Optimal Power Grid Attack

Sean Harnett

May 11, 2011

Abstract

I study a continous formulation of the following problem: given a power grid, find a small set of lines whose failure will cause maximum damage. Instead of removing lines from the grid, I consider increasing their impedance continuously, and attempt to maximize the resulting disruption to stable voltages. For all test cases, I show that a fast and simple first-order optimization method finds the same set of lines as more sophisticated techniques.

1 Introduction

In the study of power grid vulnerabilities, the so-called “N - k” problem asks: is there a set of k power lines whose simultaneous outage will result in system failure? Of primary interest are small values of k; for large k (greater than 5 or so) the answer is “yes”, and for $k = 1$ simple enumeration will work [1].

For large grids, the combinatorial explosion makes the discrete problem intractable. One way to bypass this complexity is to consider a related continuous problem: instead of looking at power line failures, let the impedances of the lines be modified. To simulate the idea of a small k, impose constraints on how much the impedances can be changed. The hope is that this modified problem will have a combinatorial solution – the optimal attack will be to greatly modify the impedance of only a small number of lines, while perhaps slightly modifying many more lines. Later we'll see that this is indeed the case.

Power grids can fail in a number of ways. One is voltage instability, where the bus voltages stray beyond certain limits. The usual way for this to occur is "voltage collapse" – voltages progressively

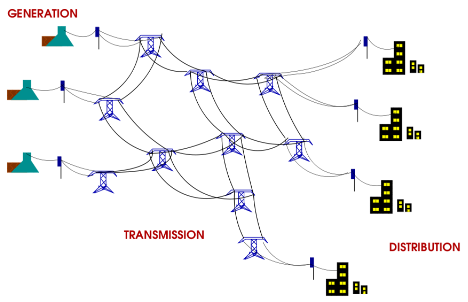


Figure 1: The power transmission network. It can be thought of as a graph, with generators and distribution substations as the vertices (called *buses*) and the transmission lines as edges.

decline due to an imbalance of reactive power supply and demand, eventually leading to blackout. It is also bad for voltages to be too high, though this is less common in practice.

The optimization problem is as follows: find the set of changes in transmission line impedances, subject to constraints, that maximizes voltage instability. The objective, voltage instability, is a non-convex black box which requires solving the non-linear power flow equations at each evaluation. In particular, this creates difficulties for finding derivatives.

2 The AC Power Flow Model

I present a short summary of the derivation of the power flow equations [2]. To find the bus voltages (and hence compute the voltage disruption), the AC power flow model is used. The quantities of interest in the model at each bus i are as follows:

- known - transmission line parameters (impedance, capacitance, etc)
- known - complex power demands $S_i = P_i + jQ_i$ (real + reactive) at each demand bus
- unknown - complex voltage $V_i = |V_i|e^{j\theta_i}$ at each demand bus
- known - real power supply P_i and voltage magnitude $|V_i|$ at each generator bus
- unknown - reactive power Q_i and voltage angle θ_i at each generator bus

We obtain the unknowns by accounting for the total current and power at each node. Specifically, the current injected into the system at bus i is the difference between the current generated and current demanded (Kirchoff's law):

$$I_i = I_{G_i} - I_{D_i} = \sum_{k=1}^{n} Y_{ik} V_k$$

where n is the number of buses and Y is the admittance matrix, whose diagonal terms Y_{ii} are the sums of the admittances of all connections to bus i and off-diagonal terms Y_{ik} are the opposite of the admittance between buses i and k. The admittance matrix captures the physical parameters of the transmission lines. [1]

Similarly, we have for the power injected into the system at bus i (conservation of complex power):

$$S_i = S_{G_i} - S_{D_i} = V_i I_i^* = V_i \sum_{k=1}^{n} Y_{ik}^* V_k^*$$

The real and imaginary parts of this equation must match, so for each demand bus (where S_i is entirely known and we need to find V_i) we have two quadratic equations. For each generator bus, we need only match the real part, $P_i = \mathrm{Re}\left[V_i \sum_{k=1}^{n} Y_{ik}^* V_k^*\right]$ by finding the right θ_i's (the $|V_i|$'s are known). We can then compute the unknown Q_i if desired.

Assume the first bus is a generator, let $\theta_1 = 0$ and measure all other angles relative to this bus, called the *slack bus*. We thus ignore this equation and are left with a system of $n+m-1$ quadratic equations and unknowns, where m is the number of demand buses.

In general this system will not have a unique solution. Under normal practical conditions, however, there is exactly one realistic solution. It is the solution where all the $|V_i|$ are large and approximately

[1] Admittance is the inverse of impedance: $Y = Z^{-1} = (R + jX)^{-1}$, where R is resistance and X is reactance.

equal (the problem is usually scaled so that $|V_1| = 1$), and the θ_i all small. Thus the problem is usually amenable to solution by Newton's method, with initial guess $|V_i| = 1$ and $\theta_i = 0$ for all i.

In summary, to find the voltages V_i for each bus, we apply Newton's method to the equations

$$\left(V_i \sum_{k=1}^{n} Y_{ik}^* V_k^* \right) - S_i = 0, \text{ for each bus } i \tag{1}$$

The software library *MATPOWER* [3] includes a number of MATLAB scripts to solve power flow problems. I used it extensively in this project.

3 Formulation of the Optimization Problem

We want to maximize "voltage instability". As mentioned above, the system is stable when all (scaled) voltages are approximately equal to one in magnitude. This suggests the following objective function:

$$f(x) = \sum_{i=1}^{n} (1 - |V_i|)^2$$

Here x denotes the vector of impedance multipliers on the transmission lines. Note that the objective is not an explicit function of x; changes in x affect the admittance matrix Y, which in turn affects the solution to the power flow equations (1), whose solutions V_i finally directly change the objective f.

To capture the idea of only removing a small number of lines, impose the following conditions on x:

$$\begin{aligned} 1 \leq x_i \leq \alpha \quad & \text{for each line } i \\ \sum_{i=1}^{\#lines} (x_i - 1) \leq \beta & \end{aligned} \tag{2}$$

In words, the impedance for each line can be multiplied up to some maximum amount α, and the total amount of multiplication across all lines is at most β. By varing these two parameters, one can attempt to simulate different values of k in the $N - k$ problem. Thus we have a non-linear, non-convex maximization problem with linear constraints:

$$\max_{x} f(x)$$

$$\text{subject to } Ax \leq b$$

4 Solving the Optimization Problem

My first attempt at a solution to the optimization problem is via the reduced gradient method, aka Frank-Wolfe algorithm. It is a simple way to accomodate constraints while using steepest ascent. Essentially, instead of taking a step in the direction of the gradient $\nabla f(x_k)$, which might lead to a constraint violation, we take a step towards the point y_k on the constraint boundary which maximizes the inner product with the gradient. In other words, we maximize the first-order Taylor expansion about the current point:

$$\max_{y \in S} f(x_k) + \nabla f(x_k)^T (y - x_k) \tag{3}$$

where S is the feasible region. This is a linear program which produces the extreme point y_k. Define the search direction $p_k = y_k - x_k$ and do a line search to find the next iterate: $x_{k+1} = x_k + \alpha p_k$.

4.1 Estimating ∇f

Since explicit derivatives of f are unavailable, I approximate it via finite differences. For large grids, this estimation of the gradient is by far the most costly part of the optimization process, which suggests the use of a “true” derivate free method, such as Nelder-Mead.

I looked into this briefly, using a simple extension of MATLABs fminsearch called fminsearchbnd, available in the file exchange of the Mathworks website. It implements fminsearch (Nelder-Mead) with simple bound constraints by internally transforming the variables. I ignored the sum constraint temporarily. I found that this method was far, far slower (something like 100 times) than estimating the gradient, and stopped there. Perhaps a better implementation, or a different DFO algorithm altogether would produce better results.

The finite difference estimation of ∇f is fairly straightforward; I used forward differences and the SAS/OR User’s Guide for guidance [4].

$$(\nabla f(x))_i \approx \frac{f(x + he_i) - f(x)}{h}$$

I used one trick to speed things up. When solving the non-linear system (1) to evaluate f, instead of performing pure Newton’s method for each component of the gradient, I find the LU factors of the Jacobian only once, at $f(x)$. Then to find $f(x + he_i)$, I simply solve with the same factors repeatedly. This typically requires an additional iteration or two for Newton to converge for each i, but ultimately speeds things up by about an order of magnitude, as the LU factorization is very expensive. This is a significant savings; for my largest test case of 15029 buses, even this improved gradient approximation takes over ten minutes on an Intel i7 machine.

4.2 Solving the LP

MATLAB’s linear program solver *linprog*, included in the optimization toolkit, was perfectly well suited for this problem. As an example, for the 15029-bus case, it solves the linear program (3) for each iteration in about 1.2 seconds. Out of curiosity, I installed the commercial solver Gurobi [5] with an academic license. Gurobi is state-of-the-art software which also solves mixed-integer and quadratic optimization problems. It solved the same problem as before in ~.035 seconds, and so I continued to use it moving forward.

4.3 The line search: maximize $f(x + \alpha p)$

I first considered a simple backtracking line search to satisfy the Armijo condition. To improve upon this, I considered adding some sort of Wolfe condition, to get closer to a minimizer of $f(x_k + \alpha p_k)$. Since computing ∇f is so expensive, this was a bad idea. Next I simply evaluated $f(x_k + \alpha p_k)$ on a grid of 25 evenly spaced points with $\alpha \in [0, 1]$, and picked out the maximum. I eventually realized there must be a better way: I settled on MATLAB’s *fminbnd* for minimizing single-variable functions on a fixed interval. According to the doc file, it uses a golden section search and parabolic interpolation. This consistently finds the minimum to a tolerance of 1e-4 with fewer than 20 function evaluations, better than my grid method.

4.4 Advanced software: IPOPT

To contrast with the simple Frank-Wolfe algorithm, I also solved the problem with the modern, open-source software library IPOPT [6]. It uses a primal-dual interior-point method as described in the final lecture, with an adaptive strategy for updating the barrier parameter and a filter line search method. From what I understand, interior-point methods are typically preferred over SQP for large-scale problems, as the latter can suffer from combinatorial complexity associated with determining the active set. So this seemed like a good choice for my problem.

IPOPT requires ∇f, so I had it use my finite difference approximation. It uses Hessian information if available. If not, as in my case, IPOPT does a limited-memory BFGS update to approximate the Hessian.

MATLAB's optimization toolbox also includes solvers for constrained non-linear problems, including SQP and interior-point algorithms. I looked into these briefly, getting poor results.

5 Numerical Results

MATPOWER includes a large number of test cases of various sizes, from a toy example of 9 buses up to actual Polish data with 4736 buses. I also have access to some old data for the U.S./Canada Eastern interconnection, with 15029 buses.

An example which is typical of all the small ($\leq$ 300 buses) cases is the 1961 W IEEE 57 Bus Test Case. With constraints (2) set with the maximum line multiplier $\alpha = 3$ and the total multiplication to $\beta = 6$, both algorithms place the entire budget on three lines. Here are the results for the Frank-Wolfe algorithm:

iteration	objective	step size	43	46	50	15	25	42
0	0	-	1	1	1	1	1	1
1	.14687	1	1	3	3	1	3	1
2	.16041	.249	1.50	2.50	2.50	1.50	2.50	1.50
3	.20666	1	3	3	3	1	1	1
4	.20666	1	3	3	3	1	1	1

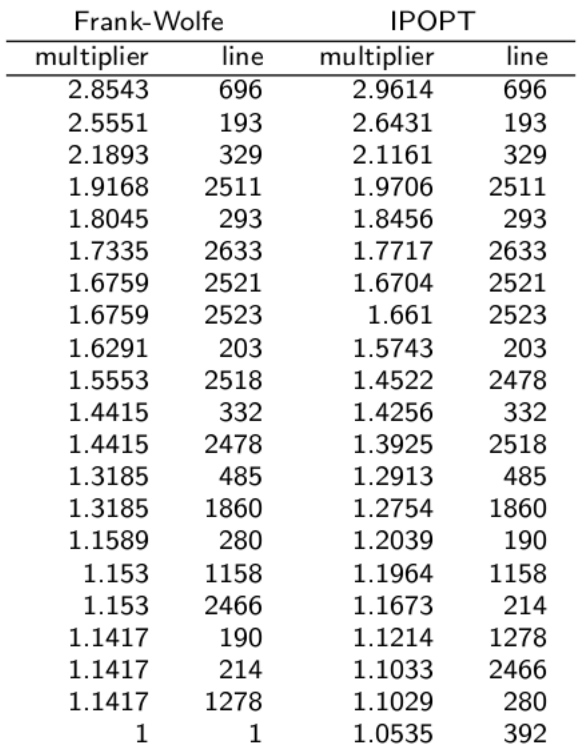
Figure 2: Results for Frank-Wolfe algorithm on 57 bus case with $\alpha = 3$, $\beta = 6$.

I've at each iteration subtracted off the objective of the initial, unperturbed system, to report only the change in voltage disruption caused by the attack. The numbers at the top are the lines being attacked. The algorithm terminated in .84 seconds. The fourth row is due to a tiny improvement in the last iteration that is beyond the precision shown in the table.

IPOPT produced the exact same attack in 147 iterations and 11.8 seconds. These trends were true for all the small cases, namely:

- Frank-Wolfe and IPOPT produce the same attack, focused entirely on a small set of lines
- IPOPT takes 5-10 times as long to terminate

For the larger cases, the attack is sometimes less concentrated. As an example, the attack against the 2746 bus Polish winter 2003-2004 case, with $\alpha = 5$ and $\beta = 12$ is distributed over about 20 lines. However, the objective for this case is much smaller, so small in fact that we can likely conclude that the case is safe against this sort of attack. Frank-Wolfe achieved an objective of .012496 in 134 seconds, while IPOPT got .0125193 in 379 seconds. In this case, IPOPT was only about three times slower, though again it picks out essentially the exact same lines. See figure 3.



Frank-Wolfe		IPOPT	
multiplier	line	multiplier	line
2.8543	696	2.9614	696
2.5551	193	2.6431	193
2.1893	329	2.1161	329
1.9168	2511	1.9706	2511
1.8045	293	1.8456	293
1.7335	2633	1.7717	2633
1.6759	2521	1.6704	2521
1.6759	2523	1.661	2523
1.6291	203	1.5743	203
1.5553	2518	1.4522	2478
1.4415	332	1.4256	332
1.4415	2478	1.3925	2518
1.3185	485	1.2913	485
1.3185	1860	1.2754	1860
1.1589	280	1.2039	190
1.153	1158	1.1964	1158
1.153	2466	1.1673	214
1.1417	190	1.1214	1278
1.1417	214	1.1033	2466
1.1417	1278	1.1029	280
1	1	1.0535	392

Figure 3: Polish winter 2003-2004 case, 2746 buses, $\alpha = 5$ and $\beta = 12$. Frank-Wolfe and IPOPT pick essentially the same attack.

LINE	MULTIPLIER
22039	2.2134
22037	2.1495
22046	1.8873
22047	1.8564
21667	1.7675
22348	1.6733
23091	1.6014
23090	1.597
22922	1.4845
21607	1.4153
23452	1.4091
23451	1.4091
21836	1.4064
21833	1.4034
21606	1.3222
23371	1.2661
23368	1.2652
22350	1.2622
21919	1.2483
21920	1.2436
21893	1.18
21888	1.1728
22349	1.1481
23396	1.1402
23397	1.1395
22719	1.1317
21887	1.0911
21892	1.0909
22757	1.0165
22756	1.0084

Figure 4: Eastern interconnect estimated summer 2003, 15029 buses, $\alpha = 5$ and $\beta = 12$. IPOPT achieves an objective of .217 in ten hours and five minutes.

As a final example, consider the 15029-bus Eastern interconnection case. Unfortunately, I failed to run this with IPOPT and Frank-Wolfe with the same parameters in time for the deadline. Frank-Wolfe takes around an hour and a half to terminate, while IPOPT takes around ten hours. In figure 4, I present the attack computed by IPOPT with $\alpha = 5$ and $\beta = 12$. As I email this in, Frank-Wolfe has completed three iterations in 35 minutes, objective is .188, and it has distributed the attack over the same lines as IPOPT as seen in the figure.

In all cases, IPOPT and Frank-Wolfe distribute the attack over the same set of lines. IPOPT sometimes achieves a slightly higher objective, but at a significant cost in time. As the true purpose of my problem is to determine which lines are the most vulnerable, the slighly higher objective is not worth the cost.

6 Conclusion

I've reformulated the $N - k$ problem into a continuous form amenable to continous optimization methods. I used a simple first-order method to solve the optimization problem, which produces a combinatorial solution as hoped; the solution to the continuous problem is to focus the attack onto a small number of lines. This captures the spirit of a small k in the original $N - k$ problem. I compare the simple optimization method to a more sophisticated one, and observe that they produce the same results.

References

[1] D. Bienstock and A. Verma. The $N - k$ Problem in Power Grids: New Models, Formulations, and Numerical Experiments. *SIAM J. Optim.,* 20, 2352-2380, 2010.

[2] A. Bergen and V. Vittal. *Power Systems Analysis.* Prentice Hall, Upper Saddle River, New Jersey, 2000.

[3] http://www.pserc.cornell.edu/matpower/

[4] SAS/OR User's Guide: Mathematical Programming. Retrieved May 11, 2011, from http://www.otago.ac.nz/sas/ormp/chap5/sect28.htm

[5] http://www.gurobi.com/

[6] https://projects.coin-or.org/Ipopt