Fast Solution of Load Shedding Problems via a Sequence of Linear Programs

Harish S. Bhat, Garnet J. Vaz, and Juan C. Meza
Applied Mathematics Unit
University of California, Merced
Merced, CA USA
email: hbhat@ucmerced.edu

***Abstract*—Given a power network consisting of nodes (generators/loads) and edges (lines), there exist a set of constraints that must be satisfied in order for the system to be operational. When one or more power lines are cut, the bus phases and load/generator power values may need to be altered in order to restore the system to operation. The load shedding problem is to find the smallest adjustment to the loads that achieves this restoration. In this work, we show how to solve this nonlinear optimization problem by solving a sequence of linear programs. On random graphs with 1500 edges, we find that our method is at least 40 times faster than competing nonlinear optimization methods. We show that our method is capable of solving load shedding problems for a real graph with 19840 edges, and that the method scales with an $\mathrm{O(n^2)}$ running time where n is the number of edges. For real subgraphs with hundreds of edges, we are able to rapidly solve all possible load shedding problems in which at most two lines are deleted. This work takes a first step towards a scalable load shedding algorithm capable of handling large networks that will arise in the future.**

***Keywords*-load shedding; power network; nonlinear optimization; sequential linear programming**

I. Introduction

Electric power grids are commonly modeled using graphs that may contain thousands of edges and nodes. The nodes in these graphs represent generators and loads, while the edges represent power lines. Given such a graph, one seeks algorithms to analyze the power system's vulnerability and to improve its robustness.

The proliferation of smart grid technologies gives us the ability to collect data on power consumption and generation on small length scales such as individual buildings and rooms. This will only increase the size of power networks that must be modeled and optimized. When we examine algorithms for analyzing network vulnerability, there is a large gap between the capabilities of current methods and the massive network sizes that we may expect to confront in the future.

Here we focus on the *load shedding problem*: if we cut a set of lines from the power network, what is the minimal load that must be shed from the system to restore its operation? We are interested in developing an algorithm for solving this problem that scales well to large networks.

Viewed as an optimization problem, the load shedding problem is a nonlinear program. In this paper, we reformulate the nonlinear program as a sequence of linear programs (SLP). This reformulation enables us to solve the load shedding problem more rapidly than if we apply off-the-shelf interior point or sequential quadratic programming codes to the original nonlinear program.

We also analyze real data obtained from the Western Electricity Coordinating Council (WECC). This data set, a proposed power grid model for the year 2021, consists of a network of 17452 nodes and 19840 edges. First, we analyze two subgraphs with hundreds of edges; for these subgraphs, we exhaustively solve load shedding problems for *all possible* deletions of one and two edges. These collections of thousands of nonlinear programs are solved on the order of minutes using a single laptop computer. In this way, we are able to rapidly assess the robustness of these subgraphs.

Second, we show performance results using a Python (Numpy/Scipy) implementation of the SLP approach. The results show that it is possible to solve one load shedding problem on the entire WECC graph in approximately 10^3 seconds on a single laptop computer. By considering subgraphs of the WECC graph, we also show that the load shedding running time scales as $O(n^2)$ where n is the number of edges in the graph.

Note that the presence of the sine nonlinearity in our load shedding problem is due to the fact that we analzye a dynamic/active or alternating current (AC) formulation of the problem [1], [2]. Linear versions of the problem that assume a direct current (DC) or static formulation have been solved by algorithms that scale well to graphs with thousands of nodes and edges [3].

More generally, linear programming has been successfully applied to solve optimization problems with millions of variables and constraints. We expect that the SLP formulation of the load shedding problem will enable, in forthcoming work, the analysis of the entire WECC graph and other large graphs that arise.

II. Model

We model a power network using a directed graph with m buses (nodes) and n lines (edges). Let A be the $n \times m$ arc-incidence matrix for this graph, and let B be a diagonal $n \times n$ matrix containing all line admittances. For each line, we define γ_i to be 0 (respectively, 1) if the line is in service

(respectively, out of service). Then let Γ be the $n \times n$ diagonal matrix defined by $\Gamma = \mathrm{diag}(1-\gamma)$. We let θ denote the $m \times 1$ vector of bus phases, while P denotes an $m \times 1$ vector of power injections. Let

$$F(A, B, \Gamma, \theta, P) = A^T B \Gamma \sin(A\theta) - P, \tag{1}$$

where $\sin$ is applied component-wise to the $n \times 1$ vector $A\theta$. Then we refer to

$$F(A, B, \Gamma, \theta, P) = 0 \tag{2}$$

as the *power flow equations*. Note that in the above formulation, we have made the following assumptions: (i) the system is lossless, (ii) bus voltages are fixed, rendering unnecessary the reactive power constraints, and (iii) bus phases are determined completely by the active power constraints.

For more information on this formulation, see [2], [4].

A. Load Shedding

We assume that the power flow equations (2) are satisfied when Γ is the identity matrix, i.e., when all lines are in service. When at least one line has been cut, Γ contains at least one zero on its diagonal, and our old values of θ and P may not satisfy (2).

We identify the positive (generator) components of P as the subvector P^g; similarly, we identify the nonpositive (load) components of P as the subvector P^l. Let Z^l and Z^g be vectors of the same lengths as P^l and P^g, respectively. We think of Z^l (respectively, Z^g) as the adjustment to the power P^l (respectively, P^g).

Let $e = [1, \ldots, 1]^T$ denote a column vector of all ones where T denotes transpose. In what follows, we will take e to be of the appropriate size based on the vector/matrix it multiplies. We can now define the load shedding problem, as in [2]:

$$\min_{Z,\theta} e^T Z^l \tag{3a}$$

$$F(A, B, \Gamma, \theta, P + Z) = 0 \tag{3b}$$

$$P^l \le P^l + Z^l \le 0, \quad 0 \le P^g + Z^g \le P^g \tag{3c}$$

$$-\pi/2 \le A\theta \le \pi/2. \tag{3d}$$

The load shedding problem (3) seeks the smallest total adjustment to the load power vector such that the power flow equations (3b) are satisfied for the adjusted power vector $P + Z$. The adjustments are constrained in (3c) so that nodes that were loads and generators retain their identities as, respectively, loads and generators. The final constraint (3d) ensures steady-state stability of the system.

In (3c), the load/generator adjustments are also constrained so that none of the components of the respective vectors move further away from zero. While it is natural to think of cutting system loads to restore system operation, the idea of cutting system generation may seem counterintuitive. However, because each row of A contains precisely one $+1$ and one -1, we have $Ae = 0$. Therefore, multiplying (2) on the left by e^T and using the fact that (2) is satisfied by P when $\Gamma = I$, we see that if $P + Z$ satisfies (2), then $e^T Z^g = -e^T Z^l$. In short, in this lossless system, cutting load power is equivalent to cutting generator power.

B. Transforming the Nonlinear Problem

First we transform the nonlinear problem (3) into a different nonlinear problem that is easier to solve using a sequence of linear programs. Using definition (1), constraint (3b) can be solved for Z:

$$Z = A^T B \Gamma \sin(A\theta) - P. \tag{4}$$

Let m_g and m_l denote the number of entries of the vectors P_g and P_l, respectively. Clearly $m_g + m_l = m$. Let us now define two matrices, ϕ^g of size $m_g \times m$ and ϕ^l of size $m_l \times m$. For each j, the j-th row of ϕ^g (respectively, ϕ^l) has only one nonzero entry, which is an entry of 1 in the column whose index corresponds to the j-th positive (respectively, nonpositive) entry of P. Then it is clear that

$$\phi^g P = P^g \quad \text{and} \quad \phi^l P = P^l.$$

The same equalities hold with all instances of P replaced by Z, of course. Combining this with (4), we have

$$Z^g = \phi^g Z = \phi^g A^T B \Gamma \sin(A\theta) - P^g \tag{5a}$$

$$Z^l = \phi^l Z = \phi^l A^T B \Gamma \sin(A\theta) - P^l \tag{5b}$$

Next, note that (3d) implies that $A\theta = \sin^{-1}(s)$ for some vector s, where $\sin^{-1}$ is applied component-wise to s. Therefore, we define $s = \sin(A\theta)$. Applying this definition of s to (5) and then substituting in (3), we obtain an equivalent nonlinear program:

$$\min_{s} e^T \phi^l A^T B \Gamma s \tag{6a}$$

$$P^l \le \phi^l A^T B \Gamma s \le 0, \quad 0 \le \phi^g A^T B \Gamma s \le P^g \tag{6b}$$

$$-1 \le s \le 1 \tag{6c}$$

$$\exists \theta \in \mathbb{R}^m \text{ such that } s = \sin(A\theta) \tag{6d}$$

We now work on the last constraint, still assuming (3d):

$$\begin{aligned} &\exists \theta \in \mathbb{R}^m \text{ such that } s = \sin(A\theta) \\ &\quad \iff \exists \theta \in \mathbb{R}^m \text{ such that } \sin^{-1}(s) = A\theta \\ &\quad \iff \sin^{-1}(s) \in \text{Image}(A) \\ &\quad \iff \sin^{-1}(s) \perp \text{Kernel}(A^T), \end{aligned}$$

where the last line follows from the Fredholm Alternative. Let $\mathcal{N}$ denote a matrix whose columns form an orthonormal basis for the kernel of A^T. Then the final condition can be written $\mathcal{N}^T \sin^{-1}(s) = 0$, leading to the equivalent

nonlinear program

$$\min_s e^T \phi^l A^T B \Gamma s \tag{7a}$$

$$P^l \leq \phi^l A^T B \Gamma s \leq 0, \quad 0 \leq \phi^g A^T B \Gamma s \leq P^g \tag{7b}$$

$$-1 \leq s \leq 1 \tag{7c}$$

$$\mathcal{N}^T \sin^{-1}(s) = 0 \tag{7d}$$

Note that all of the nonlinearity in the problem has been concentrated in the equality constraint (7d). The remaining problem (7a-7c) is a linear program.

C. Sequence of Linear Programs: First Approach

We now describe the linear program that must be solved to obtain the next iterate s^{i+1}, assuming we have already obtained s^i. We write $s^{i+1} = s^i + \Delta s$ and linearize (7d) about s^i. Starting with $\mathcal{N}^T \sin^{-1}(s^{i+1}) = 0$, linearization yields

$$\mathcal{N}^T \sin^{-1}(s^i) + \mathcal{N}^T \operatorname{diag}\left[\left(1 - (s^i)^2\right)^{-1/2}\right](\Delta s) = 0.$$

Using $\Delta s = s^{i+1} - s^i$, we write this as a linear equality constraint on s^{i+1}: $T^i s^{i+1} = \xi^i$, where $T^i = \mathcal{N}^T \operatorname{diag}\left[\left(1 - (s^i)^2\right)^{-1/2}\right]$ and $\xi^i = T^i s^i - \mathcal{N}^T \sin^{-1}(s^i)$. Note that in these two expressions, the quantity s^i is a vector and operations such as squaring must be interpreted component-wise. The $(i+1)$-st linear program in the sequence is then:

$$s^{i+1} = \mathcal{L}^{i+1}(A, B, \Gamma, P) = \arg\min_s e^T \phi^l A^T B \Gamma s \tag{8a}$$

$$P^l \leq \phi^l A^T B \Gamma s \leq 0 \tag{8b}$$

$$0 \leq \phi^g A^T B \Gamma s \leq P^g \tag{8c}$$

$$-1 \leq s \leq 1 \tag{8d}$$

$$T^i(s^i) s = \xi^i(s^i). \tag{8e}$$

We have written (8e) to emphasize that both T^i and ξ^i depend on the previous solution s^i.

We set $s^0 = \sin(A\theta)$, which would be the solution of (6) if Γ were the identity matrix. Armed with s^0, we can then solve the sequence of linear programs $\mathcal{L}^1$, $\mathcal{L}^2$, etc. We terminate the sequence when $\|\mathcal{N}^T \sin^{-1}(s^i)\|_2 < \epsilon$; in what follows, we set $\epsilon = 10^{-6}$ regardless of system size.

Note that:

- Equation (8e) is simply Newton's method applied to the nonlinear constraint (7d). Because the rest of the optimization problem was a linear program, the convergence of the sequence will largely be governed by the convergence of the Newton approximation (8e).
- It is clear that the matrix T^i will blow up when $s^i = \pm 1$. To stop this from happening, we replace (8d) with $-1+\delta \leq s \leq 1-\delta$ for a fixed $\delta > 0$. Results presented below are for $\delta = 10^{-9}$.
- Our contribution above is in the transformation of the original nonlinear program (3) into the problem (7), which is close to a linear program. There exists a substantial literature on sequential/successive linear programming (SLP), in which Newton's method is often applied to nonlinear constraints. Also typical is the use of additional constraints to bound the Newton step and thereby improve convergence [5], [6].
- In our present version of the algorithm/code, we do not place any bounds on the Newton step, and we observe convergence in a small number of iterations for many real and simulated problems.

D. Sequence of Linear Programs: Second Approach

One of the deficiences of the method we just derived is that even if A is extremely sparse, the null space matrix $\mathcal{N}$ will be dense. This yields a dense system of equality constraints in the linear program described above. As we suspect that this leads to suboptimal performance, we now return to (6). Let us treat both s and θ as decision variables. Assume we have already solved for the i-th iterate (s^i, θ^i). Define the function $\varphi : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$ by

$$\varphi(s, \theta) = s - \sin(A\theta) \tag{9}$$

Then we can express (6d) by writing $\varphi(s, \theta) = 0$. Applying Newton's method to this equation, we obtain the equation for the Newton step:

$$\underbrace{\begin{bmatrix} -I & \operatorname{diag}(\cos(A\theta^i))A \end{bmatrix}}_{-D\varphi(s^i,\theta^i)} \begin{bmatrix} s^{i+1} - s^i \\ \theta^{i+1} - \theta^i \end{bmatrix} = \underbrace{s^i - \sin(A\theta^i)}_{\varphi(s^i,\theta^i)}, \tag{10}$$

where $D\varphi$ is the Jacobian of φ. We rearrange this and formulate the following linear program:

$$\begin{bmatrix} s^{i+1} \\ \theta^{i+1} \end{bmatrix} = \mathcal{L}^{i+1}(A, B, \Gamma, P) = \arg\min_{s,\theta} e^T \phi^l A^T B \Gamma s \tag{11a}$$

$$P^l \leq \phi^l A^T B \Gamma s \leq 0 \tag{11b}$$

$$0 \leq \phi^g A^T B \Gamma s \leq P^g \tag{11c}$$

$$-1 + \delta \leq s \leq 1 - \delta \tag{11d}$$

$$(-\pi/2)(1-\delta) \leq A\theta \leq (\pi/2)(1-\delta) \tag{11e}$$

$$-D\varphi(s^i, \theta^i) \begin{bmatrix} s \\ \theta \end{bmatrix} = \operatorname{diag}(\cos(A\theta^i))A\theta^i - \sin(A\theta^i) \tag{11f}$$

$$-2\pi \leq \theta \leq 2\pi \tag{11g}$$

We set $(s^0 = \sin(A\theta), \theta^0 = \theta)$, which would be the solution of (7) if Γ were the identity matrix. Starting from (s^0, θ^0), we then solve the sequence of linear programs $\mathcal{L}^i$ for $i = 1, 2, \ldots$. We terminate when $\|\varphi(s^i, \theta^i)\|_2 < \epsilon$, where again we set $\epsilon = 10^{-6}$ regardless of system size.

Note that the derivative matrix $D\varphi(s^i, \theta^i)$ becomes singular when $A\theta^i = \pm\pi/2$. The presence of the parameter $\delta \geq 0$ in (11d-11e) is designed to stop this from happening. In the Matlab tests of this algorithm described below for randomly generated problems, we run with $\delta = 0$. However,

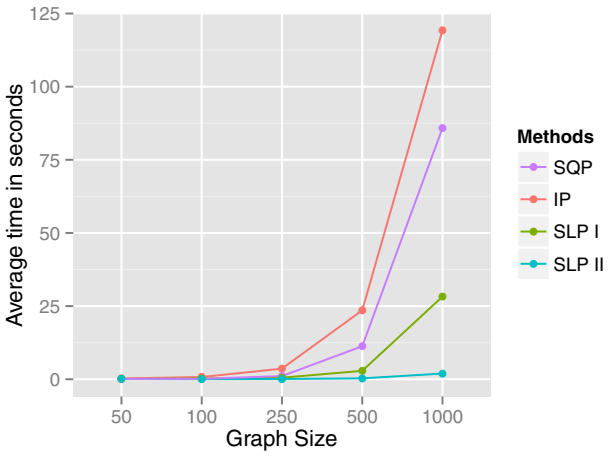

Figure 1. Average running times on randomly generated load shedding problems for the three methods under comparison. The methods SLP I and SLP II refer, respectively to the methods presented in Subsections II-C and II-D. For the largest graphs we tested, SLP II is 44 times faster than sequential quadratic programming (SQP) and 61 times faster than interior point (IP). Note that graph size refers to the number of nodes, and that each running time is the average across 60 runs.

when analyzing real data with Python in Section IV-B, we set $\delta = 10^{-6}$.

III. Comparison of Algorithms

We have written Matlab codes that implement our SLP approaches to the load shedding problem. In what follows, we use SLP I and SLP II to denote, respectively, the algorithms described in Sections II-C and II-D.

Each linear program in the sequence is solved by IBM ILOG CPLEX 12.5.1. When possible, basis information from the previous linear programming solution is used to initialize the next linear program.

We test this Matlab implementation against two algorithms for directly solving the nonlinear program (3) using the `fmincon` function in Matlab's Optimization Toolbox. The two algorithms are `interior-point` (IP) and `sqp` (SQP, or sequential quadratic programming). For both of these algorithms, we provide analytic gradients of both the objective function (3a) and the power flow constraint (3b).

In order to compare our method against the two Matlab algorithms, we must generate random test problems. For a desired number of nodes and edges (m, n_d), we generate a random Erdös-Rényi graph on m nodes with edge probability $p = 2n_d/(m^2-m)$ chosen such that the number of edges equals n_d in expectation. With this graph, we randomly orient each edge to generate the oriented arc-incidence A. We let n denote the actual number of edges.

The diagonal entries of the admittance matrix B consist of n random samples from the uniform distribution on $[0.8, 1.2]$. The initial bus phase vector θ is found as follows: (i) generate a vector ψ of m random samples uniformly from $[-\pi/4, \pi/4]$, (ii) solve a linear program in θ with zero objective function and the constraints $\psi - \pi/4 \leq A\theta \leq \psi + \pi/4$ together with $0 \leq \theta \leq 2\pi$. Armed with A, B, and θ, we compute P using (2) with $\Gamma = I$, the identity matrix. This yields a natural random partition of nodes as generators and loads corresponding to positive and nonpositive entries of P, respectively.

Table I
For randomly generated load shedding problems on graphs of each designated size, we record the running time and number of iterations required for the SLP II algorithm to converge. All results are averaged across 60 runs.

Graph Size (m, n_d)	Running Time (seconds)	Number of Iterations
$(50, 75)$	0.08901158	8
$(100, 150)$	0.04945610	3.05
$(250, 350)$	0.07742607	3.25
$(500, 700)$	0.29740150	3.38
$(1000, 1500)$	1.93605200	3.13

Given this random instance of the problem, we knock out two lines and solve the load shedding problem using (i) both SLP approaches, (ii) `fmincon` with the `interior-point` algorithm, and (iii) `fmincon` with the `sqp` algorithm. Note that methods (i) and (ii) make use of sparse linear algebra, while method (iii) does not. We attempted to use `fmincon` with the `active-set` algorithm, but too many instances of the problem failed to converge to provide useful results.

We run the tests for graphs of size $(m, n_d) = (50, 75)$, $(100, 150)$, $(250, 350)$, $(500, 700)$, and $(1000, 1500)$. Each test was run 60 times before averaging the running times.

The results, plotted in Figure 1, show that both SLP approaches are faster than the competing nonlinear programming approaches. The fastest algorithm overall is clearly SLP II. In Table I, we record for SLP II the running time and the average number of iterations for convergence. Roughly speaking, to obtain convergence with SLP II at the largest graph size, we need to solve about 3 or 4 linear programs, which takes a total of less than 2 seconds on a single laptop.

In Table II, we record the time required by the four algorithms relative to the time required by SLP II. The results show that the gap between the algorithms' performance becomes more appreciable as the graph size increases. Note that for the largest size graphs, SLP II is more than 14 times faster than SLP I, showing that a large system of sparse equality constraints yields better performance than a smaller dense system.

Finally, we note that the solutions found by SLP II are comparable in quality to those found by SQP. In the worst case, the final objective function value for SLP II is less than 0.0031% greater than that produced by SQP. Additionally, in the worst case we observed for SLP II, the maximum constraint violation was $\sim 10^{-9}$. Note that the final constraint violation for SLP II can be made smaller by decreasing the termination criterion ϵ described above, and

Table II
FOR RANDOMLY GENERATED LOAD SHEDDING PROBLEMS ON GRAPHS OF EACH DESIGNATED SIZE, WE RECORD THE RUNNING TIME OF EACH ALGORITHM RELATIVE TO THAT OF SLP II. ALL RESULTS ARE AVERAGED ACROSS 60 RUNS.

Graph Size (m, n_d)	SQP	IP	SLP I	SLP II
$(50, 75)$	1.12	2.95	1.17	1
$(100, 150)$	2.88	15.31	4.75	1
$(250, 350)$	13.82	46.70	6.69	1
$(500, 700)$	37.91	79.22	9.65	1
$(1000, 1500)$	44.34	61.61	14.58	1

by changing the default tolerances for CPLEX; the results quoted above are for $\epsilon = 10^{-6}$. We leave further exploration of this for future work.

IV. REAL DATA

A. Exhaustive Analysis

Starting with the large WECC network described in Section I, we extract subgraphs corresponding to:

- *CFE:* The northern Baja California portion of Mexico's Comisión Federal de Electricidad (196 nodes, 216 edges).
- *IID:* The Imperial Irrigation District (114 nodes, 128 edges).

For these two subgraphs, we use our Matlab/CPLEX implementation of SLP I to solve *all possible* load shedding problems involving deleting either $k = 1$ or $k = 2$ lines. For the CFE subgraph, this corresponds to a total of 23436 load shedding problems (solved in ≈ 75 minutes), while for the IID subgraph, the total is 8256 (solved in ≈ 15 minutes).

Note that all running times reported in Sections III and IV are for calculations performed on a single laptop computer. This computer has a 2.2GHz Intel Core i7-2670QM chip with 4 GB of memory. We used Matlab R2012a running on the Ubuntu 12.04 operating system.

Let *severity* denote the value of the objective function (3a) at the computed optimal solution. This value is how much load power must be shed by the system to restore operability when the corresponding lines are cut. For both subgraphs, we sort all solutions by severity s and plot in Figure 2 the fraction of total load shedding problems whose solution has severity at least equal to s. Clearly, when $s = 0$, the fraction must be unity; also, the fraction cannot increase as the severity increases.

The plot shows the probability that a randomly chosen cut (of at most two lines) will force a load shed of at least s. These plots give us an overall view of the relative robustness of the two networks. Note that the vertical axes are the same for the two plots. The larger area under the curve for the IID subgraph indicates that there is a higher probability (relative to the CFE subgraph) that random line cuts will lead to small blackouts. Since the horizontal axes differ for the two subgraphs, we see that in absolute terms, it is possible for a particular two-line cut to cause a more severe blackout for the CFE subgraph than for the IID subgraph.

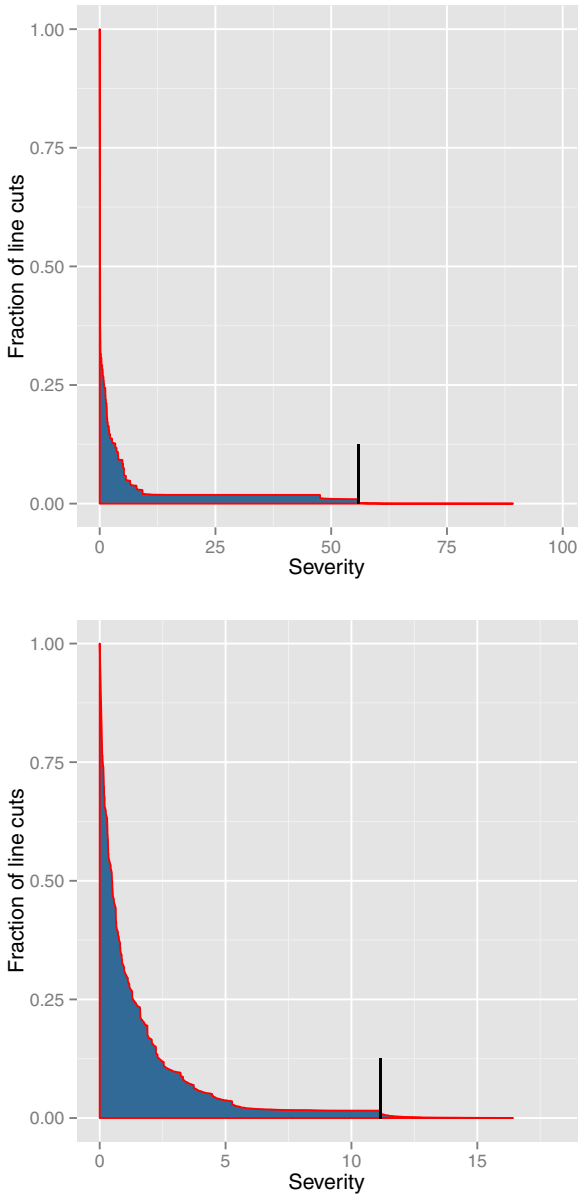

Figure 2. The top (CFE) and bottom (IID) panels correspond to two subgraphs of the WECC power grid. For both subgraphs, we plot the fraction of all $k = 1$ and $k = 2$ line cuts that result in a load shed greater than or equal to the indicated severity value. We use black vertical lines to indicate the maximum severity $k = 1$ solutions: 55.84 and 11.16 for the left and right plots, respectively. As the maximum severity $k = 2$ solutions are far to the right of the black lines, we see that computing the $k = 2$ solutions is important. Note that all computations were carried out on a single laptop computer using a Matlab implementation of SLP I.

We also plot in Figure 2 black vertical lines corresponding to the maximum severity $k = 1$ solutions. The portion of the curve to the right of the black line corresponds to $k = 2$ cuts with greater severity than the most severe $k = 1$ cut. For

the CFE and IID subgraphs, respectively, this corresponds to 220 and 76 two-line cuts. Together with the fact that the $k = 2$ solutions are far to the right of the black lines, this indicates that computing the $k = 2$ solutions is important.

B. Scalability

Next we describe results for a Python implementation of SLP II. We call CPLEX using its Python API in order to solve each linear program in the sequence. This Python/CPLEX implementation is able to solve load shedding problems for the entire WECC network.

The calculations in this section were carried out using a single laptop computer. This computer has a 1.6GHz Intel Core i7-Q720 chip with 4 GB of memory. We used Python 2.7.3 on the Ubuntu 12.10 operating system with Numpy 1.6.2 and Scipy 0.12.0.

Starting with the entire WECC graph with 19840 edges, we randomly select subgraphs with 620, 1240, 2480, 4960, and 9920 edges. For each such subgraph, we knock out two lines, solve the load shedding problem, and record the running time in seconds. All results are averaged over 10 such runs.

In Figure 3, we plot the running time T versus the number of edges n. The axes on this plot have been scaled logarithmically with base 2. The least-squares line of best fit therefore indicates $\log_2 T \approx -20.2432 + 2.097 \log_2 n$, providing empirical evidence that the running time for SLP II is $O(n^2)$. For the line of best fit, the adjusted R^2 value is 0.9865.

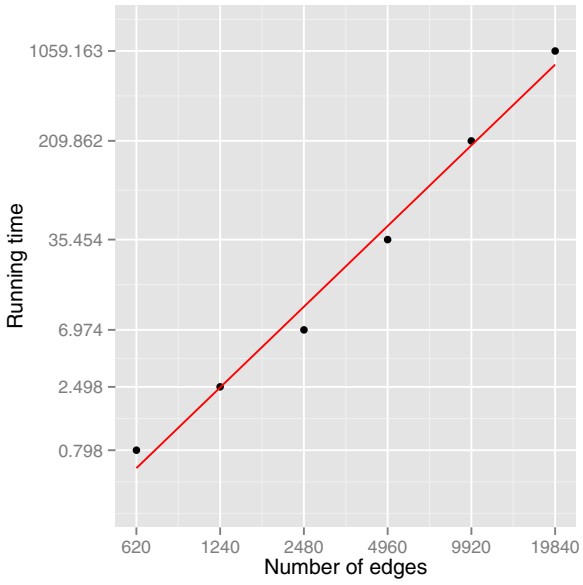

Figure 3. We present a $\log_2$-$\log_2$ plot of running time (seconds) versus number of edges n for a Python implementation of the SLP II load shedding algorithm. The raw data points (black) are averaged over 10 runs. The least-squares line of best fit has slope 2.097, indicating that this method has $O(n^2)$ running time.

V. Conclusion

The SLP II method shows much promise for future work. If the $O(n^2)$ scaling continues for larger graphs, it will be possible to apply this method in a high-performance computing context, and thereby solve much more massive load shedding problems than those considered here.

To plot curves such as in Figure 2 for larger graphs, we plan to couple the SLP II method with a method for intelligently sampling the space of all $k = 2$ cuts [7], [8].

We have already begun the process of developing a large-scale version of our code that can leverage cluster/cloud resources. In this way, we plan to use the SLP II algorithm to analyze the $k = 2$ contingencies for the full WECC network.

Acknowledgment

This work was supported by the U.S. Department of Energy (Contract No. DE-AC02-05CH11231, Subaward 7041635). The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

References

[1] D. Bienstock and A. Verma, "The $N - k$ problem in power grids: New models, formulations, and numerical experiments," *SIAM Journal on Optimization*, vol. 20, no. 5, pp. 2352–2380, 2010.

[2] A. Pinar, J. Meza, V. Donde, and B. Lesieutre, "Optimization strategies for the vulnerability analysis of the electric power grid," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1786–1810, 2010.

[3] J. Salmeron, K. Wood, and R. Baldick, "Worst-case interdiction analysis of large-scale electric power grids," *IEEE Transactions on Power Systems*, vol. 24, no. 1, pp. 96–104, 2009.

[4] J. J. Grainger and W. D. Stevenson, *Power System Analysis*. New York: McGraw-Hill, 1994.

[5] F. Palacios-Gomez, L. Lasdon, and M. Engquist, "Nonlinear optimization by successive linear programming," *Management Science*, vol. 28, no. 10, pp. 1106–1120, 1982.

[6] J. Zhang, N.-H. Kim, and L. Lasdon, "An improved successive linear programming algorithm," *Management Science*, vol. 31, no. 10, pp. 1312–1331, 1985.

[7] V. Donde, V. Lopez, B. Lesieutre, A. Pinar, C. Yang, and J. Meza, "Identification of severe multiple contingencies in electric power systems," *IEEE Transactions on Power Systems*, vol. 23, pp. 406–417, 2008.

[8] A. Kumar and P. Tripathi, "Recognition of critical "n-2" contingencies in power system using non-linear optimization," in *Power India Conference, 2012 IEEE Fifth*, 2012, pp. 1–6.