SAND2012-9177P

COOPR

Modeling and Optimization for the Electric Grid

John D. Siirola

Sandia National Laboratories
Albuquerque, NM USA

With special thanks to

David L. Woodruff
University of California, Davis

William E. Hart
Jean-Paul Watson
Sandia National Laboratories

Zev Freidman
University of Wisconsin

Jack Ingalls
Stanford University

EWO Seminar, Carnegie Mellon University
25 October 2012

NNSA National Nuclear Security Administration

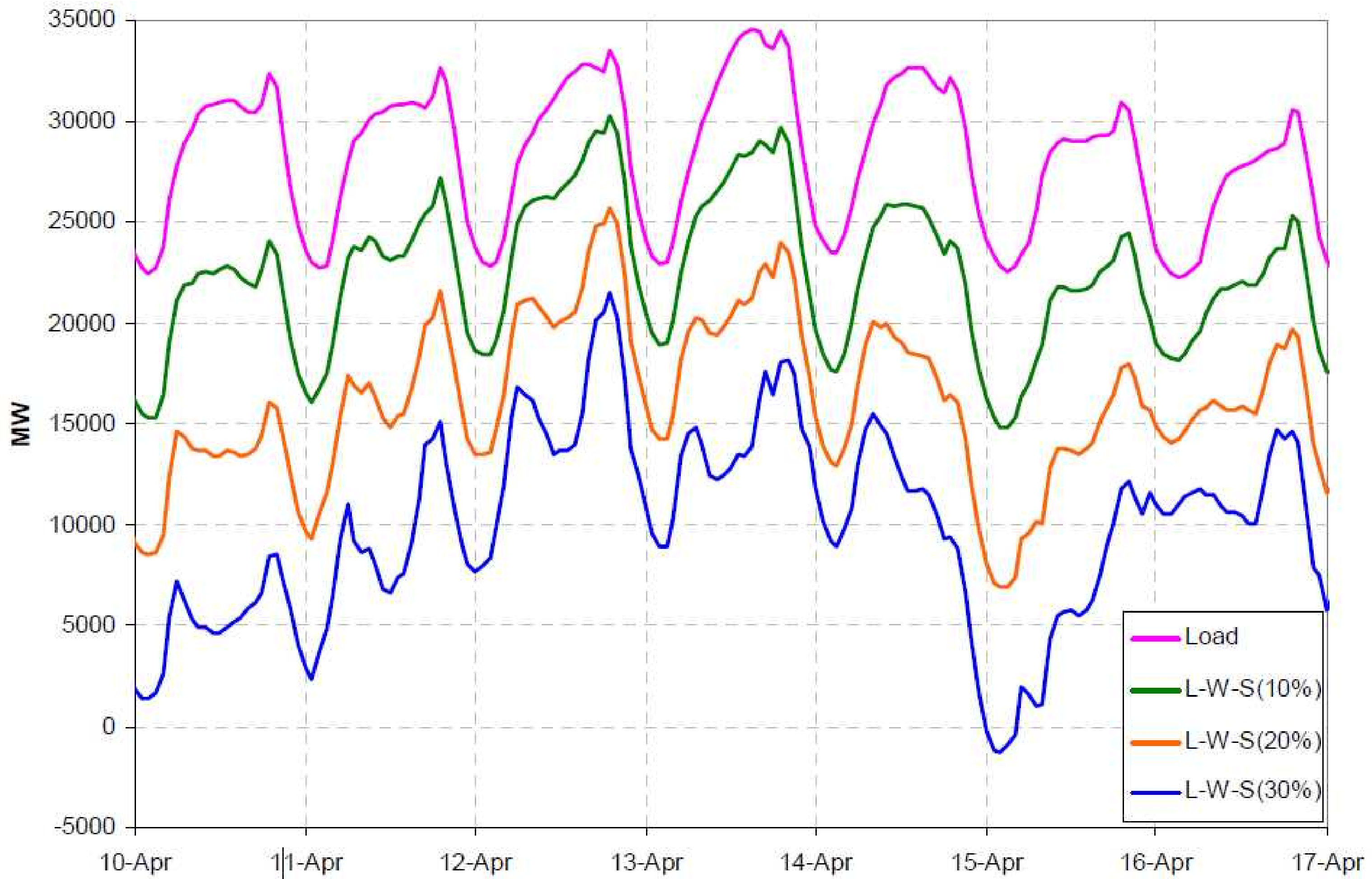
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Integrating renewable energy generation

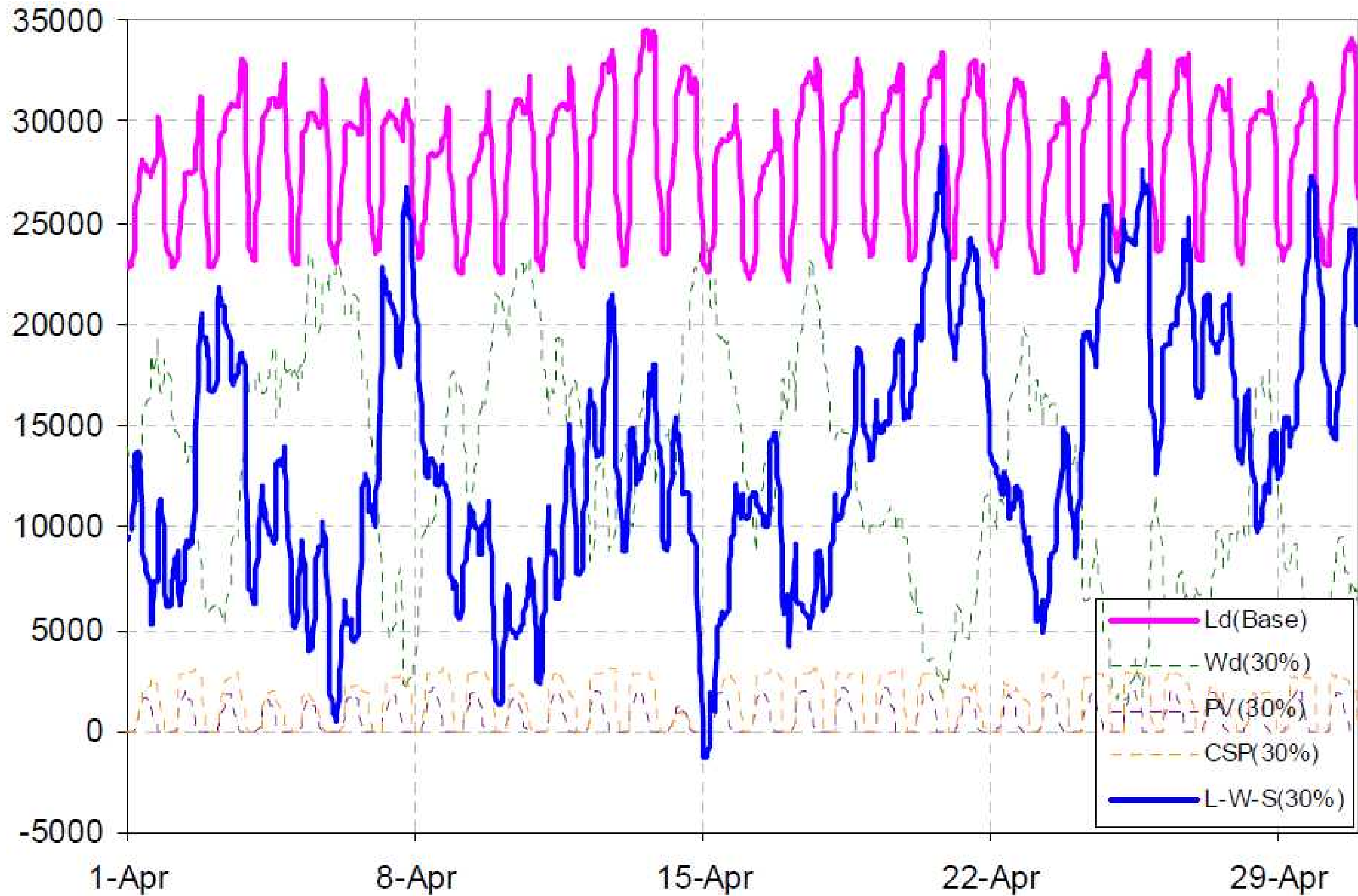
- The grid is managed by
  - Independent System Operators (ISO)
  - Regional Transmission Organizations (RTO)
  - Balancing Authorities
- Operator must balance load and generation at all times
  - Supply demand at lowest possible cost
  - Little to no storage in the grid
  - Unit-specific production ramp limits, startup and shutdown times
  - Disturbances absorbed by (spinning) reserve requirements
- Key challenges:
  - Load variability / forecast errors
  - Variability in *non-dispatchable* (renewable) generation

Dispatch must match *net load*



Plot reproduced from NREL "***Western Wind and Solar Integration Study***"
http://www.nrel.gov/electricity/transmission/western_wind.html

“Loss” of weekly periodicity

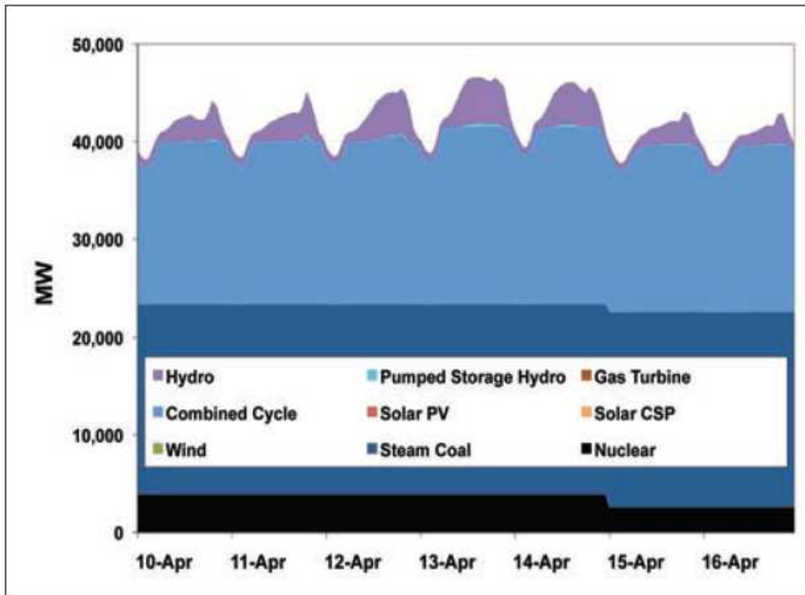


Plot reproduced from NREL “***Western Wind and Solar Integration Study***”
http://www.nrel.gov/electricity/transmission/western_wind.html

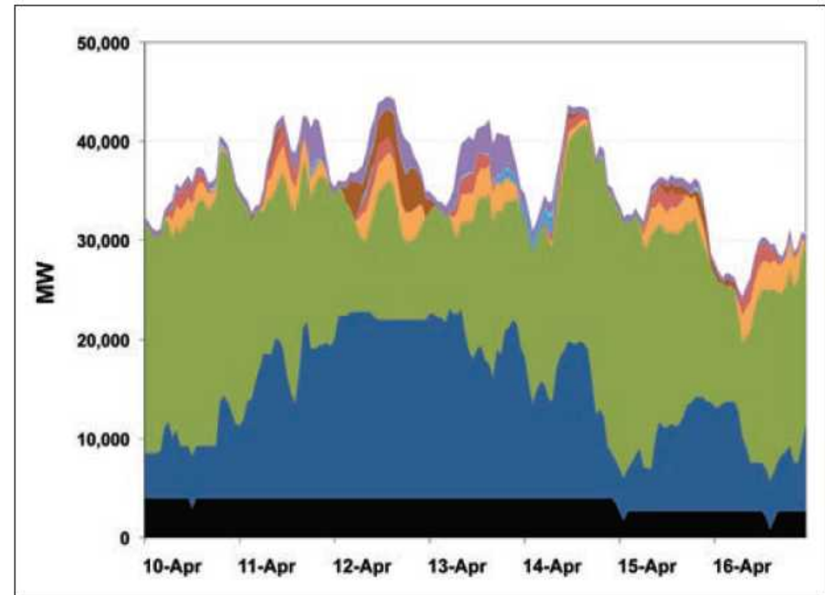
Sandia National Laboratories

Significant impact on "base load" generators

0% Renewable Penetration



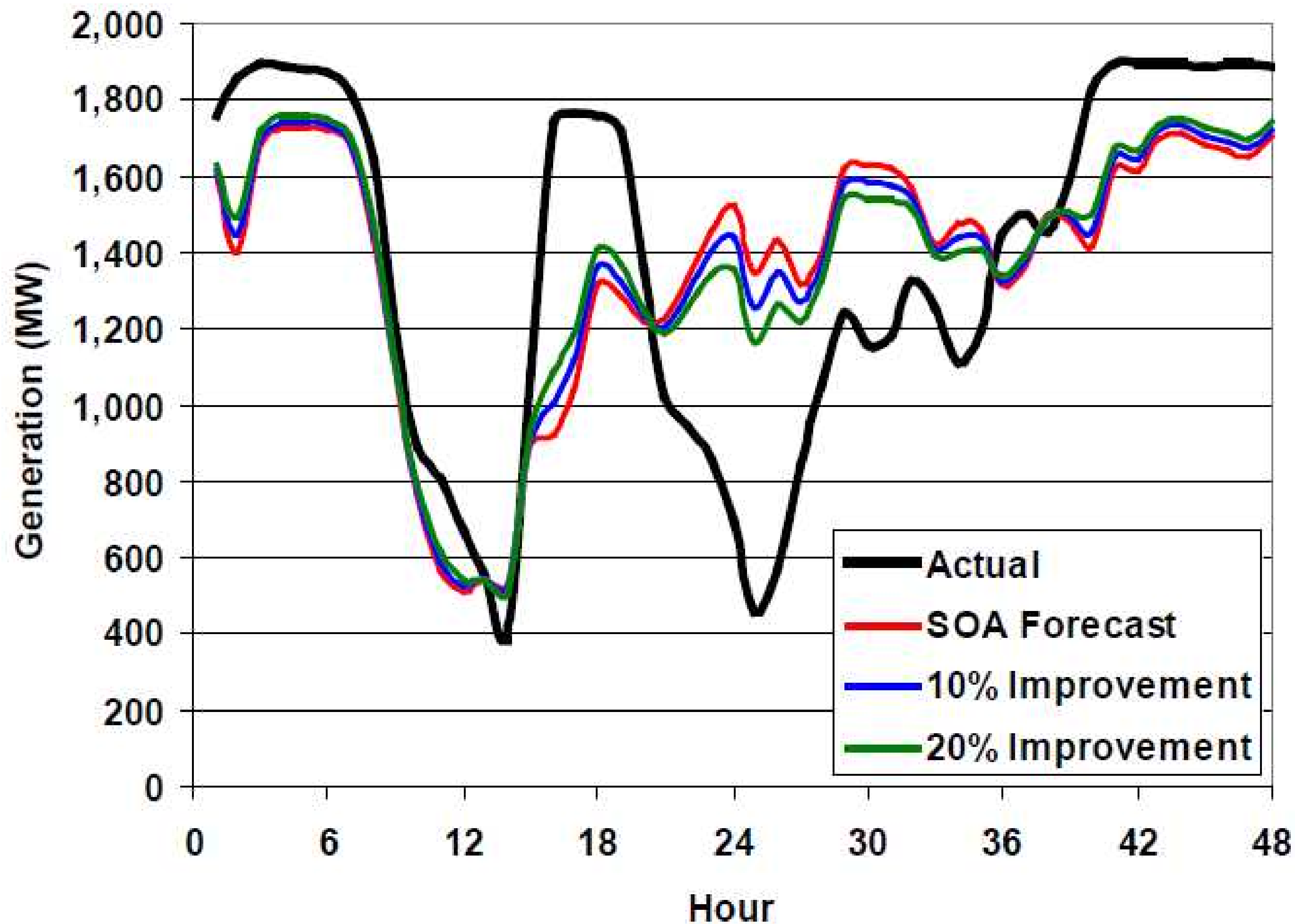
30% Renewable Penetration



Plot reproduced from NREL "***Western Wind and Solar Integration Study***"
http://www.nrel.gov/electricity/transmission/western_wind.html

Sandia National Laboratories

Significant gaps in renewable forecasts



Plot reproduced from NREL "***Value of Wind Power Forecasting***"
http://www.nrel.gov/electricity/transmission/western_wind.html

A word about the example problem…

- (In the US) Sequential markets (run by ISO/RTO):
  - “Unit commitment” (UC) / “Day-ahead Market” (DAM)
    - MIP run ~10 hours before the start of a day
    - Sets on/off state for all generator units hourly for 24 hours
  - “Reliability Unit Commitment” (RUC)
    - MIP run ~8 hours before the start of the day
    - Commits additional generators to meet spinning reserve and reliability (N-1 robustness) requirements
  - “Economic Dispatch” (ED) / “Security-constrained ED” (SCED)
    - “Real-time” markets: LP run hourly / every 5 minutes
    - Set generation levels, prices to meet realized demand

- Problem scale
  - 100’s – 1000’s of buses; 2-3x lines

The Challenge: MP is dense and subtle

$$\text{Minimize}: \sum_t \sum_g \left(c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt}\right) \quad (1)$$

$$\text{S.t.} \quad \theta^{\min} \le \theta_{nct} \le \theta^{\max}, \quad \forall\, n, c, t \quad (2)$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{g0t} = d_{nt},$$
$$\forall\, n, \quad c = 0, \text{ transmission contingency states } c, t \quad (3a)$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{gct} = d_{nt},$$
$$\forall\, n, \text{ generator contingency states } c, t \quad (3b)$$

$$P_{kc}^{\min} N1_{kc} z_{kt} \le P_{kct} \le P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall\, k, c, t \quad (4)$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \ge 0, \quad \forall\, k, c, t \quad (5a)$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \le 0, \quad \forall\, k, c, t \quad (5b)$$

$$P_g^{\min} N1_{gc} u_{gt} \le P_{gct} \le P_g^{\max} N1_{gc} u_{gt}, \quad \forall\, g, c, t \quad (6)$$

$$v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall\, g, t \quad (7)$$

$$\sum_{q=t-UT_g+1}^{t} v_{g,q} \le u_{g,t}, \quad \forall\, g, t \in \{UT_g, \ldots, T\} \quad (8)$$

$$\sum_{q=t-DT_g+1}^{t} w_{g,q} \le 1 - u_{g,t}, \forall\, g, t \in \{DT_g, \ldots, T\} \quad (9)$$

$$P_{g0t} - P_{g0,t-1} \le R_g^{+} u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall\, g, t \quad (10)$$

$$P_{g0,t-1} - P_{g0,t} \le R_g^{-} u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall\, g, t \quad (11)$$

$$P_{gct} - P_{g0,t} \le R_g^{+}, \quad \forall\, g, c, t \quad (12)$$

$$P_{g0,t} N1_{gc} - P_{gct} \le R_g^{-}, \quad \forall\, g, c, t \quad (13)$$

$$0 \le v_{g,t} \le 1, \quad \forall\, g, t \quad (14)$$

$$0 \le w_{g,t} \le 1, \quad \forall\, g, t \quad (15)$$

$$u_{g,t} \in \{0, 1\}, \quad \forall\, g, t \quad (16)$$

Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010

The Challenge: MP is dense and subtle

Minimize : $\sum_t \sum_g (c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt})$

S.t. $\theta^{\min} \leq \theta_{nct} \leq \theta^{\max}, \quad \forall\, n, c, t$

$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{g0t} = d_{nt},$

$\forall\, n, \quad c = 0,$ transmission contingency states c, t

$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{gct} = d_{nt},$

$\forall\, n,$ generator contingency states c, t

$P_{kc}^{\min} N1_{kc} z_{kt} \leq P_{kct} \leq P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall\, k, c, t$

$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \geq 0, \quad \forall\, k, c, t$

$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \leq 0, \quad \forall\, k, c, t$

$P_g^{\min} N1_{gc} u_{gt} \leq P_{gct} \leq P_g^{\max} N1_{gc} u_{gt}, \quad \forall\, g, c, t$

$v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall\, g, t$

$\sum_{q=t-UT_g+1}^{t} v_{g,q} \leq u_{g,t}, \quad \forall\, g, t \in \{UT_g, \ldots, T\}$

$\sum_{q=t-DT_g+1}^{t} w_{g,q} \leq 1 - u_{g,t}, \forall\, g, t \in \{DT_g, \ldots, T\}$

$P_{g0t} - P_{g0,t-1} \leq R_g^{+} u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall\, g, t$

$P_{g0,t-1} - P_{g0,t} \leq R_g^{-} u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall\, g, t$

$P_{gct} - P_{g0,t} \leq R_g^{+}, \quad \forall\, g, c, t$

$P_{g0,t} N1_{gc} - P_{gct} \leq R_g^{-}, \quad \forall\, g, c, t$

$0 \leq v_{g,t} \leq 1, \quad \forall\, g, t$

$0 \leq w_{g,t} \leq 1, \quad \forall\, g, t$

$u_{g,t} \in \{0, 1\}, \quad \forall\, g, t$

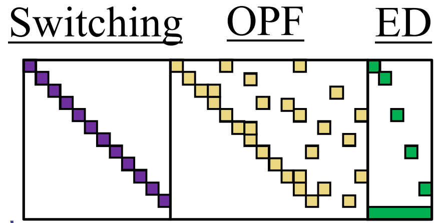
To a first approximation:

- DCOPF
- Economic dispatch
- Unit commitment
- Transmission switching
- N-1 contingency

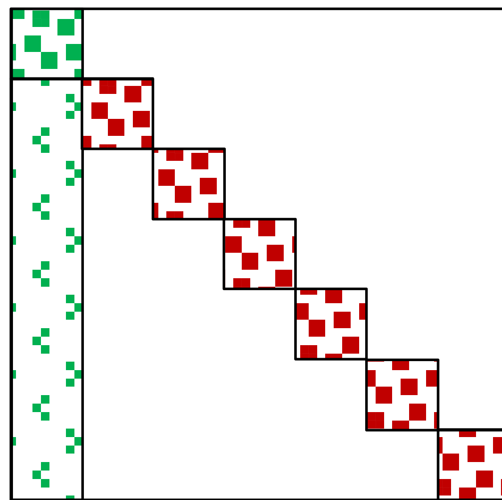
Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010

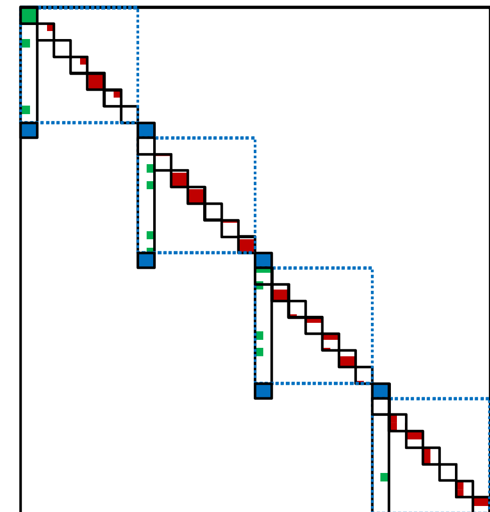
Sandia National Laboratories

(Nonobvious) Inherent structure



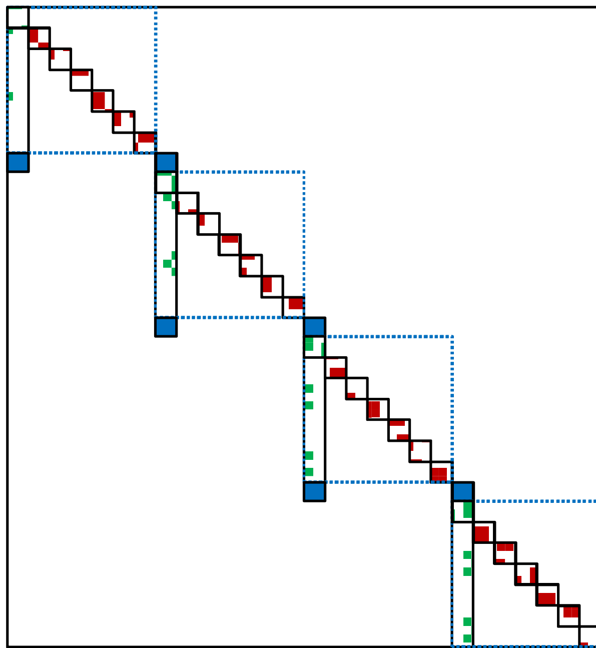
Key feature:
Layered (nested) model complexity

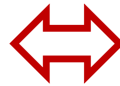


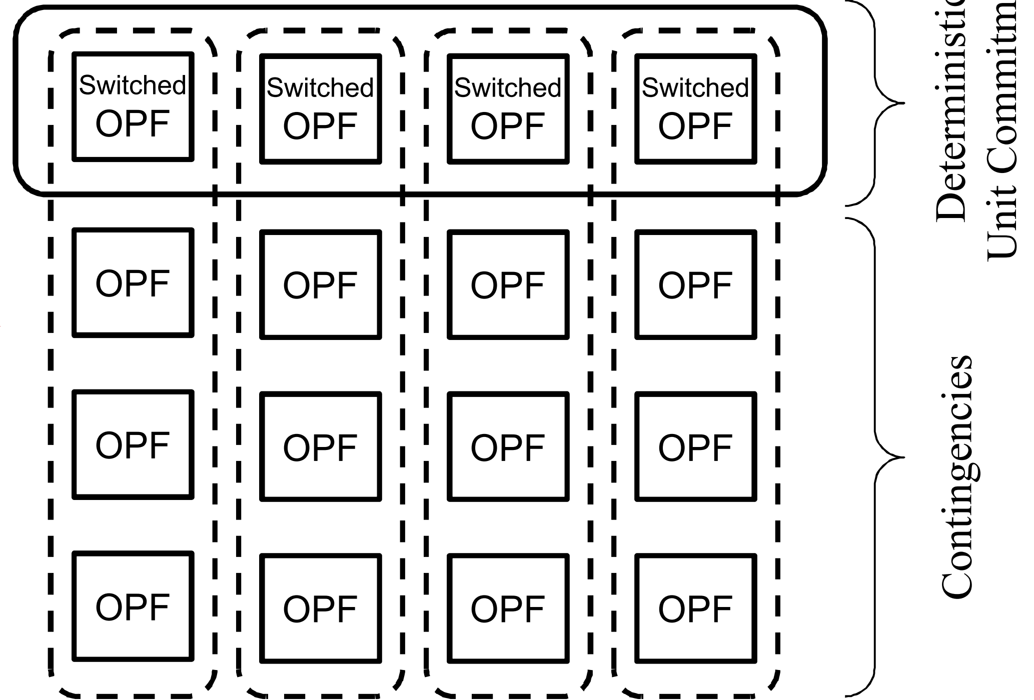


Sandia National Laboratories

This still doesn't *quite* tell the whole story





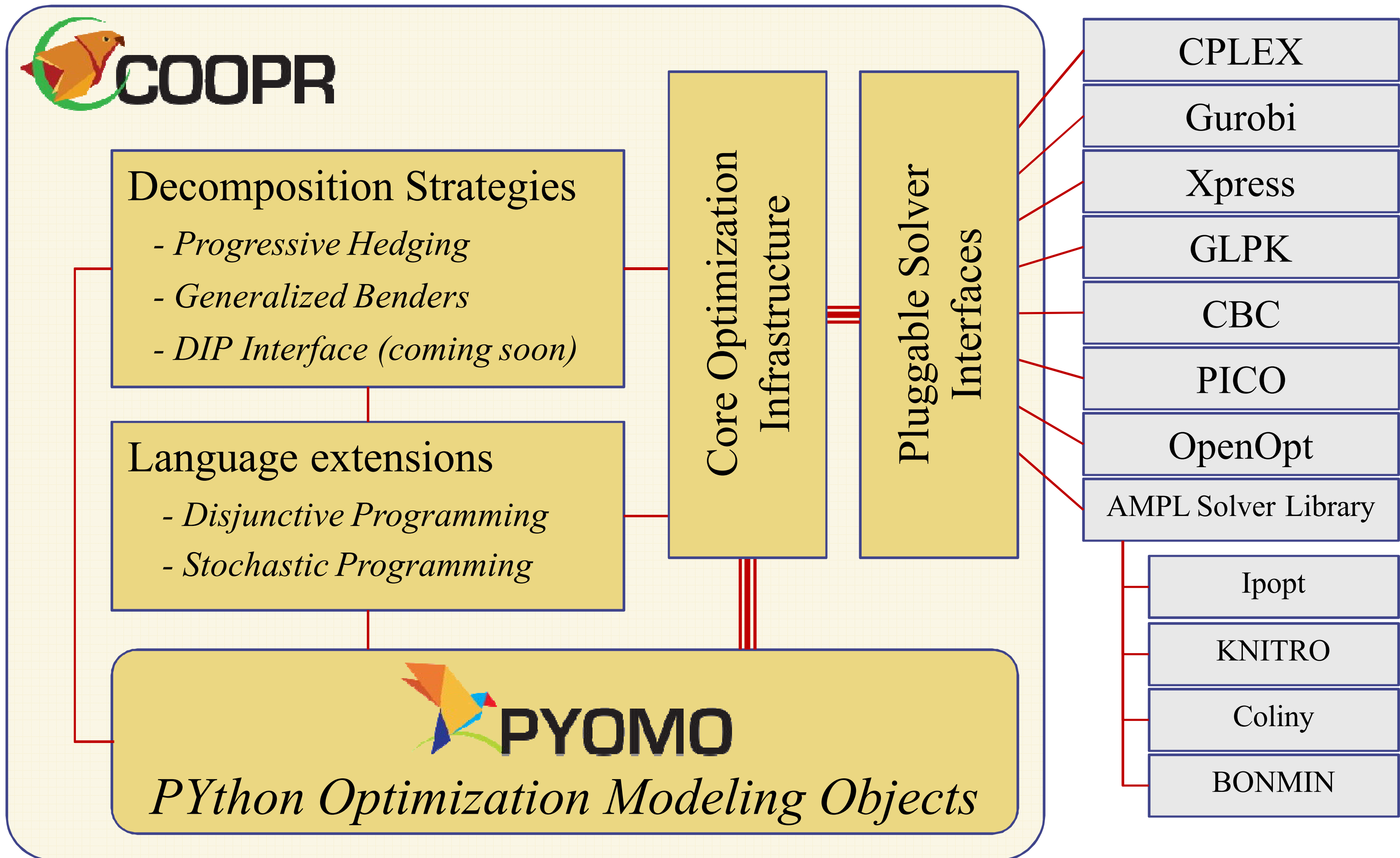


Sandia National Laboratories

Block-oriented modeling

- "Blocks"
  - Collections of model components
    - Var, Param, Set, Constraint, etc.
  - Blocks may be arbitrarily nested
- Why blocks?
  - Support reusable modeling components
  - Express distinctly modeled concepts as distinct objects
  - Manipulate modeled components as distinct entities
  - Explicitly expose model structure (e.g., for decomposition)
- Prior art
  - Ubiquitous in the simulation community
  - Rare in Math Programming environments
    - *Notable exceptions:* ASCEND, JModelica.org
    - This is more than just suffixes!

Coopr: *a COmmon Optimization Python Repository*



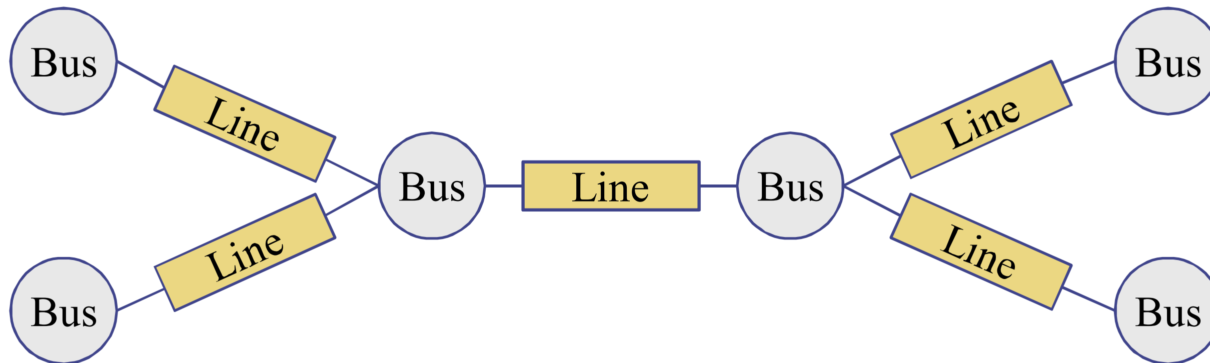
Pyomo overview

- Formulating optimization models natively within Python
  - Provide a natural syntax to describe mathematical models
  - Formulate large models with a concise syntax
  - Separate modeling and data declarations
  - Enable data import and export in commonly used formats
- Highlights:
  - Clean syntax
  - Python scripts provide a flexible context for exploring the structure of Pyomo models
  - Leverage high-quality third-party Python libraries, e.g., SciPy, NumPy, MatPlotLib

```
from coopr.pyomo import *

m = ConcreteModel()

m.x1 = Var()
m.x2 = Var(bounds=(-1,1))
m.x3 = Var(bounds=(1,2))

m.obj = Objective(
   sense = minimize,
   expr = m.x1**2 + (m.x2*m.x3)**4 +
          m.x1*m.x3 + m.x2 +
          m.x2*sin(m.x1+m.x3) )
model = m
```

Rethinking RUC: *a "Tinkertoy" approach*

- Capture connected block structure, e.g., *network flow*



  - Embed physical component models within separate blocks
  - Connect blocks using conceptual interfaces:
    - *Connectors*: groups of named numeric values
      - Constant, Parameter, Variable, Expression
    - "Connect" connectors with simple constraints

Sandia National Laboratories

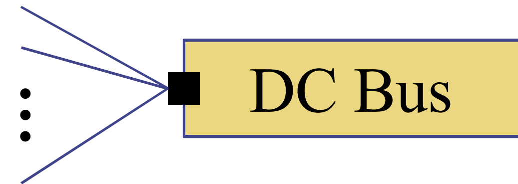
Simple input-output blocks

```
def dc_line_rule(line, id):
    line.B          = Param()
    line.Limit      = Param()

    line.Angle_in   = Var()
    line.Angle_out  = Var()
    line.Power      = Var( bounds= ( -line.Limit, line.Limit ) )

    line.power_flow = Constraint( expr=
        line.Power == line.B*(line.Angle_in - line.Angle_out) )

    line.IN  = Connector( initialize=
        { “Power”: -line.Power, “Angle”: line.Angle_in } )

    line.OUT = Connector( initialize=
        { “Power”:  line.Power, “Angle”: line.Angle_out } )
```


Sandia National Laboratories

Arbitrary inputs: conservation blocks

```
def dc_bus_rule(bus, id):
    bus.D       = Param()

    bus.Angle   = Var()
    bus.Power   = VarList()

    def _power_balance(bus, P):
        return summation(P) == bus.D

    bus.BUS = Connector( initialize={ "Angle": bus.Angle })
    bus.BUS.add( bus.Power, "Power", aggregate=_power_balance )
```



- The *VarList* provides a unique local variable for every connection.
- The *aggregation* rule is called after expanding the connections

Sandia National Laboratories

General power flow model

```
from power_flow import \
        dc_line_rule as line_rule, \
        dc_bus_rule as bus_rule, \
        dc_generator_rule as generator_rule

model.BUSES      = Set()
model.LINES      = Set()
model.GENERATORS = Set()

model.links      = Param( model.LINES, ['IN', 'OUT'] )
model.bus        = Block( model.BUSES,      rule=bus_rule )
model.line       = Block( model.LINES,      rule=line_rule )
model.generator  = Block( model.GENERATORS, rule=generator_rule )

def _network(model, l):
   yield model.line[l].IN  == model.bus[ value(model.links[l, 'IN' ]) ].BUS
   yield model.line[l].OUT == model.bus[ value(model.links[l, 'OUT']) ].BUS
   yield ConstraintList.End
model.network = ConstraintList( model.LINES, rule=_network )

def _generator_placement(model, g):
   return model.generator[g].OUT == model.bus[ value(model.generator[g].bus) ].BUS
model.generator_placement = Constraint( model.GENERATORS, rule=_generator_placement )
```

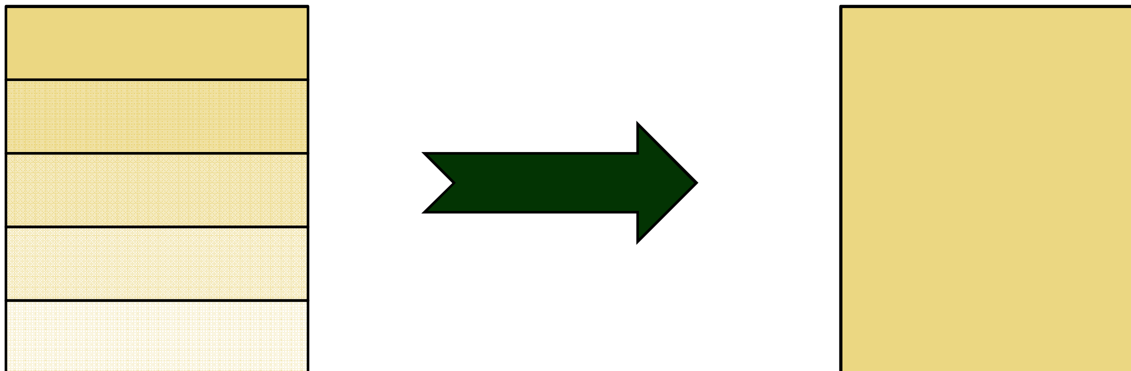
Only domain-specific component

(Note: we have only shown the line rule and not the bus or generator rules)

Sandia National Laboratories

So, what’s really happening?

1) Construct hierarchical model
   - Generate blocks (Variables + Internal constraints)
   - “Connect” blocks by forming constraints over block connectors
2) An automatic *model transformation* “flattens” the model
   - Replicates connector constraints for each variable in connector
   - Generates aggregating constraints
   - (Eliminates redundant variables)



Leveraging components: AC power flow

```
from power_flow import \
        ac_line_rule as line_rule, \
        ac_bus_rule as bus_rule, \
        ac_generator_rule as generator_rule
model.BUSES      = Set()
model.LINES      = Set()
model.GENERATORS = Set()

model.links      = Param( model.LINES, [‘IN’, ‘OUT’] )
model.bus        = Block( model.BUSES,      rule=bus_rule )
model.line       = Block( model.LINES,      rule=line_rule )
model.generator  = Block( model.GENERATORS, rule=generator_rule )

def _network(model, l):
   yield model.line[l].IN  == model.bus[ value(model.links[l, ‘IN’ ]) ].BUS
   yield model.line[l].OUT == model.bus[ value(model.links[l, ‘OUT’]) ].BUS
   yield ConstraintList.End
model.network = ConstraintList( model.LINES, rule=_network )

def _generator_placement(model, g):
   return model.generator[g].OUT == model.bus[ value(model.generator[g].bus) ].BUS
model.generator_placement = Constraint( model.GENERATORS, rule=_generator_placement )
```


Sandia National Laboratories

Manipulating model blocks

- Generalized Disjunctive Programming (GDP)
  - Switching entire blocks on/off through binary variables
- Introduce new Pyomo modeling components:
  - "Disjunct"
    - a new form of model block
  - "Disjunction"
    - a new constraint for enforcing logical XOR over disjunctive sets

$$
\begin{aligned}
\min \quad & \sum_k c_k + f(x) \\
s.t. \quad & g(x) \le 0 \\
& \bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ h_{ik}(x) \le o \\ c_k = \gamma_{ik} \end{bmatrix} \\
& \Omega(Y) = true \\
& Y_{ik} \in \{true, false\}
\end{aligned}
$$

Sandia National Laboratories

Creating a “switchable line”

- Sidebar: we need an “open line” model

```
def dc_line_rule(line, id):
    line.Limit     = Param()

    line.Angle_in  = Var()
    line.Angle_out = Var()
    line.Power     = Var( bounds= ( -line.Limit, line.Limit ) )

    line.power_flow = Constraint( expr= line.Power == 0 )

    line.IN  = Connector( initialize=
        { “Power”: -line.Power, “Angle”: line.Angle_in } )

    line.OUT = Connector( initialize=
        { “Power”:  line.Power, “Angle”: line.Angle_out } )
```

Creating a "switchable line"

```
def switchable_dc_line_rule(line):
   line.CLOSED = Disjunct(rule=closed_line_rule)
   line.OPENED = Disjunct(rule=opened_line_rule)
   line.switch = Disjunction(expr=[line.CLOSED, line.OPENED])

   line.FROM  = Connector()
   line.TO    = Connector()

   def connections_rule(line, id):
      yield line.FROM == line.CLOSED.FROM
      yield line.FROM == line.OPENED.FROM
      yield line.TO == line.CLOSED.TO
      yield line.TO == line.OPENED.TO
      yield ConstraintList.End
   line.connections = ConstraintList(rule=connections_rule)
```


Sandia National Laboratories

Creating a transmission switching model

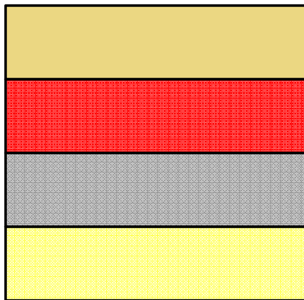
```
from power_flow import  switchable_dc_line_rule as line_rule, \
                        dc_bus_rule as bus_rule, \
                        dc_generator_rule as generator_rule
```

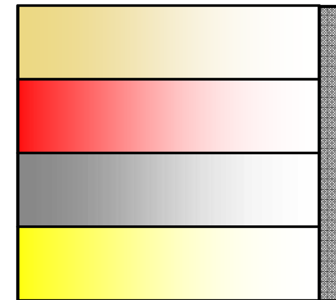
```
model.bus       = Block( model.BUSES, rule=bus_rule )
model.line      = Block( model.LINES, rule=line_rule )
model.generator = Block( model.GENERATORS, rule=generator_rule )

def _network(model, l, end):
   if endpoint == 'IN':
      return model.line[l].IN == model.bus[ value(model.links[l, end]) ].PORT
   else:
      return model.line[l].OUT == model.bus[ value(model.links[l, end]) ].PORT
model.network = Constraint(model.LINES, model.ENDPOINTS, rule=_network)

def _generator_placement(model, g):
   return model.generator[g].OUT == model.bus[ value(model.generator[g].bus) ].PORT
model.generator_placement = Constraint(model.GENERATORS, rule=_generator_placement)
```


Sandia National Laboratories

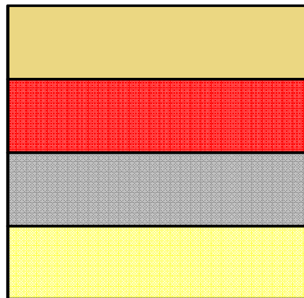
Solving GDP models

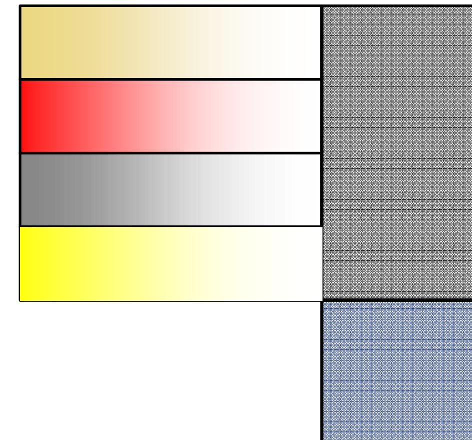
- Automated transformations generate "flat" MI(N)LPs
  - Big-M relaxation





  - Convex hull relaxation





Sandia National Laboratories

Putting it all together: UC + switching + N-1

Network Model

Switchable Transmission Line

Y_i

Transmission Line Power Flow Model

V

$\neg Y_i$

Current Balance (KCL)

Bus model

Y_i^{SU}

Start-Up Model

V

Y_i^{G}

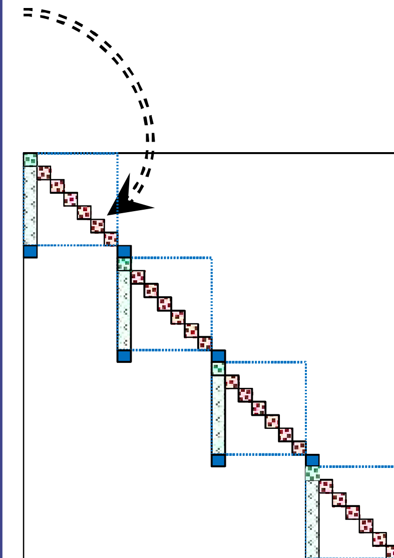
Generation Model

V

$\neg(Y_i^{SU} \mid Y_i^{G})$

Ramp Limits (Y_i)

Switchable Generator



Sandia National Laboratories

The cost of flexibility: transformation time

Instance	Flat	Blocked	Blocked+GDP (line only)	Blocked+GDP (line + generator)
5-bus (24-hour)				
Instantiation	0.88	4.75	5.56	12.2
Connector expansion	–	4.71	5.45	6.1
GDP transformation	–	–	3.11	8.2
Total	2.14	14.1	19.3	34.3
RTS-96 (4-hour)				
Instantiation	39.4	205	231	427
Connector expansion	–	206	224	184
GDP transformation	–	–	139	362
Total	67.6	574	788	1280
RTS-96 (8-hour)				
Instantiation	73.8	386	459	908
Connector expansion	–	389	449	272
GDP transformation	–	–	292	733
Total	148	1130	1610	2600
RTS-96 (12-hour)				
Instantiation	110	571	687	*
Connector expansion	–	657	679	*
GDP transformation	–	–	434	*
Total	219	1740	2470	*

Expanded constraints: *presolve required*

Model	Rows	Columns	Nonzeros	Binaries
Raw generated model				
Flat	20923	6697	53591	312
Blocked	46979	34961	121583	6727
Blocked+GDP (L)	64833	50943	157269	8599
Blocked+GDP (L+G)	88689	54142	211853	15806
After CPLEX presolve				
Flat	10640	4356	34344	285
Blocked	10327	4377	33100	292
Blocked+GDP (L)	10392	4377	33474	292
Blocked+GDP (L+G)	10643	4950	28582	1302

[5-bus, 24-hour test case]

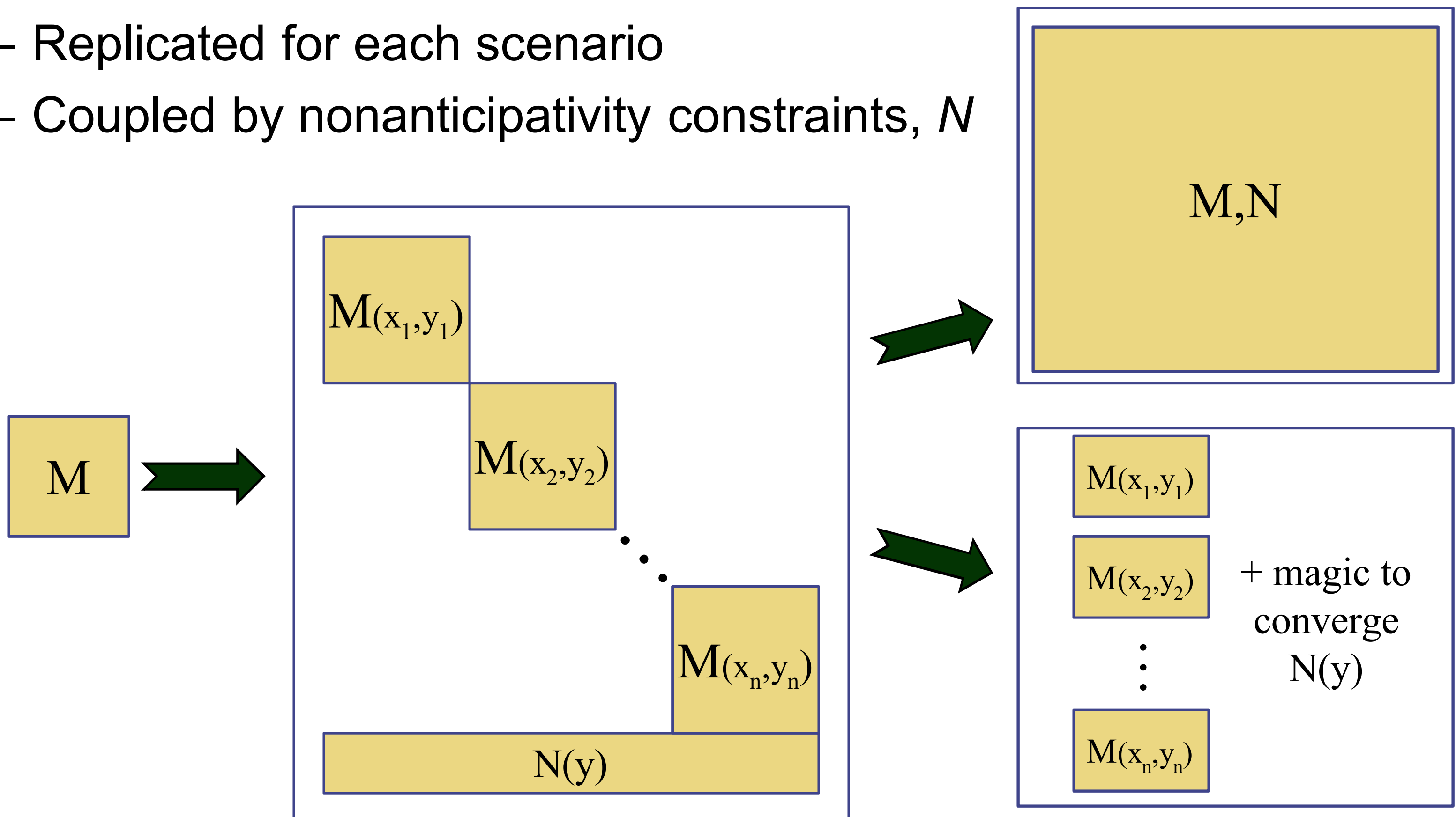
Sandia National Laboratories

The key challenge is *managing uncertainty*

- Historically, absorbed by (spinning) reserves
  - Nominally, 5-10% base demand
  - Approximates the “true” constraint: reliability requirements
  - Absorbing non-dispatchable generation requires significantly higher reserves due to poor forecasts
- Alternative: directly model reliability requirement
  - Robust optimization (e.g., N-1)
  - Stochastic programming + “appropriate” expectation
    - Optimize expectation over a sufficiently large set of scenarios
- Challenges:
  - Multiple stages
  - Integer variables at *any* stage
  - Enormous scenario trees

Forming stochastic programs

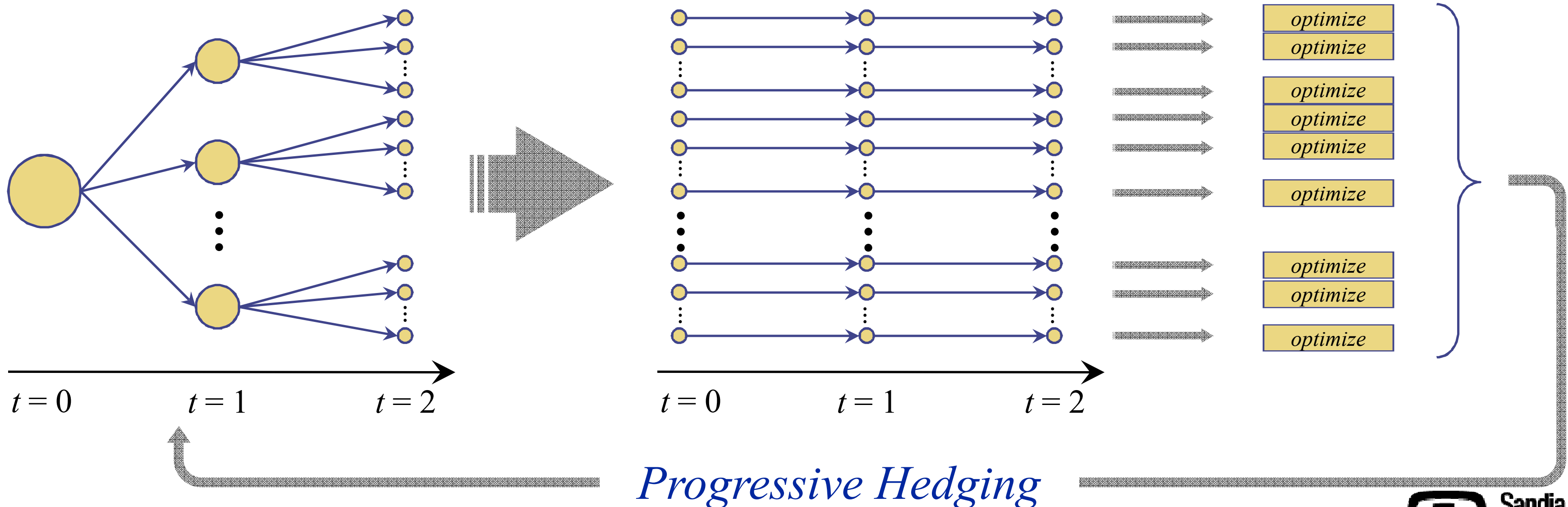
- Exploit "block diagonal" structure
  - Deterministic model, *M*
  - Replicated for each scenario
  - Coupled by nonanticipativity constraints, *N*



Sandia National Laboratories

What about when the extensive form is too big / hard?

- Progressive Hedging (PH) [Rockafellar & Wets]
  - Solve scenarios independently
  - Iteratively converge nonanticipativity constraints
  - PySP: generic implementation of PH
    - Automatic problem construction
    - Numerous tricks / heuristics for handling integer decisions
    - Parallelization on large clusters



Sandia National Laboratories

Applying PH to the *N-1* problem

- CPLEX can solve the EF at the root node (for our test cases)
  - …using heuristics
  - …in 3 days [RTS-96 test case, with 217 contingencies]
- Scenario generation is slightly more complex
  - Choice of decomposition axis: contingency or time?
  - Bundles: nominal case + 1 contingency
- Using PH:
  - The good news:
    - Root nodes solve in < 1 minute
    - This parallelizes "trivially"
  - The bad news
    - Individual scenarios enter the B&B tree
    - … with a relatively large gap (>30%)
  - This is the focus of ongoing research; "so stay tuned"

“Blocks” fundamentally change modeling

- Explicit model blocks
  - Component reuse
  - Implicit transformations when generating model instances
- Generalized Disjunctive Programs
  - Explicit transformations to create standard forms
  - (Solver-specific decomposition)
- Block diagonal models
  - Implicit transformation to create standard forms
  - Solver-specific decompositions (e.g., progressive hedging)
- BUT… a parting shot:
  - The real problem is the ACOPF (nonconvex nonlinear)
  - Actually *solving* that problem is “nontrivial”

Acknowledgements

- Sandia National Laboratories
  - Bill Hart
  - Jean-Paul Watson
  - John Siirola
  - David Hart
  - Tom Brounstein
- University of California, Davis
  - Prof. David L. Woodruff
  - Prof. Roger Wets
- Texas A&M University
  - Prof. Carl D. Laird
  - Daniel Word
  - James Young
  - Gabe Hackebeil
- Carnegie Mellon University
  - Bethany Nicholson
- Texas Tech University
  - Zev Friedman
- Rose Hulman Institute
  - Tim Ekl
- William & Mary
  - Patrick Steele
- North Carolina State
  - Kevin Hunter

Plus our many users, including:

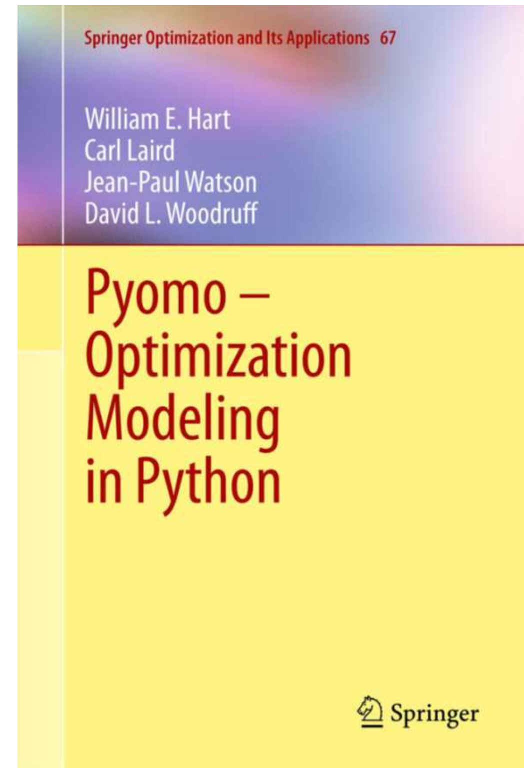
- University of California, Davis
- Texas A&M University
- University of Texas
- Rose-Hulman Institute of Technology
- University of Southern California
- George Mason University
- Iowa State University
- N.C. State University
- University of Washington
- Naval Postgraduate School
- Universidad de Santiago de Chile
- University of Pisa
- Lawrence Livermore National Lab
- Los Alamos National Lab

COIN|OR — Computational Infrastructure for Operations Research

Sandia National Laboratories

For more information…

- Project homepage
  - http://software.sandia.gov/coopr
- “The Book”



- Mathematical Programming Computation papers
  - Pyomo: Modeling and Solving Mathematical Programs in Python (Vol. 3, No. 3, 2011)
  - PySP: Modeling and Solving Stochastic Programs in Python (Vol. 4, No. 2, 2012)