An asynchronous distributed subgradient algorithm for solving stochastic unit commitment

Anthony Papavasiliou, Ignacio Aravena
Center for Operations Research and Econometrics
Université catholique de Louvain, Belgium
Outline

• Motivation
• Deterministic and stochastic unit commitment
• Lagrange relaxation and the subgradient method
• Synchronous distributed algorithm
• Asynchronous block-coordinate descent method
• Asynchronous distributed algorithm
  • Dual iterations and lower bound computation
  • Primal solution recovery
• Case study: Central Western Europe
  • High performance computing implementation and results
• Conclusions
Motivation

- Continental Europe leading in renewable energy integration, 82 GW of solar PV and 108 GW of wind power
- Renewable energy integration levels will continue to increase worldwide following environmental targets
- Questions:
  - Are current practices for scheduling energy and reserves effective under high renewable integration levels?
  - Are proposed stochastic models solvable in operational time frames?
Deterministic unit commitment

- Schedules production based on renewable infeed forecast, while reserving additional capacity to handle forecast errors

\[
\begin{align*}
\min_{p,r,u,v} & \quad \sum_{g \in G} \left( NLC_g \ 1^T u_g + SUC_g \ 1^T v_g + c_g(p_g) \right) \\
\text{s.t.} & \quad \sum_{g \in G} p_{g,t} \geq \hat{D}_t, \quad \forall t \in T \\
& \quad \sum_{g \in G} r_{g,t} \geq R, \quad \forall t \in T \\
& \quad (p_g, r_g, u_g, v_g) \in D_g^R, \quad \forall g \in G
\end{align*}
\]
Stochastic unit commitment

- Schedules production based on the possible realizations of renewable infeed (explicitly modelling uncertainty)
- Problem size grows proportionally to the number of scenarios

\[
\min_{p,u,v,\pi,w,z} \quad \sum_{s \in S} \pi_s \sum_{g \in G} \left( NLC_g \ 1^T u_{g,s} + SUC_g \ 1^T v_{g,s} + c_g(p_{g,s}) \right) \\
\text{s.t.} \quad \sum_{g \in G} p_{g,s,t} \geq D_{s,t}, \quad \forall t \in T, s \in S \\
(p_{g,s}, u_{g,s}, v_{g,s}) \in D_{g,s}, \quad \forall g \in G, s \in S \\
(w_g, z_g) \in D_{g,wz}, \quad \forall g \in G \\
u_{g,s} = w_g, \quad v_{g,s} = z_g, \quad \forall g \in G, s \in S
\]
Lagrange relaxation

- Stochastic unit commitment can be dualized by associating Lagrange multipliers $\mu, \nu$ to non-anticipativity constrains, leading to the following dual function

$$g(\mu, \nu) = \min_{p, u, v} \quad \sum_{w, z} \sum_{s \in S} \pi_s \sum_{g \in G} \left( (NL\mathbf{c}_g \mathbf{1} + \mu_{g,s})^T u_{g,s} + (SU\mathbf{c}_g \mathbf{1} + \nu_{g,s})^T v_{g,s} + c_g(p_{g,s}) \right) -$$

$$\sum_{g \in G} \left( \left( \sum_{s \in S} \pi_s \mu_{g,s} \right)^T w_g + \left( \sum_{s \in S} \pi_s \nu_{g,s} \right)^T z_g \right)$$

s.t. $\sum_{g \in G} p_{g,s,t} \geq D_{s,t}, \quad \forall t \in T, s \in S$

$(p_{g,s}, u_{g,s}, v_{g,s}) \in D_{g,s}, \quad \forall g \in G, s \in S$

$(w_g, z_g) \in D_{g}^{w,z}, \quad \forall g \in G$
Lagrange relaxation (cont.)

- The dual function \( g(\mathbf{\mu}, \mathbf{\nu}) \) has the following properties:
  - \( g(\mathbf{\mu}, \mathbf{\nu}) \) is concave
  - \( g(\mathbf{\mu}, \mathbf{\nu}) \) is non-differentiable (piecewise linear)
  - \( g(\mathbf{\mu}, \mathbf{\nu}) \) is a lower bound for the optimal solution of the stochastic unit commitment problem
  - \( g(\mathbf{\mu}, \mathbf{\nu}) \) is decomposable, \( g(\mathbf{\mu}, \mathbf{\nu}) = f_0(\mathbf{\mu}, \mathbf{\nu}) + \sum_{s \in S} f_s(\mathbf{\mu}_s, \mathbf{\nu}_s) \), where

\[
f_0(\mathbf{\mu}, \mathbf{\nu}) = \min_{w,z} \sum_{g \in G} \left( \left( - \sum_{s \in S} \pi_s \mathbf{\mu}_{g,s} \right)^T w_g + \left( - \sum_{s \in S} \pi_s \mathbf{\nu}_{g,s} \right)^T z_g \right)
\]

s.t. \((w_g, z_g) \in D_g^{wz}, \quad \forall g \in G\)

and
Lagrange relaxation (cont.)

\[
f_s(\mu_s, \nu_s) = \min_{p,u,v} \pi_s \sum_{g \in G} \left( (NLC_g 1 + \mu_{g,s})^T u_g + (SUC_g 1 + \nu_{g,s})^T v_g + c_g(p_g) \right)
\]

s.t. \[
\sum_{g \in G} p_{g,t} \geq D_{s,t}, \quad \forall t \in T
\]

\[
(p_g, u_g, v_g) \in D_{g,s}, \quad \forall g \in G
\]

- Evaluation of \( g(\mu, \nu) \) is amenable to distributed computing
- Maximizing \( g(\mu, \nu) \) (dual problem) can lead to a tight lower bound on stochastic unit commitment
- Primal recovery heuristics can be used to compute good solutions for stochastic unit commitment, starting from approximate solutions to the dual problem
Solving the dual problem: subgradient method

- Let’s assume that we want to minimize $h(x)$, which is a convex and non-differentiable function.

- Recall: a sub-gradient $\mathbf{d}$ of $h(x)$ at $x_0$ is a vector such that
  \[ \mathbf{d}^T (x - x_0) \leq h(x) - h(x_0) \quad \forall x \]

- Let $x^k$ be the current iterate and $\mathbf{d}^k$ a subgradient of $h(x)$ at $x^k$. A subgradient iteration is performed as:
  \[ x^{k+1} := x^k - \alpha_k \mathbf{d}^k \]
  where $\alpha_k$ is a suitable step size (decreasing, constant, dynamic)
Subgradient method applied to \( g(\mu, \nu) \)

- Minimizing \(-g(\mu, \nu)\) is equivalent to maximize \( g(\mu, \nu) \) (\(-g(\mu, \nu)\) is convex and non-differentiable)
- A subgradient of \(-g(\mu, \nu)\) at \((\mu^k, \nu^k)\) can be computed as

\[
\begin{align*}
\mathbf{d}_{\mu_s}^k &= \pi_s (\mathbf{w}^* - \mathbf{u}^*_s), \quad \forall s \in S \\
\mathbf{d}_{\nu_s}^k &= \pi_s (\mathbf{z}^* - \mathbf{v}^*_s), \quad \forall s \in S
\end{align*}
\]

where \((\mathbf{w}^*, \mathbf{z}^*)\) are obtained from the evaluation of \( f_0(\mu^k, \nu^k) \) and \((\mathbf{u}^*_s, \mathbf{v}^*_s)\) are obtained from the evaluation of \( f_s(\mu^k_s, \nu^k_s) \)

- A subgradient update on \((\mu^k, \nu^k)\) requires the independent evaluation of \( f_0 \) and \( f_s \) \(\rightarrow\) parallelization
Synchronous distributed algorithm for stochastic UC

Let \( k = 0, \mu^0 = 0, \nu^0 = 0, LB = -\infty, UB = +\infty \)

1. Evaluate (in parallel) \( f_0(\mu^k, \nu^k) \) and \( f_s(\mu_s^k, \nu_s^k), \forall s \in S \rightarrow d \)

2. Let \( LB := \max\{LB, f_0(\mu^k, \nu^k) + \sum_s f_s(\mu_s^k, \nu_s^k)\} \)

3. Perform primal recovery (in parallel) and update \( UB \).

4. If \( UB - LB \leq \epsilon \), terminate. Otherwise, continue.

5. Make subgradient update: \((\mu^{k+1}, \nu^{k+1}) := (\mu^k, \nu^k) - \alpha_k d \)

6. Let \( k := k + 1 \) and go to 1

- Two steps of the algorithm can be parallelized
- Note that each iteration includes synchronization points, e.g. 2
Implementation

\[ \mu_{gst}, V_{gst} \]

Dual multiplier update

Second-stage subproblems \( P2_s \)

\[ u_{gst}^*, v_{gst}^* \]

First-stage subproblem \( P1 \)

\[ z_{gt}^*, w_{gt}^* \]

Second-stage feasibility runs \( ED_s \)

Monte Carlo economic dispatch \( ED_c \)
Synchronous distributed algorithm for stochastic UC

• Why are synchronization points undesirable? They can lead to **inefficient** use of distributed computing infrastructure

• How bad can this be for stochastic UC? We have seen instances where the evaluation of $f_s$ for one scenario can take up **75 times** more than the rest!

• For MILP problems, **balanced size does not imply balanced solution times**

Block-coordinate descent method

- Performs updates on subsets of variables (block-coordinate)
- Useful alternative whenever computing the subgradient with respect to a subset of variables is considerably cheaper than computing a full subgradient
- Serial algorithm: maximizing $g(\mu, \nu)$, starting from $k = 1$
  1. Select a scenario $s^k \in S$ (sequentially or randomly)
  2. Evaluate $f_0(\mu^k, \nu^k)$ and $f_s(\mu^{s_k}_s, \nu^{s_k}_s) \rightarrow d_s^k$
  3. Make update: $(\mu^{s_k}_{s+1}, \nu^{s_k}_{s+1}) := (\mu^{s_k}_s, \nu^{s_k}_s) - \alpha_k d_s^k$
     
     $(\mu^{s+1}_s, \nu^{s+1}_s) := (\mu^s_s, \nu^s_s) \forall r \in S - \{s^k\}$
  4. Let $k := k + 1$ and go to 1.
Asynchronous distributed block-coordinate descent method

- Assume you have as many processors as block-coordinates. Ideally, at all times, each processor should be performing a block-coordinate update.
- Associate each block-coordinate $s$ with a processor $m_s$.
- Asynchronous algorithm:

At each processor $m_s$, starting from $k(s) = 1$

1. Get current $(\mu_r^{k(s)}, \nu_r^{k(s)}) := (\mu_r^{k(r)}, \nu_r^{k(r)}), \forall r \in S - \{s\}$

2. Evaluate $f_0(\mu_s^{k(s)}, \nu_s^{k(s)})$ and $f_s(\mu_s^{k(s)}, \nu_s^{k(s)}) \rightarrow d_s$

3. Make update: $(\mu_s^{k(s)+1}, \nu_s^{k(s)+1}) := (\mu_s^{k(s)}, \nu_s^{k(s)}) - \alpha_k^s d_s$

4. Let $k(s) := k(s) + 1$ and go to 1.
Asynchronous distributed block-coordinate descent method

- Asynchronous algorithm fully exploits distributed computing infrastructure, but...
- Remark 1: *Get current*...
  Requires communications between processors. Expensive if the number of processors is large.
- Remark 2: \( \alpha_k^s \) (\( k \) is the global iterator count, i.e. \( k := \sum_{s \in S} k(s) \))
  Some block-coordinates will be updated less frequently than others. Step sizes need to compensate for this difference: larger step sizes for less frequently updated block-coordinates.
- Remark 3: Value of \( g(\mu^{k(s)}, \nu^{k(s)}) \)?
Asynchronous distributed algorithm for stochastic UC
Asynchronous algorithm: Dual process
Asynchronous algorithm: Dual process

1. Evaluate \( f^{k(s)}_s := f_s \left( \mu^{k(s)}_s, \nu^{k(s)}_s \right) \rightarrow u^* \)

2. Gather \( \mu^{k(r)-1}_r, \nu^{k(r)-1}_r, f^{k(r)-1}_r \) \( \forall r \in S - \{s\} \), from Coordinator

   Let \( \overline{\mu} := \left( \mu^{k(s_1)-1}_{s_1}, \ldots, \mu^{k(s)}_s, \ldots, \mu^{k(r|S|-1)}_{r|S|} \right) \)

   \( \overline{\nu} := \left( \nu^{k(s_1)-1}_{s_1}, \ldots, \nu^{k(s)}_s, \ldots, \nu^{k(r|S|-1)}_{r|S|} \right) \)

3. Evaluate \( f_0(\overline{\mu}, \overline{\nu}) \xrightarrow{(using \ u^*)} \overline{d}_s \)

4. Compute \( LB^{k(s)} := g(\overline{\mu}, \overline{\nu}) = f_0(\overline{\mu}, \overline{\nu}) + f^{k(s)}_s + \sum_{r \neq s} f^{k(r)-1}_r \)

5. Make update \( \left( \mu^{k(s)+1}_s, \nu^{k(s)+1}_s \right) := \left( \mu^{k(s)}_s, \nu^{k(s)}_s \right) - \alpha_k \overline{d}_s \)

6. Let \( k(s) := k(s) + 1 \)

7. Send \( \mu^{k(s)}_s, \nu^{k(s)}_s, f^{k(s)-1}_s, LB^{k(s)-1}, u^* \) to Coordinator and go to 1.
Asynchronous algorithm: Primal recovery
Asynchronous algorithm: Primal recovery

1. Receive assignment $u^l, s_i$ from Coordinator

2. Determine $v^l$ and solve 2nd stage problem for $s_i$,

$$p^* := \arg \min_p \sum_{g \in G} c_g(p_g)$$

s.t. $$\sum_{g \in G} p_{g,t} \geq D_{s_i,t}, \quad \forall t \in T$$

$$\left( p_g, u^l_g, v^l_g \right) \in D_{g,s_i}, \quad \forall g \in G$$

3. Compute $UB^l(s_i) := \pi_{s_i} \sum_{g \in G} \left( NLC_g 1^T u^l_g + SUC_g 1^T v^l_g + c_g(p^*_g) \right)$

4. Send $UB^l(s_i)$ to Coordinator and wait for next assignment
Asynchronous algorithm: Coordinator
Asynchronous algorithm: Coordinator

Let: $k(s) = 1$, $\mu_s^0 = \nu_s^0 = \mu_s^1 = \nu_s^1 = 0$, $\tilde{f}_s^0 = -\infty, \forall s \in S$

$\mathcal{L}_u = \emptyset$, $LB = -\infty$, $UB = +\infty$

1. **Launch Dual process** $s$ with $(\mu_s^1, \nu_s^1) \forall s \in S$

2. **Assign** as many queued $(u^l, s_i)$ to Primal recovery processes as possible, remove them from $\mathcal{L}_u$ and wait for next event

3. If receive $\mu_s^{k(s)}, \nu_s^{k(s)}, \tilde{f}_s^{k(s)-1}, LB^{k(s)-1}, u^*$ then,
   
   i. Store $\mu_s^{k(s)}, \nu_s^{k(s)}, \tilde{f}_s^{k(s)-1}$ and add $\{(u^*, r) \forall r \in S\}$ to $\mathcal{L}_u$

   ii. Update lower bound $LB := \max\{LB, LB^{k(s)-1}\}$ and go to 4.

Otherwise (received $UB^l(s)$),

i. If $UB^l(s)$ is available $\forall s \in S$ then, update upper bound $UB := \min\{UB, \sum_{s \in S} UB^l(s)\}$ and go to 4. Otherwise, go to 2.

4. If $UB - LB \leq \epsilon$, terminate. Otherwise, go to 2.
Asynchronous algorithm: lower bound initialization

• Recall: evaluation of $f_s$ requires to solve a UC problem,

$$
    f_s(\mu_s, \nu_s) = \min_{p,u,v} \pi_s \sum_{g \in G} \left( (NLC_g 1 + \mu_{g,s})^T u_g + (SUC_g 1 + \nu_{g,s})^T v_g + c_g(p_g) \right)
$$

  s.t. \quad \sum_{g \in G} p_{g,t} \geq D_{s,t}, \quad \forall t \in T

  \quad (p_g, u_g, v_g) \in D_{g,s}, \quad \forall g \in G

• Evaluation of $f_s$ can take up to 75 times more for certain scenarios than others → one scenario can delay the computation of the first “full” lower bound

• We can use a relaxation of UC to compute a fast lower bound during the first iteration ($k(s) = 1 \forall s \in S$)

  • Linear relaxation of UC

  • Sequence of OPF problems
Case study: Central Western European system

CWE grid model of Hutcheon and Bialek (2013): 7 countries, 679 nodes, 1073 lines
Case study: Central Western European system

- 656 thermal generators: 85 GW nuclear, 40 GW CHP, 99 GW slow, 14 GW fast and 10 GW aggregated
- 47.3 GW of solar PV power and 51.2 GW of wind power
- Multi-area renewable production and demand with 15’ time resolution
Implementation details

- Asynchronous algorithm implemented in Mosel (using *mmjobs* to manage communications) calling on Xpress
- Lawrence Livermore National Laboratory Sierra cluster: 23,328 cores on 1,944 nodes, 2.8 Ghz, 24 GB/node
- Using 10 nodes:
  - 5 nodes for *Dual processes* (6 per node)
  - 5 nodes to *Primal recovery* (12 per node)
Instances

- Comparing deterministic UC (Determ2R and Determ3R) with stochastic UC (Stoch30, Stoch60 and Stoch120) for an industrial scale model.
- 8 instances (day types) per model: 2 per season, weekday/weekend

<table>
<thead>
<tr>
<th>Model</th>
<th>Scenarios</th>
<th>Variables</th>
<th>Constraints</th>
<th>Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determ2R</td>
<td>1</td>
<td>570.432</td>
<td>655.784</td>
<td>9.552</td>
</tr>
<tr>
<td>Determ3R</td>
<td>1</td>
<td>636.288</td>
<td>719.213</td>
<td>9.552</td>
</tr>
<tr>
<td>Stoch30</td>
<td>30</td>
<td>13.334.400</td>
<td>16.182.180</td>
<td>293.088</td>
</tr>
<tr>
<td>Stoch60</td>
<td>60</td>
<td>26.668.800</td>
<td>32.364.360</td>
<td>579.648</td>
</tr>
<tr>
<td>Stoch120</td>
<td>120</td>
<td>53.337.600</td>
<td>64.728.720</td>
<td>1.152.768</td>
</tr>
</tbody>
</table>
## Running times

Solution statistics over 8 instances.

<table>
<thead>
<tr>
<th>Model</th>
<th>Nodes</th>
<th>Init.</th>
<th>Running time [hours]</th>
<th>Worst final gap [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determ2R</td>
<td>1</td>
<td></td>
<td>1.9 (0.6 – 4.2)</td>
<td>0.95</td>
</tr>
<tr>
<td>Determ3R</td>
<td>1</td>
<td></td>
<td>9.4* (6.3 – 10.0)</td>
<td>4.91</td>
</tr>
<tr>
<td>Stoch30</td>
<td>10</td>
<td>LP</td>
<td>1.1 (0.7 – 2.2)</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>OPF</td>
<td>0.8 (0.3 – 1.8)</td>
<td>1.00</td>
</tr>
<tr>
<td>Stoch60</td>
<td>10</td>
<td>LP</td>
<td>3.2 (1.1 – 8.4)</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>OPF</td>
<td>1.5 (0.6 – 4.7)</td>
<td>0.97</td>
</tr>
<tr>
<td>Stoch120</td>
<td>10</td>
<td>LP</td>
<td>6.1* (1.6 – 10.0)</td>
<td>1.68</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>OPF</td>
<td>3.0* (0.6 – 10.0)</td>
<td>1.07</td>
</tr>
</tbody>
</table>

Termination criteria: 1% optimality or 10 hours wall-time.

* Average running time computed considering models hitting wall-time.
Running times: optimality vs. wall-time

Solution statistics over 8 instances.

<table>
<thead>
<tr>
<th>Model</th>
<th>Init.</th>
<th>Worst gap [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 hour</td>
</tr>
<tr>
<td>Stoch30</td>
<td>LP</td>
<td>7,59</td>
</tr>
<tr>
<td></td>
<td>OPF</td>
<td>1,90</td>
</tr>
<tr>
<td>Stoch60</td>
<td>LP</td>
<td>23,00</td>
</tr>
<tr>
<td></td>
<td>OPF</td>
<td>4,60</td>
</tr>
<tr>
<td>Stoch120</td>
<td>LP</td>
<td>70,39</td>
</tr>
<tr>
<td></td>
<td>OPF</td>
<td>46,69</td>
</tr>
</tbody>
</table>

- Lower bound initialization using sequential OPF observed to be very effective, sometimes avoiding to evaluate $f_s$ for hard scenarios
- Asynchronous stochastic UC algorithm capable of achieving acceptable optimality gaps within operational time frames
Room for improvement: evaluation of primal candidates

Bounds and primal candidates.

Stoch120 (OPF), summer weekday (worst case).

- Valuable computational resources spent in **detailed** evaluation of sub-optimal candidates.
- Future research: pre-screening and pruning of primal candidates
Conclusions and perspectives

- Asynchronous algorithms and high performance computing have the potential to solve stochastic UC within the same time frame required to solve deterministic UC using a state-of-the-art MILP solver.
- Sequential OPF initialization provides fast lower bounds, significantly reducing running times.
- Obtaining good primal candidates from first iterations drastically accelerates the convergence of the algorithm.
- Future extensions:
  - Pruning and scoring primal candidates
  - Dynamical queue management for dual and primal processes
  - Multi-stage stochastic UC
Thank you.

Contact:
Anthony Papavasilou, anthony.papavasilou@uclouvain.be
Ignacio Aravena, ignacio.aravena@uclouvain.be