Artelys_Columbia GO3 Competition Overview

Daniel Bienstock, Columbia University

 Richard Waltz, Artelys

Some elements of our software approach

- Underlying solver: **Knitro for all optimization solves.**

- **Python. Why.** We envisioned switching to a C-based approach, eventually (or maybe Julia).

- **AMPL. Why?** Interior-point methods require that the gradient and the Hessian of each constraint be provided at each iteration via a callback. AMPL seems quite capable in this regard.

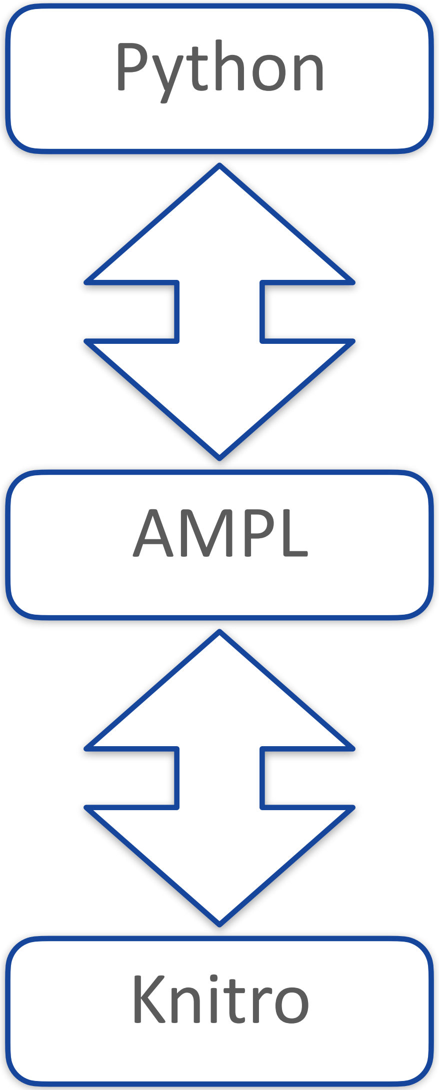
  However, there are limitations/bottlenecks in AMPL, especially in high-dimensional cases.

  - **Overhead loading data** from Python into AMPL
  - **AMPL overhead processing model** (performing variable substitutions, presolving, writing .nl file)
  - About 1/3 of optimization time spent in callbacks to **AMPL automatic differentiation**

- **Reader.** We used the basic JSON setup in Python to read the data in. No performance issues in the large cases.
  However, we had some issues digesting the documentation.
  Moreover, some of the provided JSON files were unkindly structured: one run-on line of text, with no carriage returns.
  This made it quite difficult to understand/debug the data.

- **Evaluator.** We modified the provided evaluator and incorporated into our code.
  Note: Good for debugging, but cumbersome to use in actual solution process.

PYTHON+AMPL EXAMPLE



- easy modeling
- presolver
- derivatives

Basic structure of our method

General solution process

1. Possibly fix some subset of (integer) decision variables to reduce problem size

2. Relax any remaining integer variables and solve NLP to low precision

3. Round and fix any remaining integer variables

4. Re-solve NLP to high precision (optimize over all continuous vars)
   - Warm-start this solve

5. If still not feasible (rare) try to repair solution to be feasible

Basic structure of our method

- Optimize over all time periods in one shot (some simplifications for largest networks)
- Run **4-8 solution processes in parallel** with different fixing strategies
- Fixing strategies used **depend on network size and time limit**/division
- Some customization/modification for smallest and largest cases

Strategies for fixing integer variables

Line switching variables

- Optimize (via relax+round procedure) — check connectedness after
- Fix to prior values
- Fix all on

Strategies for fixing integer variables

Line switching variables

- Optimize (via relax+round procedure) — check connectedness after
- Fix to prior values
- Fix all on

Producer/consumer (PR/CS) binary variables

- Optimize (via relax+round procedure)
- Fix to prior values

Strategies for fixing integer variables

Line switching variables

- Optimize (via relax+round procedure) — check connectedness after
- Fix to prior values
- Fix all on

Producer/consumer (PR/CS) device binary variables

- Optimize (via relax+round procedure)
- Fix to prior values

Switched shunts variables

- Usually optimize (via relax+round procedure)
- Fix to prior values in largest cases

Run **8 solves in parallel** with difference combinations of above strategies.

Modified approach for smallest cases

- Try to solve full mixed-integer NLP model (over all time periods) with Knitro!
- Run some feasibility heuristics in Knitro MINLP solver and stop at first feasible solution (good solution usually found at root node heuristics)
- Gave solutions with more changes across time periods
- Helped in some difficult 73-bus cases where standard fixing strategy (for producer/ consumer device variables) did not give a great solution
- Also run more standard (relax+round) strategy in parallel as backup options
- Only used for 73-bus cases. With more time could have extended to 617 bus cases also.

Modified approach for larger cases

General solution process

1. Possibly fix some subset of (integer) decision variables to reduce problem size

2. Relax any remaining integer variables; *use linear approximations for balance equations and line limit constraints and solve LP*

3. Round and fix any remaining integer variables

4. Re-solve NLP to high precision (optimize over all continuous vars)
    - Warm-start this solve

5. If still not feasible (rare) try to repair solution to be feasible

Modified approach for largest division 1 cases

- Try fixing **all** integer variables so we just solve one continuous NLP
    - Fix lines on
    - Fix binary PR/CS devices to prior
    - Fix integer switched shunts to prior values
- Treat some continuous variable/parameters as constant across time periods. $(e.g. p_{jt}^{on}, p_{jt}^{max}, p_{jt}^{min})$
- Also fix (continuous) reserve variables to 0

Device reserve variables are nonnegative.

$$p_{jt}^{\mathrm{rgu}} \geq 0 \ \forall t \in T, j \in J^{\mathrm{pr,cs}}$$
$$p_{jt}^{\mathrm{rgd}} \geq 0 \ \forall t \in T, j \in J^{\mathrm{pr,cs}}$$
$$p_{jt}^{\mathrm{scr}} \geq 0 \ \forall t \in T, j \in J^{\mathrm{pr,cs}}$$
$$p_{jt}^{\mathrm{nsc}} \geq 0 \ \forall t \in T, j \in J^{\mathrm{pr,cs}}$$
$$p_{jt}^{\mathrm{rru,on}} \geq 0 \ \forall t \in T, j \in J^{\mathrm{pr,cs}}$$
$$p_{jt}^{\mathrm{rru,off}} \geq 0 \ \forall t \in T, j \in J^{\mathrm{pr,cs}}$$
$$p_{jt}^{\mathrm{rrd,on}} \geq 0 \ \forall t \in T, j \in J^{\mathrm{pr,cs}}$$
$$p_{jt}^{\mathrm{rrd,off}} \geq 0 \ \forall t \in T, j \in J^{\mathrm{pr,cs}}$$
$$q_{jt}^{\mathrm{qru}} \geq 0 \ \forall t \in T, j \in J^{\mathrm{pr,cs}}$$
$$q_{jt}^{\mathrm{qrd}} \geq 0 \ \forall t \in T, j \in J^{\mathrm{pr,cs}}$$

Strategy for repairing infeasible solutions

- Almost never invoked.
- When optimizing line switching we had a routine to check network connectedness of the optimized solution. When disconnected, this routine provided minimal set of lines to turn on to establish connectedness.
- Otherwise, when infeasible, turn on all lines/generators and try again.
- Not sophisticated, but good enough.

Some other details

- **Not much attention** paid to contingencies in our submitted code – out of time. More later.

- Many different methods run in **parallel.** Pick the best one that terminates on time.

- On small instances, **MINLP** solver included in Knitro

- On most instances, **solve relaxation, then round and re-solve**

- Many **numerical tricks** involving Knitro
  1. Tolerances and convergence parameters
  2. Starting point: flat start, prior solution, other
  3. Sometimes, force strict feasibility for certain constraints (e.g., line limits)

- **Heuristic** for patching connectivity constraint
  1. Typically, paying no attention to connectivity constraint seemed to work. But not always.
  2. Compute a minimum number of ‘off’ branches to restore to ‘on’, and re-solve.

- Exploit **independence** with respect to time, whenever possible for largest cases

Time-(nearly) independent cases

- **A relatively frequent feature:** no ”forced-on” or “-off” constraints on devices, no max energy intervals, etc.

- **In fact, sometimes (often?), nothing was period dependent.**
  In that case, one can solve a 1-period problem, and obtain a feasible solution with all devices and branches on.
  Or, a model with 3 periods, where the first two are used to switch devices or branches. (not implemented)

- **More generally,** there may be a small (really small) number of time periods where a change may be necessary.
  In such a case, one could solve an equivalent problem over a **small** number of time-periods.
  Scalability, i.e., size of problem provided to Knitro, was often an issue for us.

- **Often,** for example, the interval-based constraints for producers/consumers were on aligned intervals.
  And only involving a few of the ‘devices’.

We are ~~sure we did not exploit~~ **not sure we exploited these features** to the maximum possible!

Why use multi-period approach?

- **We assume if we can solve over all time periods simultaneously, this would be better.** Generally we were able to do that (with some simplifications for the largest — primarily division 1 — cases).
  - **Knitro able to solve non-convex NLP with ~10 million variables in reasonable time**
- Some constraints linking multiple time periods made it a bit more complex
- Not clear that solving several smaller problems would be faster/better than 1 large problem
- Nonetheless, we wanted to implement and try as a comparison, but ran out of time.
- For largest cases, we took a middle approach where we still solved (generally) over all time periods in 1 solve, while treating some variables as constant across time periods to reduce problem size.

We did not have time for: adequate modeling of contingencies

- In GO3, each contingency consists of a branch loss
- Post contingency, only active power flows are modeled
- A DC power flow model is used – phase angles are controllable
- Bus injection mismatches are penalized
- Most important feature: line overloads are penalized.
- We can express everything through appropriate inequalities; what is the problem? AMPL.
- In the end we did nothing for contingencies other than ensuring network connectedness.
- Generally, we did not find in practice that the penalties from ignoring contingencies were that significant.

Summary of Final Event Results

Network	Positive scores	% Best All	% Best D1	% Best D2	% Best D3
73	104/104	98%	86%	95%	99.9%
617	102/102	94%	98%	96%	93%
1576	48/48	97%	88%	NA	98%
2000	39/39	99%	99.7%	99.7%	98%
4224	76/76	99.6%	93%	100%	100%
6049	78/78	96%	95%	96%	96%
6708*	43/43	99.6%	95%	99.8%	99.7%
6717	55/57	93%	89%	94%	94%
8316	115/116	92%	86%	90%	92%
23643	4/6	56%	68%	59%	54%
Total	664/669				

Comments on Test Networks/Scenarios

- Not much performance variation among different scenarios in a network (e.g. generally if we did well on one, we did well on all of them and vice versa).

- Not much performance variation among scenarios in different divisions of the same network (other than challenge of 15 minute division 1 time limit on largest networks).

- The **most difficult** scenarios were from 6717, 8316 and 23643 bus networks particularly for division 1 just because of the optimization problem size and time limit.

- The **least difficult** scenarios for us were from the 2000 bus network — but I'm not sure why.

- We did not have any difficulties with the industrial networks (6708 bus network).

Two difficult smaller cases

1. C3S3N000**73**D1, scenario 302
   - Our score was an order of magnitude worse than benchmark using our standard relax/ round (or other fixing) heuristics.
   - All producer/consumer devices turned off at t=0?
   - This bad instance was discovered 10 days before the Final Event
   - Led us to implement MINLP approach for smallest cases

2. C3E3N0**1576**D2, scenario 31 (Event 3)
   - Noticed a sizable difference between the Knitro objective and the evaluator score (not due to ignored contingencies in Knitro)
   - Max/min energy constraints over time intervals for PR/CS devices active
   - Revealed a bug in our code 3 weeks before the Final Event reading in data

More Comments on Test Networks/Scenarios

- We did not notice considerable differences in difficulty among networks.

- **We did not tune our solution approach to particular network characteristics** — other than network size (e.g. number of buses) and time limit/division.

- When we struggled it was usually because:
    1. There was a modeling/coding bug
    2. We ran out of time (e.g. for large division 1 instances)

- Most of our efforts over the last few months were dedicated to finding appropriate heuristics/simplifications to generate good solutions for the 6717, 8316 and 23643 bus networks within the 15 minute division 1 time limit.

- Unfortunately, this did not leave us time to explore other improvements on the ToDo list.

If we had more time…

- Handling of contingencies
- Analysis and Tuning to particular network structures
- Experiment with proper single-period approach
- Make use of leftover time to refine/improve solution (especially on smaller networks).
- Use linear MIP to determine settings for integer variables/UC
- Extend MINLP approach to some larger networks.
- Write our own (fast) solution evaluator to guarantee we would return best solution found.
- Code everything in C (would allow much more time for optimization in Division 1).

Some issues we had difficulty with

- **Difficult to search documentation**
  *Please: provide more **hyperlinks** in the document!*

- **Data and formulation document used different notation.** We understand this was unavoidable.

- **Computation of intervals!**

$$T_{jt}^{\text{dn,min}} = \{t' < t : a_t^{\text{start}} - a_{t'}^{\text{start}} + \epsilon^{\text{time}} < d_j^{\text{dn,min}}\} \ \forall t \in T, j \in J^{\text{pr,cs}}$$
$$T_{jt}^{\text{up,min}} = \{t' < t : a_t^{\text{start}} - a_{t'}^{\text{start}} + \epsilon^{\text{time}} < d_j^{\text{up,min}}\} \ \forall t \in T, j \in J^{\text{pr,cs}}$$

$$T_j^{\text{out}} = \{t \in T : u_{jt}^{\text{on,max}} = 0\} \ \forall j \in J^{\text{pr,cs}} \text{ with } d_j^{\text{up,0}} > 0$$
$$T_j^{\text{out}} = \{t \in T : u_{jt}^{\text{on,max}} = 0\} \cup \{t \in T : d_j^{\text{dn,0}} + a_t^{\text{start}} + \epsilon^{\text{time}} < d_j^{\text{dn,min}}\}$$
$$\forall j \in J^{\text{pr,cs}} \text{ with } d_j^{\text{dn,0}} > 0$$

$$T_{jft}^{\text{sus}} = \{t' \in T : t' < t, a_t^{\text{start}} - a_{t'}^{\text{start}} \leq d_{jf}^{\text{dn,max}} + \epsilon^{\text{time}}\} \ \forall j \in J^{\text{pr,cs}}, f \in F_j, t \in T$$
$$T_{jf}^{\text{sus}} = \{t \in T : d_j^{\text{dn,0}} + a_t^{\text{start}} > d_{jf}^{\text{dn,max}} + \epsilon^{\text{time}}\} \ \forall j \in J^{\text{pr,cs}}, f \in F_j$$

What we would like to see in GO4

- **Day-ahead markets with genuine uncertainty for real-time.**

- **In other words, multi-time period ACOPF SCUC with realistic uncertainty.**

- Better notation (and links).

- All constraint data pre-computed! Including intervals! Part of the input!

- **Thanks ARPA-E!**