Pacific Northwest NATIONAL LABORATORY

Fast evaluation of security constraints in a security constrained unit commitment algorithm

Jesse Holzer[1] (presenter),

Yonghong Chen[2], Feng Pan[1],

Ed Rothberg[3], Arun Veeramany[1]

[1]PNNL, [2]MISO, [3]Gurobi

FERC Technical Conference

June 26, 2019

U.S. DEPARTMENT OF ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

Security constraint evaluation Background in HIPPO

- HIPPO project – High Performance Computing for Power Grid Optimization
  - 3 year funding from ARPA-E
  - PNNL, MISO, GE, U. Tenn., U. Fla.
  - Goal 10x speedup over current GE method for SCUC in MISO DA market
- Current method:
  - SCUC MIP model with a small set of SCs – watchlist
  - Fix commitment variables
  - Evaluate remaining SCs on the dispatch solution
  - If any violations, add constraints and reoptimize dispatch – LP only
- Evaluation of SCs is key

Security constraint evaluation
Motivation for new method

- Current SC evaluation method is slow
  - ~10 minutes for 50K buses, 1K ctgs, 10K monitored branches, 36 time periods
  - Probably not optimized to our context – DC model, PGen bounds ignored in reaction to imbalance due to outages
  - Difficult to use to benchmark HIPPO SCUC MIP algorithms
  - Impractical to use inside SCUC algorithm – SCUC should be ~20 minutes
  - Difficult to make changes for use in HIPPO – coded in C, vs HIPPO in Python
  - Potential gain from using SC evaluation inside SCUC algorithm – optimize commitment decisions against all SCs
  - We can do better!
- New SC evaluation method in HIPPO
  - Coded in Python with open source linear algebra libraries
  - Use Sherman-Morrison-Woodbury formula to treat contingencies, instead of partial refactorization used by current method
  - Much faster – 5-20 seconds vs 10 minutes
  - Enables SC evaluation within SCUC algorithm

SCUC formulation
Highlighting security constraints

- Minimize
  - $F(X,Y)$
- Subject to
  - (X,Y) in G
  - $B\,Y \leq H$
- X – generator commitment schedules
- Y – power injections
- $B\,Y \leq H$ – security constraints
  - Flow limit on every monitored line in the base case and every security contingency in each time period
- Initial MIP model may have a very small subset of the security constraints – a watchlist – but all need to be checked and satisfied by reported solution
  - 1K ctgs, 10K monitored branches, 36 time periods, 360M total linear inequalities
  - 2K injection nodes, 720B nonzeros
  - Watchlist ~200 constraints per time period

Security constraint formulation – base case

- Base case branch flows are:
  - $R = -((C M^T Z)(A^{-1} E)) Y$
- Where
  - Y – pnode injections
  - E – convert pnode injections to bus injections
  - A – bus admittance matrix
  - Z – zero out reference bus angle
  - M – bus-branch incidence matrix
  - C – monitored branch admittance
  - R – monitored branch flows
- We use a Cholesky factorization for A^{-1}

Security constraint formulation – contingencies

- Contingency k admittance matrix is a rank s_k update of base case
  - $A_k = A + M_k C_k M_k^T$
- Some SC solvers use a partial refactorization technique to undo some pivots of a Cholesky factorization of A, then do some new pivots, to obtain a factorization of A_k. Same technique to move to the next contingency.
- We use the Sherman-Morrison-Woodbury formula:
  - $A_k^{-1} = A^{-1} – W_k V_k^{-1} W_k^T$
- W_k has s_k columns, V_k is s_k-by-s_k
  - $W_k = A^{-1} M_k$
  - $V_k = C_k^{-1} + M_k^T W_k$
- Then
  - $R_k = - (((C M^T) Z) (A^{-1} E)) Y + (((C M^T) Z) W_k) ((V_k^{-1} (W_k^T E)) Y)$
- This method can also handle bus outages and restoration of power imbalance in a contingency by prescribed participation factors. Both of these are low rank linear operators.

Performance features

- Precompute as much as possible, i.e. before calling SC evaluation on any particular dispatch vector. Minimize SC evaluation time.
  - Cholesky factorizations
  - Low rank factors
- Optimize use of sparse and dense matrices. Dense multiplication can be faster with low rank matrices.
- Compute only the most violated contingency for each monitored branch.
- After evaluating base case term and contingency term, R_k need not be computed for most branches
- Optimize order of multiplication operations to work with small matrices
- Pre-allocate work vectors during startup for computation in place during solve, i.e. without reallocating memory.
- Compute base case sensitivity matrix in startup. ~10% of startup time.
- Treat all contingencies of the same rank in a single matrix computation, rather than a loop over contingencies. Still need to loop over ranks. There are not many different ranks, ~30.

Computational results
Example with current SC evaluation method

- Case 105
  - SCUC to 0.1%, 791s
  - SC evaluation 812s
  - SCUC to 0.1%, 797s
- SC evaluation is slow

Pacific Northwest NATIONAL LABORATORY

Computational results
New method startup time

- Compute factorizations (cholesky, low rank) in startup phase
- Solve phase: given injections *Y*, evaluate flows *R*, determine SC violations.
- Need to build an SC evaluator for each of 36 time periods. These can be in parallel, but we do not want to use too many resources
- Any calls to solve must wait until startup is complete.
- Future work: build only 1 SC evaluator, use low rank perturbation idea to handle differences in base case admittance matrix between each time period and a static matrix
- Startup time is manageable, and note very fast solve time.

MIP_MSS_10901201901102309_0X_run1_um1_CONCURRENT.log				
SFT configuration	3node*12processor	1node *12 processor	1node*36processor	6node*6processor
Pre-processing #Matrix/Node	12	12	36	6
#nodes	3	1	1	6
#Matrix	36	12	36	36
SFT run time | end time | #violation	40.22 | **195.70** | 252	39.85 | **197.47** | 252	418.73 | **572.77** | 252	5.82| 161.28 | 252
	4.46 | 203.47 | 7	8.82 | 209.61 | 7	7.88 | 583.93 | 7	3.88 | 168.44 | 7
	4.34 | 237.23 | 1	8.73 | 248.44 | 1	7.84 | 620.60 | 1	3.84 | 201.45 | 1
	4.35 | 260.45 | 0	8.70 | 276.21 | 0	7.73 | 646.93 | 0	3.83 | 224.04 | 0
	4.40 | 276.81 | 0	8.23 | 296.49 | 0	7.42 | 666.12 | 0	3.80 | 239.68 | 0
	4.36 | 294.97 | 1	8.60 | 319.35 | 1	7.85 | 687.60 | 1	3.75 | 257.12 | 1
	4.35 | 312.84 | 1	8.70 | 341.97 | 1	7.65 | 708.68 | 1	3.77 | 274.27 | 1
	4.36 | 328.24 | 0	8.29 | 361.73 | 0	7.74 | 727.39 | 0	3.85 | 289.09 | 0
Total Time	419	452	816	378
	H 0 0 1.640910e+07 1.6355e+07 0.33% - 115s	H 0 0 1.640910e+07 1.6355e+07 0.33% - 116s	H 0 0 1.640910e+07 1.6355e+07 0.33% - 492s	H 0 0 1.640910e+07 1.6355e+07 0.33% - 80s

SCUC solution methods starting with a small initial set of SCs in the MIP model

- Method 1 (ED-SC iteration)
  - Solve SCUC to 0.1% mipgap for UC solution X and dispatch solution Y. Fix X.
  - Repeat:
    - ✓ Evaluate SCs on Y. If no new SC violations, stop
    - ✓ Add violated SCs and reoptimize for dispatch Y
- Method 2 (UC-SC-SQ sequential iteration)
  - Solve SCUC to 0.1% for (X,Y).
  - Repeat:
    - ✓ Evaluate SCs. If no new SC violations, stop
    - ✓ Add violated SCs and reoptimize for (X,Y). New MIP solve with MIP start from previous X
- Method 3 (UC-SC-CB callback)
  - Solve SCUC to 0.1% for (X,Y) with a callback
  - In callback, given a mip solution (X,Y) evaluate SCs, adding violated SCs if any
- Method 4 (UC-SC-H sequential-callback hybrid)
  - Solve SCUC to 0.1% for (X,Y).
  - Evaluate SCs. If no new SC violations, stop
  - Add violated SCs and reoptimize for (X,Y), using SC callback.

UC-SC-SQ, UC-SC-CB, UC-SC-H

- Case 105

SEQ			CallBack			
SCUC	SFT	Violation	Time	SFT	Violation	gap
950	5.3	156	70	6.5	158	-
964	3.2	1	84	2.5	3	-
903	3.4	0	98	2.3	0	2.45%
			...			
Total	2828.9		253	2.4s	0	0.60%
			...			
objval: 22843244.1577			1478			0.09%
objbound: 22820405.274						
			objval: 22840649.6281			
			objbound: 22818448.3573			

SEQ1+CallBack			
SCUC	gap	SFT	violation
120	0.43%	9.3s	157
Final callback			
1092	0.09%		
objval: 22839444.9628			
objbound: 22818779.0827			
Total	1212.005		

UC-SC-SQ, UC-SC-CB, UC-SC-H

- Case 605

SEQ1+CallBack (new)				CallBack			
SCUC	gap	SFT	violation	Time	SFT	Violation	gap
75	0.28%	14s	239	99	14.5	239	-
				110	3.3	3	
Final callback							
1103	0.09%						
				1483			0.09%
objval: 22839444.9628							
objbound: 22818779.0827				objbound: 22857665.9462			
Total	1178.0037			runtime: 1483.65039706			

SEQ		
SCUC	SFT	Violation
651	10	160
559	4.5	4
606	4.2	2
536	4.1	0
Total	2374.8	

- Case 116

SEQ1+CallBack (new)				CallBack			
SCUC	gap	SFT	violation	Time	SFT	Violation	gap
81	0.17%	12.7	346	106	15	345	
				118	3	4	
Final callback				130	2.7	0	0.32%
165	0.07%			178		0	0.07%
objval: 22839444.9628				objval: 39608981.6948			
objbound: 22818779.0827				objbound: 39580669.8076			
Total	246.0024						

SEQ		
SCUC	SFT	Violation
125	14.8	264
94	3.8	3
92	3.6	0
Total	333.2	
objval: 39612831.3248		
objbound: 39573964.1077		

Further SCUC/SC algorithmic possibilities

- Without bus outage and rebalance feature, tend to see multiple iterations with SC violations and new SCs added, though majority are in iteration 1
- With bus outage and rebalance feature, SC violations and new SCs added occur exclusively at iteration 1, and fixing UC variables and reoptimizing dispatch never incurs additional cost.
- We can probably be successful with the UC-SC-ED heuristic
- Need to explore SC evaluation and adding violated constraints based on LP relaxation solution.
- Full exploration of UC/ED/SC configuration made possible by efficient SC evaluation algorithm.
- HIPPO has multiple LB and UB algorithms. Need to communicate violated SCs found in one algorithm with the others to avoid redundant SC evaluations

Pacific Northwest
NATIONAL LABORATORY

Thank you