The Art of Simplification: Tackling Large-Scale Unit Commitment and AC Optimal Power Flow Problems

Wenyuan Tang

(Occam's Razor)

Grid Optimization (GO) Competition Challenge 3

November 15, 2023

Experience

- Before the competition, ACOPF had been the "most familiar stranger" to me
- My research had been focused on analysis and design of market mechanisms in electricity markets, where economic dispatch and pricing is based on DCOPF
- I knew there are tools for ACOPF (e.g., MATPOWER), but I had never written my own ACOPF program (even for a toy case)
- I have followed the GO Competition since Challenge 1, and I do believe those are important open problems that are worth our efforts, but participating in the competition seems too risky for junior faculty under the tenure clock
- Recently, I participated in some American-Made Challenges (Solar Forecasting Prize, Net Load Forecasting Prize, Hydropower Operations Optimization Prize), through which I built self-confidence and felt ready to compete with other prestigious teams in the GO Competition, the most difficult competition so far
- I got 6 weeks of (relatively) free time before the deadline of Event 4, and decided it was time to take the challenge!
- I thought it was a great deal: even if I couldn't get an award, at least I would learn how to formulate the UC and ACOPF problem in practice and how to write a program to solve it (at least for a toy case)

Experience

- Still, 6 weeks is too short: I kept myself extremely focused
- The problem formulation is already overwhelming: more comprehensive than any model I have ever seen in the literature
- I even thought some assumptions or equations in the problem formulation were not correct, which turned out to be my misunderstanding
- At the beginning, I tried to use Python and Gurobi (my favorite solver)
- While Gurobi currently supports general nonconvex constraints, it is a global optimizer and may not be suitable for such a hard problem
- Then I found Ipopt, a popular nonlinear solver for local optimization
- Unfortunately, it was not easy for me to interface Ipopt from Python (it shouldn't be the case, but I didn't have time to figure it out)
- Then I found the benchmark algorithms from previous competitions were written in Julia/JuMP, and I decided to learn Julia/JuMP in one day and convert the previous code from Python to Julia in another day (thanks to ChatGPT)
- It was a wise choice: it is very convenient to specify the constraints with a modeling language like JuMP, and calling Gurobi and Ipopt from JuMP is a breeze
- The rest is algorithm development (with lots of trial and error, back and forth)

Methodology

- I believe in the principle of Occam's razor: *Entia non sunt multiplicanda praeter necessitatem* (entities must not be multiplied beyond necessity)
- A similar saying attributed to Albert Einstein: *Everything should be made as simple as possible, but not any simpler*
- My realistic goal was to develop a good (but not necessarily the "best") algorithm that can stably generate feasible and reasonably good solutions within the time limits
- The original problem is computationally intractable: large-scale, mixed-integer, non-convex; multi-period, zonal reserve, etc.
- To generate a feasible solution within reasonable amount of time, a lot of simplifications need to be made
- At the upper level, my proposed algorithm has a simple structure: UC for the entire time horizon, and then ACOPF for each time period
- At the lower level, the purpose is to alter the feasible region and the objective function through relaxation, restriction, and approximation, to simplify the structure of each subproblem
- The key is to find adequate alterations and (if needed) projections onto the original feasible region: the art of simplification

Algorithm

Algorithm Pseudocode of Occam's Razor's solution

```
(1–135) read elementary data and compute derived data
(136–185) configuration of algorithm parameters
if difficult case then
    (186–238) very simple UC: min (# of startups + # shutdowns) (MILP by Gurobi)
else
    (239–544) simple UC: assuming a one-bus system (MILP by Gurobi)
end if
(545–567) set variable bounds for ramping constraints given UC
for $t = 1$ to $T$ do
    (568–975) ACOPF with or without reserves (NLP by Ipopt)
    (976–1076) fix voltages (if needed, adjustment for feasibility) and flows
    (1077–1336) ACOPF with reserves given fixed voltages and flows (LP by Gurobi)
end for
(1337–1406) write solution
```

- For difficult cases, conduct a very simple UC which tries to follow the initial statuses as much as possible (similar to the prior operating point algorithm)

Algorithm

Algorithm Pseudocode of Occam's Razor's solution

```
(1–135) read elementary data and compute derived data
(136–185) configuration of algorithm parameters
if difficult case then
    (186–238) very simple UC: min (# of startups + # shutdowns) (MILP by Gurobi)
else
    (239–544) simple UC: assuming a one-bus system (MILP by Gurobi)
end if
(545–567) set variable bounds for ramping constraints given UC
for $t = 1$ to $T$ do
    (568–975) ACOPF with or without reserves (NLP by Ipopt)
    (976–1076) fix voltages (if needed, adjustment for feasibility) and flows
    (1077–1336) ACOPF with reserves given fixed voltages and flows (LP by Gurobi)
end for
(1337–1406) write solution
```

- For most cases, conduct a simple UC which ignores flows (and therefore also voltages and line losses)

Algorithm

Algorithm Pseudocode of Occam's Razor's solution

```
(1–135) read elementary data and compute derived data
(136–185) configuration of algorithm parameters
if difficult case then
    (186–238) very simple UC: min (# of startups + # shutdowns) (MILP by Gurobi)
else
    (239–544) simple UC: assuming a one-bus system (MILP by Gurobi)
end if
(545–567) set variable bounds for ramping constraints given UC
for $t = 1$ to $T$ do
    (568–975) ACOPF with or without reserves (NLP by Ipopt)
    (976–1076) fix voltages (if needed, adjustment for feasibility) and flows
    (1077–1336) ACOPF with reserves given fixed voltages and flows (LP by Gurobi)
end for
(1337–1406) write solution
```

- Since we will conduct ACOPF for each time interval given fixed UC, we need to set additional bounds for variables (such as power) to respect ramping constraints

Algorithm

Algorithm Pseudocode of Occam's Razor's solution

```
(1–135) read elementary data and compute derived data
(136–185) configuration of algorithm parameters
if difficult case then
    (186–238) very simple UC: min (# of startups + # shutdowns) (MILP by Gurobi)
else
    (239–544) simple UC: assuming a one-bus system (MILP by Gurobi)
end if
(545–567) set variable bounds for ramping constraints given UC
for $t = 1$ to $T$ do
    (568–975) ACOPF with or without reserves (NLP by Ipopt)
    (976–1076) fix voltages (if needed, adjustment for feasibility) and flows
    (1077–1336) ACOPF with reserves given fixed voltages and flows (LP by Gurobi)
end for
(1337–1406) write solution
```

- This step is the computational bottleneck: depending on the case, we may conduct a full ACOPF (with reserves) or a partial ACOPF (without reserves)

Algorithm

Algorithm Pseudocode of Occam's Razor's solution

```
(1–135) read elementary data and compute derived data
(136–185) configuration of algorithm parameters
**if** difficult case **then**
    (186–238) very simple UC: min (# of startups + # shutdowns) (MILP by Gurobi)
**else**
    (239–544) simple UC: assuming a one-bus system (MILP by Gurobi)
**end if**
(545–567) set variable bounds for ramping constraints given UC
**for** $t = 1$ to $T$ **do**
    (568–975) ACOPF with or without reserves (NLP by Ipopt)
    (976–1076) fix voltages (if needed, adjustment for feasibility) and compute flows
    (1077–1336) ACOPF with reserves given fixed voltages and flows (LP by Gurobi)
**end for**
(1337–1406) write solution
```

- This step ensures the voltages and flows are feasible: we only adopt (or adjust) voltages from the previous ACOPF solution

Algorithm

Algorithm Pseudocode of Occam's Razor's solution

```
(1–135) read elementary data and compute derived data
(136–185) configuration of algorithm parameters
if difficult case then
    (186–238) very simple UC: min (# of startups + # shutdowns) (MILP by Gurobi)
else
    (239–544) simple UC: assuming a one-bus system (MILP by Gurobi)
end if
(545–567) set variable bounds for ramping constraints given UC
for $t = 1$ to $T$ do
    (568–975) ACOPF with or without reserves (NLP by Ipopt)
    (976–1076) fix voltages (if needed, adjustment for feasibility) and flows
    (1077–1336) ACOPF with reserves given fixed voltages and flows (LP by Gurobi)
end for
(1337–1406) write solution
```

- Finally, we conduct a full ACOPF which is simple since voltages and flows are given: it is also convenient to ensure numerical precision in this step

Some Other Simplifications

- (Objective) ignore downtime-dependent startup costs
- (Objective) ignore max/min energy over multiple intervals (except simple UC)
- (Objective) ignore post-contingency
- (Variable) set step for each shunt as initial status
- (Variable) set phase difference and winding ratio for each transformer as initial status
- (Variable) turn on all branches if switching is allowed
- (Constraint) relax max in endogenous reserve requirements

$$p^{\text{max}}_{nt} = \max_{j \in J^{\text{pr}}_n} p_{jt} \qquad \longrightarrow \qquad p^{\text{max}}_{nt} \geq p_{jt},\ \forall j \in J^{\text{pr}}_n$$

- (Constraint) restrict violation in max/min energy over multiple intervals (simple UC)

$$\begin{cases} \sum_{t \in T^{\text{en,max}}_w} d_t p_{jt} \leq e^{\text{max}}_w + e^{+}_w,\ \forall w \in W^{\text{en,max}}_j \\ \sum_{t \in T^{\text{en,min}}_w} d_t p_{jt} \geq e^{\text{min}}_w - e^{+}_w,\ \forall w \in W^{\text{en,min}}_j \end{cases}$$

$$\longrightarrow \qquad \begin{cases} \sum_{t \in T^{\text{en,max}}_w} d_t p_{jt} \leq e^{\text{max}}_w + e^{+}_j,\ \forall w \in W^{\text{en,max}}_j \\ \sum_{t \in T^{\text{en,min}}_w} d_t p_{jt} \geq e^{\text{min}}_w - e^{+}_j,\ \forall w \in W^{\text{en,min}}_j \end{cases}$$

Configuration

	8316	6717	6708	6049	4224	2000	1576	other
`max_wall_time`	21/120/280 sec			24/120/280 sec				
`lazy_uc`	T				F/T/T	T	F	
`co_opt`	F	F/T/T	F					T
`fast_solver`	T/F/F				F			
`stable_iter`	F/T/T	T	F/T/T		F			T

- `max_wall_time`: time limit (Divisions 1/2/3) per time interval for (568–975) ACOPF
- `lazy_uc`: use very simple UC (T) or simple UC (F)
- `co_opt`: reserves are consider (T) or not (F) in (568–975) ACOPF
- `fast_solver`: the linear solver in Ipopt is MA57 (T) or MA27 (F)
- `stable_iter`: the initial value for the barrier parameter in Ipopt is 0.03 (T) or 0.1 (F)
- Difficult cases are more sensitive to the algorithm parameters
- The above configuration is determined based on the previous datasets, which may not be suitable for Event 4 datasets

Results

- The proposed algorithm is easy to understand
- The proposed algorithm is easy to implement: 1406 lines of code
- The proposed algorithm is configurable for various datasets
- The proposed algorithm is guaranteed to generate feasible solutions within the time limits: true for all 669 cases in Event 4
- The generated solutions are reasonably good

Event 4 Division	1	2	3
Rank	4th	7th	5th
Event 4 Division	4	5	6
Rank (best score count)	6th (3)	8th (1)	8th (0)

- The proposed algorithm is not good at finding the "best" solutions: achieving the best scores for only 4 out of 669 cases
- The same program would have achieved better performance on Event 3 datasets (especially for Divisions 4/5/6): Event 4 datasets may be very different from Event 3 datasets; other teams may have improved substantially after Event 3

Questions Regarding the Algorithm

1. What process did your team use in deciding the algorithmic approach?
   See the aforementioned; try to make it as simple as possible, but not any simpler
2. Did your team consider/use a hybrid approach by running different types of algorithms in parallel?
   No, but I may consider it if I have time
3. Did you/your team consider adjusting the parameters/heuristics of your algorithm based on network characteristics? If yes, explain how?
   Yes, see the aforementioned
4. Did you/your team try to use any machine learning approach to learn the Sandbox datasets?
   No, because I may need many more cases to train such a machine learning model
5. Did you/your team consider changing the algorithmic approach/modeling approach when new datasets are published? If yes, why?
   No, because I wanted to keep a simple structure of the algorithm
6. Did the teams consider a "simultaneous multi-period" OPF approach (as opposed to considering each time period individually)? If so, how did it scale and what, if any, were the benefits to solution quality?
   No, but I may consider it if I have time

Questions Regarding the Algorithm

7. How (if at all) did your team incorporate reserve constraints into the OPF subproblem(s)?
   See the aforementioned
8. UC determines the binary variables and some continuous variables. We understand fixing the binary variables makes the remaining ACOPF a continuous non-convex programming. How do you treat the decision of the continuous variables determined by UC?
   Ignore the decision of the continuous variables
9. How do you update UC decisions if you find the first UC is not optimal or feasible?
   The UC from the proposed algorithm is always feasible because most constraints are soft constraints; given fixed UC, the ramping constraints (which are hard constraints) have been taken care of in the proposed algorithm

Questions Regarding the Data

1. What were the 10 most difficult scenarios to solve? And why?
   Some scenarios in 2000D2, 2000D3, 4224D2, 6717D1, and all scenarios in 23643, which I got negative objective values
2. What were the 10 least difficult scenarios to solve? And why?
   The others are relatively easy; smaller cases are generally easier to solve
3. Did you find difficulties with industrial networks (not released networks)? Explain why?
   No before Event 4, but yes in Event 4; I didn't find the reason
4. Did you notice considerable differences in difficulty amongst the networks?
   Yes, and sometimes smaller cases (such as 2000) can also be difficult
5. Did you find any idiosyncrasies of specific grids?
   UC for some cases is difficult; reserve requirements for some cases are stringent
6. Did you notice unusual behaviors/data in any grids?
   For some cases, the top teams' scores have large differences
7. Which constraints were more challenging to satisfy?
   Reserve requirements; bus power balance (when ACOPF does not converge)

Questions Regarding the Data

8. How would you compare the computational complexity of larger grids with the small grids?
   Network size is only a rough indicator (other data also matters); computational complexity also depends on the algorithm
9. Regarding the data input format, is it easy to parse?
   Yes
10. What are the main differences in the optimization behavior of D1, D2 and D3?
    Either D1 (which has a small time limit) or D2 (which has the largest number of time intervals) can be the most difficult; D3 is generally easier than D2; it also depends on the algorithm

Thank you, the GO Competition Administration Team!