Learning to Solve Large Scale Security-Constrained Unit Commitment

Alinson Xavier, Feng Qiu, Energy Systems Division Argonne National Laboratory

Shabir Ahmend, Xiaoyi Gu, Santanu S. Dey Industrial & Systems Engineering Georgia Institute of Technology
The work in this presentation is based on:

Learning to solve large-scale security-constrained unit commitment problems
Álinson Xavier, Feng Qiu, Shabir Ahmed
INFORMS Journal on Computing 33 (2), 739-756

Exploiting Instance and Variable Similarity to Improve Learning-Enhanced Branching
Xiaoyi Gu, Santanu S. Dey, Feng Qiu, Alinson Axavier,
(in preparation)

A relevant presentation
10-12AM on July 19 Tuesday, Governor’s square 9
Frontier of Power System Optimization and Simulation
Solving large-scale SCUC with MIPLearn+UnitCommitment.jl
Alinson Xavier
Problem in focus

• Security constrained unit commitment (SCUC)
  • seek most cost-effective generator commitment and production output levels
  • the most fundamental mixed-integer programming problem in power systems
    • electricity market clearing
      • $400 billion annually; 0.1 optimality gap in 20 mins; often ends up with large gaps
    • reliability analysis, production cost modeling, etc.
• Increasingly challenging
  • new energy components, e.g., combined cycle, energy storage, distributed energy resources
  • sub-hourly commitment, e.g., 15-min commitment
  • uncertainties
• Things that don’t work
  • Cutting planes
  • Strong formulations
  • Decomposition

Learning to Optimize

Minimize \( \sum_{g \in G} c_g(x_g, y_g) \) \hspace{1cm} (1)

Subject to \((x_g, y_g) \in G_g, \forall g \in G\) \hspace{1cm} (2)

\[ \sum_{g \in G} y_{gt} = \sum_{b \in B} d_{bt} \] \hspace{1cm} \( \forall t \in T \) \hspace{1cm} (3)

\[ -F_i^c \leq \sum_{b \in B} \delta_b \left( \sum_{g \in G_b} y_{gt} - d_{bt} \right) \leq F_i^c \] \hspace{1cm} \( \forall e \in L \cup \{0\}, l \in L, t \in T. \) \hspace{1cm} (4)

\[ x_{gt} \in \{0, 1\} \] \hspace{1cm} \( \forall g \in G, t \in T \) \hspace{1cm} (5)

\[ y_{gt} \geq 0 \] \hspace{1cm} \( \forall g \in G, t \in T \) \hspace{1cm} (6)
• Relevant work (prior to 2017)
  • Solve SCUC using Artificial Neural Networks:
    • Sasaki, Watanabe, Kubokawa, Yorino & Yokoyama (1992)
    • Wang & Shahidehpour (1993)
    • Walsh & O’mally (1997)
    • Liang & Kang (2000)
  • Use ML to enhance MILP solvers:
    • Alvarez, Wehenkel, Louveaux (2014)
    • Alvarez, Louveaux, Wehenkel (2017)
    • Khalil, Dilkina, Nemhauser, Ahmed, Shao (2017)

• Our perspective
  • General framework but NOT application-agnostic
  • Help solvers become progressively better over time
Learning to Optimize

• **Learning security constraints**
  - N-1 contingency criteria (transmission/security constraints) are fundamental reliability requirements enforced by NERC
  - Security constraints have large impact on performance:
    - Quadratic number, typically very dense
    - Very few are actually binding, but hard to tell in advance
  - **Contingency Oracle**: Predict which contingency constraints should be added to the relaxation and which should be omitted
  - **Training phase**:
    - Solve problem without any transmission constraints
    - Add small subset of most-violated constraints and resolve
    - Repeat until no further violations are found
  - **Test phase**: If constraint was necessary for at least 1% of training cases, add at start, then follow previous algorithm

\[-F^c_i \leq \sum_{b \in B} \delta^c_{lb} \left( \sum_{g \in G_b} y_{gt} - d_{bt} \right) \leq F^c_i.\]
Learning initial feasible solutions

- Primal bounds are still a bottleneck
  - Modern formulations/solvers usually yield very strong dual bounds
  - Most time is spent finding high-quality primal solutions
- Warm start: find, among large set of previous solutions, ones that are likely to work well as warm starts in MILP solvers
- Training phase (instance-based learning):
  - Solve each training instance and store its solution
- Test phase:
  - Find k training instances closest to the test instance
  - Use their k optimal solutions (or partial optimal solutions) as warm starts
Learning to Optimize

• **Learning affine subspaces**
  
  • Optimal SCUC solutions have a number of patterns:
    • Some units are operational throughout the day
    • Some units are only operational during peak demand
  
  • Affine subspace: Find subspaces (described by a set of hyperplanes) where the solution is very likely to reside
  
  • Training phase (instance-based learning):
    • Consider a fixed set of candidate hyperplanes
    • Build supervised models (e.g., SVM) to predict if hyperplane is satisfied. Discard models with low precision or recall
  
  • Test phase:
    • Predict which hyperplanes are likely to satisfied using previous model and add them to the relaxation
• Learning affine subspaces
  • Hyperplanes considered:
    • $x_{gt} = 0$
    • $x_{gt} = 1$
    • $x_{gt} = x_{g,t+1}$
  • Classifier: Support Vector Machines
• Training high-quality models:
  • Discard hyperplanes with very unbalanced labels
  • Measure precision and recall using k-fold cross validation
  • Discard models with low recall or precision
• **Computational results**

  • **SCUC**: Find cost-efficient power generation schedule, subject to:
    • Production during each hour must satisfy demand
    • Power flows must be within safe limits
    • Other physical, operational & economic constraints

  • **Widely used in planning and operations:**
    • Day-ahead electricity markets, reliability assessment

  • **Benchmark set**: 9 realistic, large-scale cases from MATPOWER

  • **Training instances**: 300 random variations

  • **Test instances**: 50 random variations

  • **Randomized parameters**:
    • Peak system-wide load
    • Production and start-up costs
    • Geographic load distribution
    • Temporal load profile

<table>
<thead>
<tr>
<th>Instance</th>
<th>Buses</th>
<th>Units</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>case1888rte</td>
<td>1,888</td>
<td>297</td>
<td>2,531</td>
</tr>
<tr>
<td>case1951rte</td>
<td>1,951</td>
<td>391</td>
<td>2,596</td>
</tr>
<tr>
<td>case2848rte</td>
<td>2,848</td>
<td>547</td>
<td>3,776</td>
</tr>
<tr>
<td>case3012wp</td>
<td>3,012</td>
<td>502</td>
<td>3,572</td>
</tr>
<tr>
<td>case3375wp</td>
<td>3,374</td>
<td>596</td>
<td>4,161</td>
</tr>
<tr>
<td>case6468rte</td>
<td>6,468</td>
<td>1,295</td>
<td>9,000</td>
</tr>
<tr>
<td>case6470rte</td>
<td>6,470</td>
<td>1,330</td>
<td>9,005</td>
</tr>
<tr>
<td>case6495rte</td>
<td>6,495</td>
<td>1,372</td>
<td>9,019</td>
</tr>
<tr>
<td>case6515rte</td>
<td>6,515</td>
<td>1,388</td>
<td>9,037</td>
</tr>
</tbody>
</table>
Learning to Optimize

• Computational results
  • The best record for solving large-scale SCUC

Learning to Solve Large-Scale Unit Commitment (Xavier, Qiu, Ahmed (2020))
Learning to Optimize

- **MIPLearn**
  - Flexible, extensible, and easy-to-use open-source framework for learning-enhanced integer programming
  - MIPLearn components
    - Initial feasible solutions
    - Lazy constraints and user cuts
    - Branching priorities
    - Optimal value
  - Modeling languages: JuMP, Pyomo, Gurobi Python API
  - Compatible MIP solvers:
    - Commercial: Gurobi, CPLEX, XPRESS
    - Non-commercial: SCIP, Cbc
  - Repository:
    - [https://github.com/ANL-CEEESA/MIPLearn](https://github.com/ANL-CEEESA/MIPLearn)
    - License: Open source (3-clause BSD)
Learning to Branch

Branch & Bound

• A schema that exhaustively search a solution space in a mixed-integer programming problem
• Combining with bounding techniques, it provides a solution with an optimality gap
• Market transparent and fairness
• A better branching strategy can help B&B convergence

A maximization problem

\[ x_1, x_2, x_3 \in \{0,1\} \]

Branching variable \( x_3 \)

Fractional
\[ z = 22 \]

\[ x_3 = 0 \]
Fractional
\[ z = 21.65 \]

\[ x_3 = 1 \]

Branching variable \( x_2 \)

Fractional
\[ z = 21.85 \]

\[ x_3 = 1, x_2 = 0 \]

Integer
\[ z = 18 \]

INTEGER

\[ x_3 = 1, x_2 = 1 \]

Fractional
\[ z = 21.8 \]

\[ x_3 = 1, x_2 = 1, x_1 = 0 \]

Integer
\[ z = 21 \]

INTEGER

\[ x_3 = 1, x_2 = 1, x_1 = 1 \]

Infeasible

INFEASIBLE

https://mat.tepper.cmu.edu/orclass/integer/node13.html
Learning to Branch

Branch

- Two decisions in branching: node selection and variable selection
- Node selection: best known strategy: always choose the nodes with best lower bounds
- Variable selection:
  - Most infeasible (fractional) branching (MIB): cheap but worst
    \[ S_{\text{MIB}}(i, l) = \min\{x_i, 1 - x_i\} \]
  - Strong branching: best (smallest b&b tree) but expensive
    \[ S_{\text{MIB}}(i, l) = \max\{\Delta_i^-, \epsilon\} \times \max\{\Delta_i^+, \epsilon\} \]
    \(\Delta_i^-\): objective value change when branch down
  - Reliability branching: a light version of strong branching (RB: \(\lambda: \eta\))
    (1) at most \(\lambda\) variables will be probed at a node
    (2) for a given variable, \(\eta\) number of probes are deemed sufficient
Benchmark ML-enhanced branching


• Key idea: use machine learning to mimic strong branching; a universal model for all MIPs

• Features
  • Static problem features
    • Computed from $c, A, b$
  • Dynamic problem features
    • The solution of the problem at the current B&B node
    • E.g., up and down fractionalities of a variable
  • Dynamic optimization features
    • Overall state of the optimization
    • E.g., statistic features of objective value changes regarding a certain variable
Motivation

Most industry applications are routinely solved optimization problems sharing high similarity, e.g., constraint matrix, right-hand sides.

Dedicated ML models for a routinely solved optimization problem should work better than a generic ML model.

Revised learning to branch approach

- Per variable: each variable has its own ML model
- Per group: a group of relevant variables share a ML model
  - Per generator: time index is ignored; e.g., is_on[g,-] v.s. is_on[g,t])
  - Per time: generator index is ignored; e.g., is_on[,] v.s. is_on[g,t])
  - Per type: time and generator indices are ignored; e.g., is_on v.s. is_on[g,t]
Experiment setup

- Home-made branch&bound MIP solver, using Gurobi for solving LPs
- ML model: Extremely Randomized Trees (ExtraTree)
- 5 realistic power systems, 24-hour SCUC

<table>
<thead>
<tr>
<th>Network</th>
<th>Hours</th>
<th>Generators</th>
<th>Buses</th>
<th>Lines</th>
<th>Variables</th>
<th>Rows</th>
<th>Binaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>case1888rte</td>
<td>24</td>
<td>296</td>
<td>1,888</td>
<td>2,531</td>
<td>235,591</td>
<td>196,783</td>
<td>41,232</td>
</tr>
<tr>
<td>case1951rte</td>
<td>24</td>
<td>390</td>
<td>1,951</td>
<td>2,596</td>
<td>266,088</td>
<td>244,220</td>
<td>54,144</td>
</tr>
<tr>
<td>case2848rte</td>
<td>24</td>
<td>544</td>
<td>2,848</td>
<td>3,776</td>
<td>377,760</td>
<td>340,228</td>
<td>71,904</td>
</tr>
<tr>
<td>case3012wp</td>
<td>24</td>
<td>496</td>
<td>3,012</td>
<td>3,572</td>
<td>357,146</td>
<td>305,076</td>
<td>59,712</td>
</tr>
<tr>
<td>case3375wp</td>
<td>24</td>
<td>590</td>
<td>3,374</td>
<td>4,161</td>
<td>413,161</td>
<td>357,065</td>
<td>71,856</td>
</tr>
</tbody>
</table>

- $R_b$: 100: inf (practical strong branching) used to collect data
Learning to Branch

Branching sore prediction experiments

Figure 1  Cross-Validation Evaluation (case1888rte, 24h, value after logarithm) with MSE
### Impact on solving SCUC with B&B

#### Table 4: Relative MIP Gap, node.limit=1000

<table>
<thead>
<tr>
<th>Instances</th>
<th>MIB</th>
<th>RB:100:8</th>
<th>ML:ET</th>
<th>ML:PNA</th>
<th>ML:PTI</th>
<th>ML:PGE</th>
<th>ML:PV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>Network</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>case1888rte</td>
<td>1.78</td>
<td>0.96</td>
<td>1.31</td>
<td>1.35</td>
<td>0.90</td>
<td>1.22</td>
</tr>
<tr>
<td>24</td>
<td>case1951rte</td>
<td>0.41</td>
<td>0.20</td>
<td>0.23</td>
<td>0.24</td>
<td>0.20</td>
<td>0.22</td>
</tr>
<tr>
<td>24</td>
<td>case2848rte</td>
<td>0.83</td>
<td>0.42</td>
<td>0.54</td>
<td>0.59</td>
<td>0.45</td>
<td>0.58</td>
</tr>
<tr>
<td>24</td>
<td>case3012wp</td>
<td>0.29</td>
<td>0.02</td>
<td>0.08</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>24</td>
<td>case3375wp</td>
<td>0.48</td>
<td>0.12</td>
<td>0.52</td>
<td>0.46</td>
<td>0.50</td>
<td>0.41</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td>0.76</td>
<td>0.35</td>
<td>0.54</td>
<td>0.53</td>
<td>0.41</td>
<td>0.49</td>
</tr>
</tbody>
</table>
## Impact on solving SCUC with B&B

* Similar performance can be observed in pre-solved instances

### Table 5: Relative MIP Gap, node limit=10000

<table>
<thead>
<tr>
<th>Instances</th>
<th>Relative MIP gap (%)</th>
<th>Instances</th>
<th>Relative MIP gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>Network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>case1888rte</td>
<td>1.75</td>
<td>1.13</td>
</tr>
<tr>
<td>24</td>
<td>case1951rte</td>
<td>0.37</td>
<td>0.11</td>
</tr>
<tr>
<td>24</td>
<td>case2848rte</td>
<td>0.77</td>
<td>0.43</td>
</tr>
<tr>
<td>24</td>
<td>case3012wp</td>
<td>0.27</td>
<td>0.05</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• **Future work**
  • LP relaxation
    • Large-scale MIPs
  • Cut generation
    • Find valid and useful cuts
    • Generate cuts
ACKNOWLEDGEMENT

We appreciate the funding support from Argonne LDRD program and the Advanced Grid Modeling program (AGM) under DOE Office of Electricity
THANK YOU!

Contact
Feng Qiu
Principal Computational Scientist & Group Manager
Email: fqiu@anl.gov