Optimization Methods for the Unit Commitment Problem

Jim Ostrowski

Industrial and Systems Engineering
University of Tennessee jostrows@utk.edu

The Unit Commitment Problem

- The Unit Commitment (UC) problem is a large scale MINLP that finds a low-cost generating schedule for power generators.
- These problems have quadratic objective functions, and transmission constraints can be highly nonlinear. We are going to ignore transmission in this talk!
- These problems are typically solved as mixed integer programs.

Mixed Integer Linear Program

- A *mixed integer linear program* is defined as:

$$
\begin{aligned}
& \max\ cx + hy && && (1)\\
& \text{s.t. } Ax + Gy \le b && && (2)\\
& \qquad x \ge 0 && \text{integral} && (3)\\
& \qquad y \ge 0 && && (4)
\end{aligned}
$$

  where G is an $m \times p$ matrix and y is a p-vector.
- We call $S := \{(x, y) \in \mathbb{Z}^n_+ \times \mathbb{R}^p_+ : Ax + Gy \le b\}$ of feasible solutions is called a *mixed integer linear set*.
- A *mixed* $0/1$ *set* just restricts the x variables to be either 0 or 1.
- The linear relaxation is given by allowing x to take continuous values:

$$P_0 := \{(x, y) \in \mathbb{R}^n_+ \times \mathbb{R}^p_+ : Ax + Gy \le b\}$$

Methods For Solving Integer Programs

- More notation:

$$\text{MILP: } \max\{cx + hy \ : (x,y) \in S\}$$

where

$$S := \{(x,y) \in \mathbb{Z}^n_+ \times \mathbb{R}^p_+ : Ax + Gy \le b\}$$

Let (x^*, y^*) be the optimal solution to MILP and let the objective value be z^*

- Let P_0 be the linear relaxation of S. Let (x^0, y^0) be the optimal solution (with solution value z^0) to:

$$\max\{cx + hy \ : (x,y) \in P_0\}$$

- Since $S \subseteq P_0$, we know that $z^* \le z_0$, so the linear relaxation gives us an upper bound.

What if x_0 is fractional? Branch and Bound

- Suppose you solve the LP relaxation and the x solution is fractional? Say x_j^0 is fractional (say $f := x_j^0$). Define sets:

$$S_1 := S \cap \{(x, y) : x_j \leq \lfloor f \rfloor\}, \quad S_2 := S \cap \{(x, y) : x_j \geq \lceil f \rceil\}$$

- Every integer solution is in one of S_1 or S_2. So, the best solution to:

MILP1: $\max\{cx+hy \ : (x, y) \in S_1\}$ MILP2: $\max\{cx+hy \ : (x, y) \in S_2\}$

is equal to that of MILP.

Branch and Bound

- Similar to S, create:

$$P_1 := P_0 \cap \{(x,y) : x \leq \lfloor f \rfloor\}, \quad P_2 := P \cap \{(x,y) : x \geq \lceil f \rceil\}$$

and look at

LP1: $\max\{cx + hy \ : (x,y) \in P_1\}$ LP2: $\max\{cx + hy \ : (x,y) \in P_2\}$

More B&B

- Suppose LP_i is infeasible. Then $S_i \subseteq P_i = \emptyset$. This subproblem is *pruned by infeasibility*
- Let (x^i, y^i) be the optimal solution to LP_i with objective z_i.
  - If x^i is integer, then (x^i, y^i) is the optimal solution to MILP_i. Node i is *pruned by integrality*. Since $S_i \subseteq S$, then $z_i \leq z^*$, so z_i is a lower bound for MILP.
  - If x^i is not integer and z^i is smaller than the best known integer solution, then S_i cannot contain a better solution, we *prune* i *by bound*.
  - If x^i is not integer and z_i is better than the best known lower bound, then S_i may still contain an optimal solution to MILP. Find a fractional component of x^i, $x^i_{j'}$, and let $f = x^i_{j'}$. Define sets:

$$S_{i_1} := S_i \cap \{(x,y) : x_{j'} \leq \lfloor f \rfloor\}, \quad S_{i_2} := S \cap \{(x,y) : x_{j'} \geq \lceil f \rceil\}$$

Drivers of Computational Performance for Integer Programming in General

- Looking at the branch-and-bound algorithm, we can identify 3 key drivers of performance:
  - Quality of upper bound: Better formulations can reduce the LP bound, leading to more pruning.
  - Quality of incumbent solution: The sooner you find good solutions, the quicker you can prune nodes.
  - Number of integer variables: more integer variables can (possibly) lead to more branches and larger trees.

Impact of Optimization on UC

- Integer programming has only gained popularity in the past 10-15 years.
- Before that, Lagrangian relaxation methods were used to find decent solutions to these scheduling problems.
- The switch paid off. MISO schedules more than 1,500 power plants throughout the Midwest and Canada.
- Switching to IP saved them between 2.1 and 3.0 billion dollars from 2007 to 2010
- This won them the prestigious Edelman Award from INFORMS
- Since then, we have gotten *a lot* better at solving these problems!

The Basic Problem

The UC Problem

$$\text{Minimize} \sum_{t \in T} \sum_{j \in J} c^j(p^j_t)$$

subject to

$$\sum_{j \in J} p^j_t \geq D_t, \quad \forall\, t \in T$$

$$p^j \in \Pi^j, \quad \forall j \in J.$$

- $c(p^j_t)$ gives the cost of generator j producing p^j_t units of electricity at time t.
- In every time periods, demand D_t must be met.
- Each generator must work within its physical limits (ramping constraints, minimum shut down times, etc.).

Physical Constraints of Generators

- **Convex Production Costs**
- **Minimum & Maximum Output Levels:** If the generator is on, it must produce between $\underline{\mathrm{P}}$ and $\overline{\mathrm{P}}$ units of power.
- **Ramping Constraints:** Power output cannot change too rapidly over a short period of time.
- **Minimum Up (Down) Time:** When a generator is turned on (off), it must stay on for at least UT (DT) time units.
- **Downtime Dependent Startup Costs:** The cost of turning on a generator is dependent on how long the generator has been off.

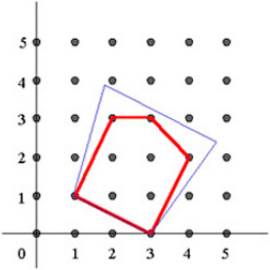
Basic Approach

- A strategy employed by many researchers is to investigate tight formulations for a generic generator, i.e., tight descriptions of Π.
- This work will employ the same tactic.

Main Result:

We will give a tight and compact (convex hull) description of the feasible operating schedule of a generator. Moreover, this description is fairly flexible and can enable a variety of additional physical constraints

Why This Approach?



- Integer Programs are best solved by looking at the linear relaxation.
- The *convex hull* of an IP is the smallest polyhedron containing all of the feasible points.
- The worse the linear relaxation resembles the convex hull, the harder the problem is to solve.

A Brief Outline

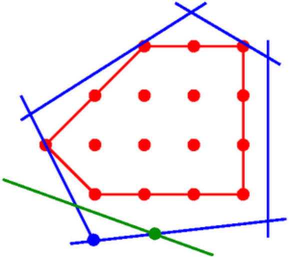
- First, we will discuss some previous work on polyhedral results related to electric generator schedules.
- Then, we will move into more general polyhedral theory and present an extension of Balas' classical theorem.
- Lastly, we will tie the two together to give our compact convex hull result.

Polyhedral Results for Generator Scheduling

"1-binary variable model"

- I can write the feasible region of a generator using two variables per time period.
- Let p_t be the (continuous) variable representing power output.
- Let u_t be the (binary) variable representing if the generator is on/off.
- The convex hull description of this polyhedron is known *if there is no ramping constraint*, but it is *large* (exponential).
- But, a polynomial-time cutting-plane method exists (Lee, Lueng, Margot: 2004).

What is a cutting-plane method?



- If the solution to the linear relaxation is outside of the convex hull, add a linear constraint that will remove it from the relaxation.

3 Binary Variable Model

3-Bin

- Now we use 4 variables per time period:
- Let p_t be the (continuous) variable representing power output.
- Let u_t be the (binary) variable representing if the generator is on at time t.
- Let v_t be the (binary) variable representing if the generator is turned on at time t.
- Let w_t be the (binary) variable representing if the generator is turned off at time t.

- Yes, the additional variable are redundant. But, they allow us to write tight descriptions of the polytope *with no ramping constraints* (Rajan & Takriti: 2005).

Not Quite the Same Thing, but Nice

- A slightly different approach to generator scheduling comes from Frangioni and Gentile, who solve the single unit commitment problem (1UC) in polynomial time using dynamic programming.
  - The 1UC model assumes prices are fixed, then optimizes a single unit's profit.
- The trick: Since the prices are known, it is easy to compute the exact production schedule at times in the interval $[a, b]$ if is is known for sure that the generator turns on at time a and then shuts down at time b (Economic Dispatch Problem).
- There are at most Tc2 many valid turn on/turn off time intervals, so you only need to consider combining the corresponding production schedules, where the only constraint is the minimum downtime constraint.

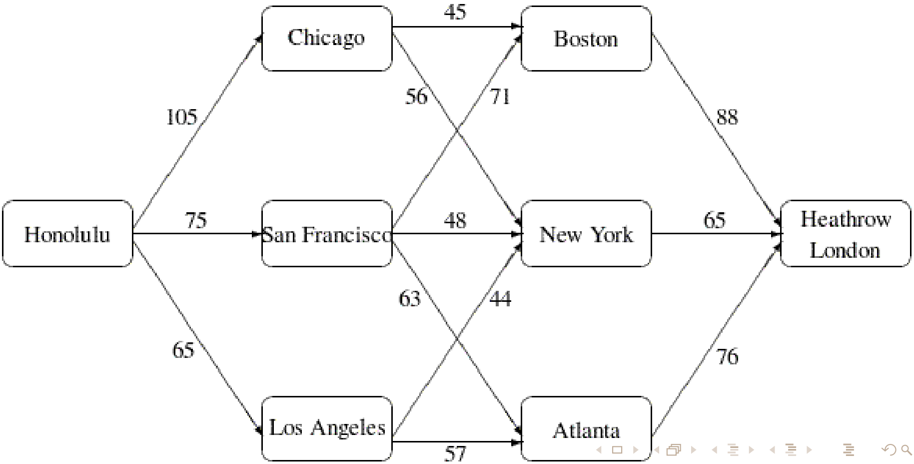
Economic Dispatch Problem

- If it is know that the generator is turned on at a and off at b, the profit during this time period is solved via the *linear program*:

$$
\begin{aligned}
p_i^{[a,b]} &\leq 0 && \forall i < a \text{ and } i > b \\
-p_i^{[a,b]} &\leq -\underline{P} && \forall i \in [a,b] \\
p_i^{[a,b]} &\leq \min(\overline{P}, SU + (i-a)RU, SD + (b-i)RD) && \forall i \in [a,b] \\
p_i^{[a,b]} &\leq p_{i-1}^{[a,b]} \min(RU, SU + (b-i)RD - \underline{P}) && \forall i \in [a+1,b] \\
p_{i-1}^{[a,b]} &\leq p_i^{[a,b]} + \min(RD, SU + (i-a)RU - \underline{P}) && \forall i \in [a+1,b].
\end{aligned}
$$

An Aside: Shortest Path Problem

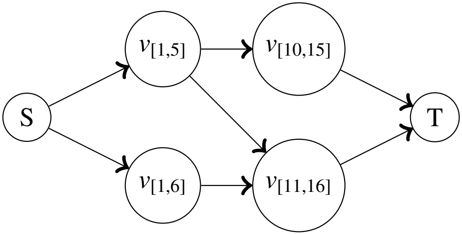
- The shortest path problem attempts to find the shortest path between two given nodes on a graph.
- This can be solved very easily (Dijkstra's Algorithm).

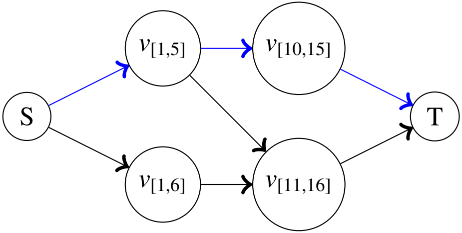


A Dynamic Programming Approach to 1UC

- The 1UC model is solved via a shortest path problem in the following digraph:
- Let s be the source node, t be the sink node.
- Let $v_{[a,b]}$ represent the action of turning on the generator at time a and shutting it off at time b. The cost of going through node $v_{[a,b]}$ is equal to negative the profit from the economic dispatch problem.
- There is an arc leaving (entering) s (t)and entering (leaving) ever other vertex.
- Arc $(v_{[a,b]}, v_{[c,d]})$ exists if $b + \texttt{mindowntime} \leq c$.
- Digraph is acyclic, shortest path is easily found.

An Example: Min Up/Downtime=5





Remarks:

- The dynamic programming approach to 1UC is a fantastic result, but it hasn't been very helpful for multi-generator models.
- Why? The DP only considers Tc2 specific schedules, not all possible production schedules. There hasn't been an obvious way of extending this idea to more general methods.

Fundamental Problem:

Consider any shortest path problem where edges & vertices represent actions represented by polyhedra. How do you efficiently represent the set of feasible solutions? To answer this question, we revisit a classic polyhedral result.

Balas and the Union of Polyhedra

Theorem

Consider m *bounded polyhedra* $P_i := \{x \in \mathbb{R}^n \mid A^i x \leq b^i\}$. *Define* $P = \text{conv}(\cup_{i \in [m]} P_i))$. *Then the polyhedron*

$$Y = \begin{cases} A^i x^i \leq \gamma_i b^i, \ i \in [m] \\ \sum_{i \in [m]} x^i = x \\ \sum_{i \in [m]} \gamma_i = 1 \\ \gamma_i \geq 0, \ i \in [m] \end{cases}$$

provides an extended formulation of P. *So, projecting* Y *back down to the original variables gives you* P.

???

- It is a lot of math, but the basic idea is that it tells you how find the smallest polyhedron that contains 2 or more polyhedron.
- It allows you to model "My solution can satisfy this set of equations or that set of equations."
- In the context of our dynamic program, it allows you to be model the situation where you can pick a solution from 1 (and only 1) Economic Dispatch polyhedron.
- This is not sufficient, since we might want to be on in 2 or more intervals!
- Instead, we need to consider sums of polyhedra.

Weighted Minkowski Sums of Polyhedra

What do I mean by sums?

- Think of polyhedra as bins. I want to construct a solution in $\mathbb{R}^n$ by grabbing vectors in each of the P_is.
- I can assume that I do not grab two or more unique vectors from a single bin.
- The γ terms represent the weight of each of my vectors.
- The set of γ terms are constrained (must be in Γ).
- Investigate $S := \{\sum_{i=1}^{m} \gamma_i P_i \mid (\gamma_1, \ldots, \gamma_m) \in \Gamma\}$

In the context of UC

- Thinking of our dynamic programming problem, this framework allows be to build a schedule by visiting different nodes in the graph.
- If I visit node $v_{[a,b]}$, I can produce in periods $[a, b]$.
- However, I have constraints on how I build my solution! If I visit $v_{[a,b]}$ I cannot visit $v_{[a+1,b]}$!
- This restriction can be modeled by adding constraints on the γ terms (where γ represents if I visit a node or not).

Extended Formulation of Sums

Let Γ be any polyhedron in $\mathbb{R}^m$.

Theorem

Consider m *nonempty polyhedra* $P_i = \{x \in \mathbb{R}^n \mid A^i x \leq b^i\}$, $i \in [m]$. *Consider the polyhedron* $P := \{\sum_{i=1}^m \gamma_i P_i \mid (\gamma_1, \ldots, \gamma_m) \in \Gamma\}$ *and consider another polyhedron* $Y \subseteq \mathbb{R}^{n+nm+m}$ *defined by*

$$Y := \begin{cases} A^i x^i \leq \gamma_i b^i, \; i \in [m] \\ \sum_{i=1}^m x^i = x \\ (\gamma_1, \ldots, \gamma_m) = \gamma \in \Gamma. \end{cases}$$

Then $P = \mathit{proj}_x(Y) := \{x \in \mathbb{R}^n \mid \exists (x^1, \ldots, x^m, \gamma) \in \mathbb{R}^{nm+m}$ *s.t.* $(x, x^1, \ldots, x^m, \gamma) \in Y\}$.

Takeaway

- We can use this theorem and the dynamic programming problem to generate a convex hull description of the "feasible dispatch polyhedra."

Sums of Dispatch Polytope

- Let $\gamma_{[a,b]}$ be multiplier of $D^{[a,b]}$.

Generator Polytope

$$
D \stackrel{\text{def}}{=} \begin{cases}
A^{[a,b]} p^{[a,b]} \leq b^{[a.b]} \gamma^{[a,b]} & \forall [a,b] \in \mathcal{T} \\
\displaystyle\sum_{[a,b] \in \mathcal{T}} p^{[a,b]} = p & \\
\displaystyle\sum_{\{[a,b] \in \mathcal{T} \;\mid\; i \in [a, b + \mathtt{mindowntime}]\}} \gamma^{[a,b]} \leq 1 \; \forall i. \in T & \\
\gamma^{[a,b]} \geq 0 & \\
p^{[a,b]} \in \mathbb{R}^n_+ . &
\end{cases}
$$

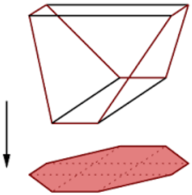
Remarks

- There is a compact & tight formulation for generators. Moreover, this a very general framework. Any additional constraints can be added so long as Γ remains integer and the feasible dispatch problem remains a polytope.
- Allows for:
  - Arbitrary startup costs
  - On-time dependent ramping constraints (to model startup and shutdown trajectories)
  - Multistage Stochastic UC
  - and more!
- Cons of this approach:
  - Tight but large! Tc3 many variables per generator. (Though only T many binomial variables are required).

Lift and Project Cuts

- Using the full model results in a huge linear programming problem. The LP takes too long to solve!
- Another idea is to use the 3-bin model in the formulation but use the convex hull description to generate cuts.
- This is called *lift and project*

Lift and Project: A Picture

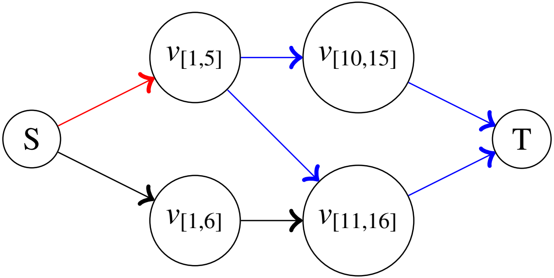


- The basic idea: We know the extended space, we are trying to generate the projected space.
- If we have a point in the projected space, we lift it to the extended space.
- If the lifted point is in the convex hull, than the original point is as well.
- If not, we can easily generate a separating cut in the extended space. Projecting that cut gives us a cut in the projected space.

Identical Generators

- Sometimes there are identical generators in the UC problem.
- Unfortunately, we cannot aggregate them in the 3-bin model.
- However, we can easily account for additional identical generators in the extended formulation!
- Using the dynamic program context, this can be seen by performing multiple walks along the network.

A Picture



How Often Are There Identical Generators?

- Looking at a test case from California ISO (CAISO):
  - Of 610 generators, 465 are unique, giving a reduction of 20%!
  - Performing this aggregation solves problems 40% faster (from about 2 minutes to about 1 minute)!

Results

- The data shows: There are a lot of almost identical generators.
- Aggregating near identical generators can reduce the number of generators from 610 to 315, for a 48% decrease (compared to 24% exactly identical).
- Solving the relaxed problems will be, we hope, much faster!
- The solutions are not always feasible, but they can be easily modified to become feasible.
- These modified solutions tend to be very close to the optimal solution (bases on limited tests, within 0.1%).

Current Work

Current Work

- The proposed methods tend to work well in these “sythentic” test problems.
- Do they work well for real? We are in the process of finding out!
- There are many identical generators in a typical MISO UC instance. We are currently trying to implement this (and more) into their code

Questions for the Future?

- We have a very tight IP formulation for UC. Can we solve it with cutting planes only (no branching)?
  - If so, we can generate more accurate prices.
- Can we use this model in expansion problems?
- What are the economic/market consequences of identical/nearly identical generators?