we would again have components K_6, K_7, K_8, K_9, plus one or more components, and now $\mu^2 \leq .61 + .1 < .75$.

Next we describe the general version of inequality (34). The inequality will strengthen (31) by eliminating some of the summands in the first term as we did in the example. In order to describe the inequality we need a few definitions.

We consider a 0/1 vector y^* such that $G(y^*)$ does not survive some scenario σ; and the cascade ends in step 2b of the extended cascade model 3.6 during round $\hat{r}$. Again we use the notation from (18)-(22) and from the extended cascade model:

- G^r is the remaining network at the start of round r $(1 \leq r \leq \hat{r})$,
- for each component $K \in \mathcal{C}^r$, Ω_K is the total capacity of the generators in K, Γ_K is the total demand within K, and $\mu_k = \min\{\Gamma_K, \Omega_K\}$,
- for each round r, μ^r is the fraction of the demand still being served at the end of the round,
- for each round r, $\mathcal{F}^r$ is the set of edges removed at the start of round r.

When referring to a specific vector $y \in \{0,1\}^m$, we will use the notation $G^r(y), \mu^r(y), \mathcal{F}^r(y)$, etc.

Assumption: Without loss of generality, in the remainder of this section we assume that the demands have been scaled so that their sum is 1, i.e., so that $\mu^r = \sum_{K \in \mathcal{C}^r} \mu_k$ for each round r.

If K is a component during any of the rounds, $V(K)$ will denote the set of nodes in K.

Definition 3.8 *Let* Π *be a partition of the components of* $G^{\hat{r}}(y^*)$*. A component* K *of* $G^r(y^*)$ *(*$1 \leq r \leq \hat{r}$*) shatters* Π *if* $V(K)$ *intersects at least two classes of* Π*, e.g.* $V(K) \cap V(K') \neq \emptyset$ *and* $V(K) \cap V(K'') \neq \emptyset$ *for some components* K'*,* K'' *of* $G^{\hat{r}}(y^*)$ *which are in different classes of* Π*.*

Consider the example provided above, and let Π be the partition

$$\{\{K_6\},\{K_7\},\{K_8\},\{K_9\},\{K_{10}, K_{11}, K_{12}, K_{13}, K_{14}, K_{15}\}\}.$$

Components K_1 and K_2 of $G^2(y^*)$ shatter this partition; not so for components $K_3 - K_5$ (which are contained in the last class of Π).

Notation 3.9 *Given a partition* Π *of the components of* $G^{\hat{r}}(y^*)$*, for* $1 \leq r \leq \hat{r}$ *we denote by* Z^r *the subset of edges* $\{k, q\} \in \mathcal{F}^r(y^*)$*, such that either*

(i) at least one of k *and* q *are in a component of* $G^r(y^*)$ *that shatters* Π*, or*

(ii) k *and* q *are in different classes of* Π*, i.e. there are components* K*,* Q *of* $G^{\hat{r}}(y^*)$ *in different classes of* Π *with* $k \in K$ *and* $q \in Q$*.*

Notation 3.10 *If* K *and* Q *are graphs,* $K \subseteq Q$ *means inclusion in the vertex and edge sense.*

In the above example, and using the same partition as before, we have that edges $11 - 13$ are in Z^3 according to the second criterion; edges $1 - 3$ are in Z^2 (either criterion) and edge 6 is in Z^2 according to the first criterion.

Our inequality is:

$$\sum_{r=2}^{\hat{r}} \left[\sum_{e \in I_0 \cap Z^r} y_e \right] + \sum_{e \in I_1} (1 - y_e) \;\; \geq \;\; 1. \qquad (35)$$

Theorem 3.11 *Let Π be a partition of the components of $G^{\hat{r}}(y^*)$, and assume that*

$$\sum_{\mathcal{J} \in \Pi} \min \left\{ \sum_{K \in \mathcal{J}} \Gamma_K, \sum_{K \in \mathcal{J}} \Omega_K \right\} \ < \ \mu. \tag{36}$$

Then (35) is a valid inequality.

[comment: each $\mathcal{J}$ is a class of Π, i.e., a set of components of $G^{\hat{r}}(y^*)$].

Proof. Let y be a 0/1 vector that violates (35), i.e., the left-hand side of (35) is zero. We claim that $G(y)$ does not survive scenario σ. To do so we will prove by induction on r ($1 \le r \le \hat{r}$) that for each component K of $G^r(y)$, either (a) or (b) hold:

(a) K is a component of $G^r(y^*)$. Further, for each edge $e \in K$, $p^r_e(y^*) = p^r_e(y)$ and $s^r_e(y^*) = s^r_e(y)$ and for each node $v \in K$, $\theta^r_v(y^*) = \theta^r_v(y)$.

(b) for some class $\mathcal{J}$ of Π, $V(K) \subseteq \bigcup_{J \in \mathcal{J}} V(J)$.

Applying this statement for $r = \hat{r}$ implies that no component of $G^{\hat{r}}(y)$ can intersect more than one class of Π, and thus $G(y)$ has a value of $\mu^{\hat{r}}$ which is at most that of $G(y^*)$, which by assumption is less than the desired target of μ, proving the claim.

Now we prove the inductive statement. For $r = 1$, by construction of the extended cascade model 3.6 we have that all components, power flows, etc. in $G^1(y)$ and $G^1(y^*)$ are identical, and (a) applies.

Suppose we have proved the inductive statement for r and now we wish to do so for $r+1$. Note that every component of $G^{r+1}(y)$ is contained in some component of $G^r(y)$. Hence, it suffices to consider some arbitrary component Q of $G^r(y)$ and show that every component K of $G^{r+1}(y)$ with $K \subseteq Q$ satisfies either (a) or (b). If Q satisfies (b) we are done, so in what follows we assume that Q satisfies (a).

Claim: If component K of $G^{r+1}(y^*)$ with $K \subseteq Q$ shatters Π, then K is also a component of $G^{r+1}(y)$ (and therefore is covered by case (a)).
Below we will prove the claim. Further, we will also prove that all *other* components of $G^{r+1}(y)$ are covered by case (b). This will complete the proof of the inductive step.
Proof of the claim. Let $K \subset Q$ be a component of $G^{r+1}(y^*)$ such that K shatters Π. By the inductive assumption, the vector s^r restricted to Q is the same in $G^r(y)$ and $G^r(y^*)$. We wish to show that any edge $e \in Q$ with at least one endpoint in K satisfies $e \in G^{r+1}(y)$ if and only if $e \in G^{r+1}(y^*)$ (this will show that K is a component of $G^{r+1}(y)$). Since $e \in Q$ then $s^r_e(y^*) = s^r_e(y)$ by the inductive assumption. So if $s^r_e(y) \le u_e$ then e is both in $G^r(y^*)$ and $G^r(y)$. If $u^{max}_e < s^r_e(y)$ then e is neither in $G^r(y^*)$ nor in $G^r(y)$. Finally, if $u_e < s^r_e(y) \le u^{max}_e$, then $e \in \mathcal{F}^{r+1}(y^*)$ precisely when $y^*_e = 0$ – but in that case $e \in I_0 \cap Z^r$, and since y violates (35) then $y_e = 0$ as well, and so $e \in \mathcal{F}^{r+1}(y)$ also. This proves the claim.

Next, consider a consider a component K of $G^{r+1}(y)$ which is not one of of those produced in the previous paragraph. We claim that there is a unique class $\mathcal{J}$ of Π such that $V(K) \cap \bigcup_{J \in \mathcal{J}} V(J) \neq \emptyset$. For otherwise, there is an edge $e \in G^{r+1}(y)$ with both ends in K, and one end in one class of Π and the other end, in a different class. We must have $e \in \mathcal{F}^{r+1}(y^*)$, or else both ends of e are in the same component of $G^{r+1}(y^*)$, and this component would shatter Π, a contradiction since we assume that K was not handled by the claim. So $s^r(y^*) > u_e$ and therefore $e \in Z^{r+1}$ (second criterion in the definition of Z^{r+1}).

If $y^*_e = 1$, then $u^{max}_e < s^r(y^*) = s^r(y)$, but then $e \notin G^{r+1}(y)$, a contradiction. So $y^*_e = 0$, and so $e \in I_0 \cap Z^{r+1}$, and hence again $y_e = 0$ (y violates (35)) and thus $e \notin G^{r+1}(y)$, a contradiction. This concludes the proof. ■

Below we will discuss how to separate over the inequalities (35). But first we discuss some simple means for further tightening the inequality as post-processing.

First improvement. The second sum in the left-hand side of (35) makes the inequality "weak" because in general it contains many terms. Suppose we have an edge e with $y^*_e = 1$, and further $e \in \mathcal{F}^r$ for some r. In other words, $s^r(y^*) > u^{max}_e$. In such a case it would be tempting to remove the term $(1 - y_e)$ from (35) – after all, "it does not matter" whether we protect e or not. This argument is flawed, because there may exist an earlier round $t < r$ such that $s^t(y^*) > u_e$ – had set $y^*_e = 0$, then e would have been removed at round $t + 1$, thereby potentially changing the cascade. This is the "Braess' law" effect. The rule that we can apply is: for each edge e, define

$$t = t_e \quad = \quad \min\{h : 1 \le h \le \hat{r} - 1,\ s^h_e(y^*) > u_e\}. \tag{37}$$

If $y^*_e = 1$, and either $t = \hat{r} - 1$ or $s^t(y^*) > u^{max}_e$, then we can remove the term $(1 - y_e)$ from (35). There are other conditions under which the term can be removed – these involve combinatorial criteria similar to those that yield (35) as an improvement of (31), but will be omitted for brevity.

Second improvement. Consider now an edge e such that y_e appears in the first sum in the left-hand side (35), for some round r. So $y^*_e = 0$ and r is the smallest t with $s^t_e(y^*) > u_e$. If in addition $s^r_e(y^*) > u^{max}_e$ then we can eliminate the term y_e from the sum.

In summary, after applying all these improvements, the final inequality that we obtain has the general structure $\sum_{e \in A_0} y_e + \sum_{e \in A_1}(1 - y_e) \ge 1$ for appropriate sets A^0, A^1.

3.1.2 Separating inequality (35).

Let y^* be a 0/1 vector such that $G(y^*)$ is not survivable w.r.t. some scenario σ. We wish to choose an inequality (35) that is violated by y^*. Such an inequality can always be found (we can choose Π to be the partition with one class per component in the last round) but we wish in addition to make the left-hand side of (35) as sparse as possible.

In this task, we are helped by the fact that a typical cascade exhibits what might be termed "random graph" behavior. By this we mean that the number of components remains relatively small until the last round or two, when it can become large. And in the last round, most components will have few nodes (often single nodes) and only a few components (often one or two) are large. Table 6 shows a typical evolution.

Round r	$\lvert\mathcal{F}^r\rvert$	**Components**	**Demand served (%)**
1	2	1	100.0
2	8	3	100.0
3	17	8	87.66
4	20	16	82.72

Table 6: A typical evolution

Table 3.1.2 shows the structure of the components at round 4 of the cascade in Table 6.

Here, "size" refers to the number of nodes in each component, Γ_k, Ω_k and μ_k are as defined before: the demand within component k, the supply within k, and the demand actually satisfied

component k	**size**	$\boldsymbol{\Gamma_k}$	$\boldsymbol{\Omega_k}$	$\boldsymbol{\mu_k = \min\{\Gamma_k, \Omega_k\}}$
1	238	0.7508	0.6887	0.6887
2	27	0.1921	0.0940	0.0940
3	11	0.0229	0.0547	0.0229
4	6	0.0015	0.0024	0.0015
5	5	0.0135	0.0629	0.0135
6	3	0.0067	0.0097	0.0067
7	1	0	0	0
8	1	0	0	0
9	1	0.0082	0	0
10	1	0.0027	0	0
11	1	0	0	0
12	1	0.0017	0	0
13	1	0	0	0
14	1	0	0.0555	0
15	1	0	0.0589	0
16	1	0	0.0391	0

Table 7: Components in round 4 of Table 6

withint k. Here all demands and supplies have been scaled by the sum of the demands (total supply ≈ 1.066).

The following heuristic exploits this structure. Suppose the cascade ends (with failure) at round $\hat{r}$.

1. Let $K_1, K_2, \ldots, , K_N$ denote the number of components of $G^{\hat{r}}(y^*)$. Without loss of generality, assume that they are numbered so that $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_N$.

2. Form a partition of the form

$$\Pi \,=\, \{\, \{K_1\}\,,\, \{K_2\}\,,\, \ldots\,,\, \{K_n\}\,,\, \{K_{n+1}, K_{n+2}, \ldots, K_N\}\,\}.$$

The parameter n is chosen smallest such that

$$\sum_{\mathcal{J} \in \Pi} \min \left\{ \sum_{K \in \mathcal{J}} \Gamma_K,\, \sum_{K \in \mathcal{J}} \Omega_K \right\} \;<\; \mu \tag{38}$$

holds. Note that this is exactly the condition in Theorem 3.11; the choice of $n = N$ by assumption satisfies the condition (i.e. at round $\hat{r}$ the network failed).

3. For $t = \hat{r} - 1, \hat{r} - 2, \ldots, 2$, consider the set of edges e with $y^*_e = 0$ that are removed during round t. If any such edge has one end in one class of Π and the other end in another class, then attempt to merge the two classes into one. The attempt is successful when the quantity in (38) remains smaller than μ.

Step 3 of this procedure attempts to reduce the number of terms in the first summand in (35). A similar step can be applied to the second summand, but again we will skip its description for brevity.

In our implementation, this is the only separation algorithm that we employ – so the default version of our cutting-plane algorithm 3.7 generates a single cut. An open question is whether it would be helpful to run multiple separation heuristics and incorporate several cuts at once. In the following section we will show how upper bound heuristics for the overall optimization problem (3.5) can be used to generate further cuts.

3.2 Upper bound heuristics

The basic cutting-plane algorithm 3.7 described in Section 3.1, equipped with the inequalities described in the previous section, can solve nontrivial problems. In order to handle more complex problems, and to speed up the algorithm, upper bound heuristics prove useful. These heuristics are used and are described below.

The common approach that we use with our heuristics is the following. Each heuristic H is an algorithm that outputs some 0/1 vector, which might or might not be feasible. Suppose that at some iteration of the cutting-plane algorithm we interrupt the algorithm, and run H, obtaining the vector $\hat{y}$. Then

H.1 If $G(\hat{y})$ survives every scenario σ then we obtain a valid upper bound on the optimization problem (3.5).

H.2 Otherwise, we obtain a cut, using the techniques from Section 3.1.1.

Step H.1 can be expensive if $G(\hat{y})$ survives many scenarios. For that reason we cannot run the heuristics very frequently. Also, obtaining tight upper bounds can in principle help the cutting-plane algorithm terminate sooner (with an approximate answer). But, in our experience, this has not been reliable. Instead, it is H.2 that proves most useful. Here, the heuristic "samples" the space of all solutions in an "interesting" region, and thus the cut added in H.2 proves significant. This observation is in line with folklore observations that have been made primarily in the context of Benders' decomposition algorithms.

The final component in our use of heuristics a pruning algorithm, which assumes that the costs w_{ij} are nonnegative. Given a feasible 0/1 vector $\hat{y}$ we pick any edge $\{k, q\}$ with $\hat{y}_{kq} = 1$. We temporarily set $\hat{y}_{kq} = 0$. If $\hat{y}$ is feasible we make $\hat{y}_{kq} = 0$ permanent; otherwise we find a cut (35) and we make $\hat{y}_{kq} = 1$ permanent. We then continue with some other edge.

The pruning algorithm is run with a time limit; each time a cheaper feasible solution is found the run-time clock is reset to zero.

Next we describe the three heuristics we have implemented.

First heuristic. Suppose we have carried out Step 1 of the basic cutting-plane algorithm to obtain the vector $y^* \in \{0,1\}^m$ which solves the (current) problem $\min\{w^T x \, : \, Ay \geq b\}$, and that furthermore $G(y^*)$ does not survive some scenario σ. Our first heuristic chooses the 0/1 vector $\hat{y}$ as follows: $\hat{y}_e = 1$ if either $y^*_e = 1$ or if $e \in \mathcal{F}^2(y^*)$.

The rationale for this heuristic is that (usually, but not always) $G(\hat{y})$ survives scenario σ. However, if $G(\hat{y})$ does not survive some other scenario σ' then the cut added in H.2 exposes some of the relationship between scenarios σ and σ'.

"Score" heuristic. As the cutting-plane algorithm runs, the extended cascade model will be repeatedly run. If some edge e frequently appears in sets $\mathcal{F}^r$, this indicates that e is "important". We could in this way assign a "score" to each edge, and use some criterion to select a subset of edges with high score.

The particular heuristic we use is as follows. Initially all edges are given a score of 0. Each time we run the extended cascade model, for each edge e that belongs to some set $\mathcal{F}^r$ we increase the score of e by $1/(t_e^2)$, where t_e is defined as in (37) (so edges that "fail" earlier are given more importance). This rules sets the scores.

The score heuristic operates by solving the 0/1 integer program $\min\{-\pi^T y \, : \, Ay \geq b\}$, where $Ay \geq b$ is the current formulation in the cutting-plane algorithm. The solution to this problem is the vector $\hat{y}$ that is input to H.1 and H.2 above.

"Deep cut" heuristic. This heuristic attempts to find a 0/1 vector that is "far" from the working formulation $Ay \geq b$ without being expensive. More precisely, suppose the working formulation has constraints $\sum_e a_{i,e}\, y_e \ \geq \ b_i$, $1 \leq i \leq H$. Recall that w_e is the cost of protecting edge e. The deep cut heuristic outputs the vector $\hat{y}$ which solves the mixed-integer program:

$$\begin{aligned} \max \quad & z \\ \text{s.t.} \quad & \sum_e a_{i,e}\, y_e \ - \ z \ \geq \ b_i \qquad 1 \leq i \leq H, \\ & \sum_e w_e\, y_e \ + \ Uz \ \leq \ 2U \\ & y \in \{0,1\}^m, \ \ z \geq 0 \end{aligned}$$

Here, U is a large number. The optimal value of z indicates a "distance" from the solution $\hat{y}$ to all the constraints, but also to the hyperplane of points with cost U. To set U in our implementation, we used the following rule. Let L be the current lower bound for our optimization problem (3.5). Then set $U = 10L$.

3.3 Implementation details and computational tests

Our implementation follows closely the algorithmic ideas described above. One adjustment concerns the optimization problems solved in the score heuristic, the deep heuristic and also in Step 1 of our cutting-plane algorithm 3.7. These are 0/1 integer programs (mixed in the deep heuristic). In our computational experience, these were fairly easy to solve using a commercial mixed-integer solver until the later iterations of the cutting-plane algorithm, where the working formulation $Ay \geq b$ may contain thousands of rows.

In our implementation, each of the mixed-integer solvers is tackled with a time limit (50 seconds) – if the time limit is exceeded then we instead solve the continuous relaxation and then round each variable y_e to 1 with probability equal to the value of y_e.

The computational tests described here were carried out on a machine with a 1.8 GHz Xeon processor, with a 1 MB cache and 2 GB of RAM. We used networks $net1$ and $net2$ and the scenario families described in Section 2.4.

Figure shows a typical evolution of our algorithm on a hard problem instance. In this figure we plot the lower and upper bounds produced by the algorithm on a problem with $\alpha = 0.5$, $\mu = 0.8$ and $R = 12$ rounds, and using random costs. This value of α is relatively small, which means that a cascade will evolve rather slowly. The impact of the relatively large choice for R is that it may take many rounds for our extended cascade model (3.6) to terminate; thus (a) each cascade simulation requires the solution of "many" (up to 12) linear programs, and (b) the structure of each a cascade is "complex", possibly rendering our cutting planes less effective.

In this run, the optimal cost equals 280.82 and is computed at iteration 4200 of the cutting plane algorithm (3.7) – the vector y^* computed in Step 1 of that iteration proves feasible. An upper bound of 313.73 is computed by our first heuristic at iteration 200; after that all attempts to improve the upper bound fail – the optimum solution is the last 0/1 vector computed in Step 1 of our cutting-plane algorithm (3.7). Even though our heuristics fail to improve upon the upper bound, part of their benefit lies in the cuts produced when running the heuristics. The total running time in this case approached 20 hours.

In Table 8 we examine the behavior of the algorithm as we change the parameter R (number of rounds) while keeping everything else fixed. Here the network is $net2$, $\alpha = 0.5$ and $\mu = 0.8$. In all these runs, the scenario set is that of all singletons.
In this table, the column headed 'support' indicates the number of edges protected in the optimal solution, and 'cost' is the cost of the optimal solution (recall that problem $net2$ has random costs). Note the growth in running time, in particular that for the case of 12 rounds – here, nearly 64,000 seconds are expended solving the 0/1 problems in Step 1 of Algorithm 3.7. Using 12 rounds allows a fair amount of complexity to emerge in a typical cascade.

In Table 9 we study network $net1$ (i.e., unit costs) under various values of α while keeping all other parameters fixed. For these instances we are using the scenario family $S7$ described in Section

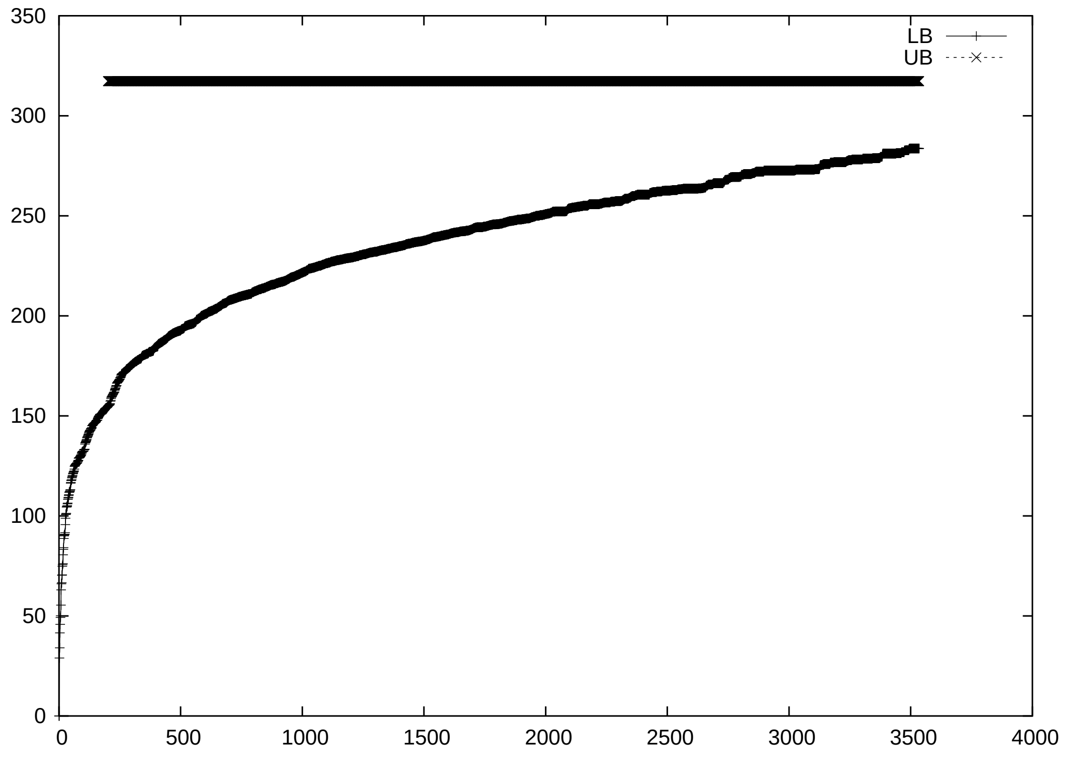

Figure 10: Evolution of bounds (final lower bound is proved optimal)

R	iterations	cuts	time (secs.)	support	cost
4	80	81	221	3	139.19
6	853	885	2172	4	216.95
8	1296	1332	2938	5	221.36
10	1988	2024	8494	7	246.32
12	4200	4258	71857	7	280.82

Table 8: Algorithm behavior as a function of maximum number of rounds

2.4, consisting of 355 scenarios of cardinality between 2 and 11. In all these runs, we assume that we have 5 rounds. The column headed 'BB time' indicates the total time spent solving the 0/1 problems in Step 1 of Algorithm 3.7. This table confirms that the larger the value of α, the more complex the problems become.

In Table 10 shows how the parameter settings indirectly affect the complexity of the problem by altering the structure of the cascade simulations. On this single run we used network *net*3, the scenario family was that of all singletons, $\alpha = 0.2$ but on the other hand we used 20 rounds. Hence the cascades should evolve relatively slowly but the large number of rounds might allow complexity to emerge.

Here, 'ave. rounds' stands for the average number of rounds until a cascade ends in Step 1b or 1c of Model (3.6) (averaged over all the cascade simulation cascades performed during the optimiza-

α	iterations	cuts	time (secs.)	BB time
0.2	9	10	391	1
0.5	53	71	1528	800
0.8	2546	2769	31014	13600
0.9	3455	3782	71905	39590

Table 9: Algorithm behavior as a function of α

ave. rounds	max	min
11.43	20	2
ave. outages	max	min
51.53	90	4

Table 10: Cascade statistics

tion), 'max' refers to the maximum and 'min' to the minimum number of rounds experienced in any cascade; 'ave. outages' refers to the average number of edges removed from the network during a cascade, and 'max' and 'min' are correspondingly defined. Note that even though α is relatively small, the large average number of rounds to termination indicates a fairly complex problem – its solution required 37099 seconds of which nearly half were spent simulating cascades.

4 The real-time problem

The two models we have studied in this paper are essentially static – their aim is to reinforce of a network so that in the event of some edge outages the network is better able to avoid a blackout.

In the real-time version of the problem we would seek to take action to stop a cascade from becoming catastrophic. One well-known but not always popular technique is that of "load-shedding" whereby demand is reduced in small amounts. At a lower level, individual power lines are equipped with equipment to turn them off if damage will result from overloads – this equipment operates automatically.

Two issues make the real-time problem difficult. First, it has been debated whether real-time information can be centralized, or if only distributed algorithms could be used. Second, the "Braess' Law" feature we have discussed above creates some unique combinatorial issues – even if all the information were available in a real-time basis the problem remains complex.

Nevertheless, a concrete problem that could be tackled from a real-time perspective is that of selectively turning off power lines (i.e., "sacrificing" part of the network) in order to arrest a cascade. Formally, we have:

Problem 4.1 *Given a network G, with weights on the edges, find a minimum-weight set $\hat{E}$ of edges such that in $G - \hat{E}$ all flows are within bounds.*

There are multiple versions of this problem, for example allowing some demand loss. However, we state without proof the following result:

Theorem 4.2 *Problem 4.1 is NP-hard.*

We take this result as theoretical confirmation that the real-time problem is difficult. In future research we plan to report on results regarding the real-time problem, in particular in connection with the adversarial and gaming versions described in Section 1.3.1.

References

[1] R. Ahuja, T. L. Magnanti, and J. Orlin. *Networks Flows: Theory, Algorithms, and Practice*, Prentice Hall (1993).

[2] G. Andersson, *Modelling and Analysis of Electric Power Systems.* Lecture 227-0526-00, Power Systems Laboratory, ETH Zürich, March 2004. Download from http://www.eeh.ee.ethz.ch/downloads/academics/courses/227-0526-00.pdf.

[3] P. Avella, S. Mattia, A. Sassano (2004). *Metric Inequalities and the Network Loading Problem.* Proceedings of the 10th International IPCO Conference. Editors: D. Bienstock, G. Nemhauser. Springer-Verlag, 16-32.

[4] F. Barahona, Network design using cut inequalities, *SIAM J. Opt.* **6** (1996) 823-837.

[5] Computational experience with an effective heuristic for some capacity expansion problems in local access networks, *Telecomm. Sys.* **1** (1993), 379-400.

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

[7] J.F. Bonnans, Mathematical study of very high voltage power networks I: The optimal DC power flow problem, *SIAM J. Optimization* **7** (1997) 979-990.

[8] J.F. Bonnans, Mathematical study of very high voltage power networks II: The AC power flow problem, *SIAM J. Applied Mathematics* **58** (1998) 1547-1567.

[9] J.F. Bonnans, Mathematical study of very high voltage power networks III: The optimal AC power flow problem. *Computational Optimization and Applications* **16** (2000) 83-101.

[10] D. Bienstock, S. Chopra, O. Günlük , C. Tsai (1998). *Minimum Cost Capacity Installation for Multicommodity Network Flows.* Mathematical Programming, 81, 177-199.

[11] B.A. Carreras, V.E. Lynch, M.L. Sachtjen, I. Dobson, and D.E. Newman, Modeling Blackout Dynamics in Power Transmission Networks with Simple Structure, 36^{th} Hawaii International Conference on System Sciences, January 2001.

[12] S. Binato, M.V.F. Pereira and S. Granville, A new Benders decomposition approach to solve power transmission network design problems, IEEE Trans. Power Systems **16**, 235-240 (2001).

[13] D. Braess, Über ein Paradox der Verkerhsplannung, Unternehmenstorchung **12**, 258-268 (1968).

[14] J. Chen, J.S. Thorp and I. Dobson, Cascading Dynamics and Mitigation Assessment in Power System Disturbances via a Hidden Failure Model, International Journal of Electrical Power and Energy Systems, **27**, 318-326 (2005).

[15] I. Dobson, B.A. Carreras, V.E. Lynch and D.E. Newman, An initial model for complex dynamics in electric power system blackouts, 36^{th} Hawaii International Conference on System Sciences, January 2001.

[16] I. Dobson, B.A. Carreras, D.E. Newman, A loading-dependent model of probabilistic cascading failure, A loading-dependent model of probabilistic cascading failure, *Probability in the Engineering and Informational Sciences* **19** 15-32 (2005).

[17] G.S. Fishman, *Monte Carlo: concepts, algorithms and applications*, Springer-Verlag (New York), 1996.

[18] X. Fito, L. Pilotto, N. Martins, A. Carvalho and A. Bianco, *Brazilian Defense Plan Against Extreme Contingencies*, CEPEL, Rio de Janeiro, Brazil (2000). Download from: http//www.transmission.bpa.gov/orgs/opi/Power_Stability/EmergencyControlBrazil.pdf.

[19] ILOG CPLEX, Incline Village, NV.

[20] *Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations*, U.S.-Canada Power System Outage Task Force, April 5, 2004. Download from: https://reports.energy.gov.

[NW88] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, Wiley, New York (1988).

[21] G.C. Oliveira, S. Binato, L. Bahiense, L. Thomé and M.V. Pereira, Security-constrained transmission planning: A mixed-integer disjunctive approach. Optimization Online (August, 2004).

[22] G.B. Sheble and G.N. Fahd, Unit Commitment Literature Synopsis, IEEE Trans. Power Systems 9 (1994), 128-135.

[23] F. Vandenberghe, *Lessons and Conclusions from the 28 Septembaer 2003 Blackout in Italy*. IEA Workshop, March 2004. Download from: http://www.iea.org/dbtw-wpd/textbase/work/2004/transmission/vandenberghe.pdf.

[24] http://www.ee.washington.edu/research/pstca